Le logging en **Java**

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

Logging

log

- Traduit souvent en journal
- Fichier contenant des informations sur l'historique de l'utilisation d'une application

Logging

log

- Traduit souvent en journal
- Fichier contenant des informations sur l'historique de l'utilisation d'une application

log, pourquoi ?

- Maintenance
- Sécurité
- SEO
- ...

Logging

Logging (journalisation) ?

Processus permettant :

- la gestion de trace (traçage)
- la gestion de fichiers de log

Logging

Logging (journalisation) ?

Processus permettant :

- la gestion de trace (traçage)
- la gestion de fichiers de log

Logger ?

- Classe ou interface
- Permettant de garder la trace et de l'écrire dans plusieurs supports différents (fichier, console...)

Logging

De l'affichage à la journalisation ?

- La méthode `System.out.print()` permet d'afficher un message ou le contenu d'une variable.
- Elle n'indique pas son emplacement (fichier, ligne, contexte...).
- Elle affiche tout dans la console (pas dans un fichier).
- Le logger propose une solution pour tous ces points.

Logging

Quels frameworks de logging pour **Java** ?

- **Java Logging Technology** : proposé par **Oracle** et contenu dans la `JDK` (`java.util.logging`).
- **Log4J** : projet open source créé par Ceki Gülcü et distribué par **Apache Software initialement**.
- **SLF4J** : projet open source créé aussi par Ceki Gülcü et distribué par **Apache Software initialement**.
- ...

Logging

Avant de commencer

- **Java Logging Technology** inclus dans la **JRE**
- Aucune dépendance à ajouter

Spring

Créons un projet **Maven**

- Aller `File > New > Maven Project`
- Cliquer sur `Next`
- Choisir `maven.archetype.quickstart`
- Remplir les champs
  - `Group Id` avec `org.eclipse`
  - `Artifact Id` avec `cours-logging`
  - `Package` avec `org.eclipse.main`
- Valider

Logging

Dans `App`, commençons par fabriquer une instance `Logger`

```
package org.eclipse.main;

import java.util.logging.Logger;

public class App {

        public static void main(String[] args) {

                Logger logger = Logger.getLogger(App
                   .class.getName());
        }
}
```

Logging

Pour afficher un premier message d'information, on utilise la méthode `info`

```
package org.eclipse.main;

import java.util.logging.Logger;

public class App {

        public static void main(String[] args) {

                Logger logger = Logger.getLogger(App.class.getName());
                logger.info("Notre premier code de traçage");
        }
}
```

Logging

Pour afficher un premier message d'information, on utilise la méthode `info`

```
package org.eclipse.main;

import java.util.logging.Logger;

public class App {

        public static void main(String[] args) {

                Logger logger = Logger.getLogger(App.class.getName());
                logger.info("Notre premier code de traçage");
        }
}
```

Résultat

```
nov. 15, 2020 1:42:39 PM org.eclipse.main.App main
INFO: Notre premier code de traçage
```

Logging

On peut faire la même chose en utilisant la méthode `log` et en indiquant le niveau de log

```
package org.eclipse.main;

import java.util.logging.Logger;
import java.util.logging.Level;

public class App {

        public static void main(String[] args) {

                Logger logger = Logger.getLogger(App.class.getName());
                logger.log(Level.INFO, "Notre premier code de traçage")
                   ;
        }
}
```

Logging

On peut faire la même chose en utilisant la méthode `log` et en indiquant le niveau de log

```
package org.eclipse.main;

import java.util.logging.Logger;
import java.util.logging.Level;

public class App {

        public static void main(String[] args) {

                Logger logger = Logger.getLogger(App.class.getName());
                logger.log(Level.INFO, "Notre premier code de traçage")
                   ;
        }
}
```

Résultat

```
nov. 15, 2020 1:42:39 PM org.eclipse.main.App main
INFO: Notre premier code de traçage
```

Spring

Sept niveaux de journalisation (de plus léger vers le plus sévère)

- `FINEST` : pour les messages de traçage très détaillé.
- `FINER` : pour les messages de traçage assez détaillé.
- `FINE` : pour les messages de traçage.
- `CONFIG` : pour les messages de configuration.
- `INFO` : pour les messages d'information.
- `WARN` : pour les messages indiquant un problème potentiel.
- `SEVERE` : pour les messages indiquant une défaillance grave.

Essayons de tester les différents niveaux de log

```
public class App {

        public static void main(String[] args) {

                Logger logger = Logger.getLogger(App.class.getName());
                logger.log(Level.FINEST, "Finest Message!");
                logger.log(Level.FINER, "Finer Message!");
                logger.log(Level.FINE, "Fine Message!");
                logger.log(Level.CONFIG, "Config Message!");
                logger.log(Level.INFO, "Info Message!");
                logger.log(Level.WARNING, "Warning Message!");
                logger.log(Level.SEVERE, "Severe Message!");
        }
}
```

Essayons de tester les différents niveaux de log

```
public class App {

        public static void main(String[] args) {

                Logger logger = Logger.getLogger(App.class.getName());
                logger.log(Level.FINEST, "Finest Message!");
                logger.log(Level.FINER, "Finer Message!");
                logger.log(Level.FINE, "Fine Message!");
                logger.log(Level.CONFIG, "Config Message!");
                logger.log(Level.INFO, "Info Message!");
                logger.log(Level.WARNING, "Warning Message!");
                logger.log(Level.SEVERE, "Severe Message!");
        }
}
```

Résultat (Seuls les messages d'information ou d'un niveau supérieur sont affichés)

```
nov. 29, 2020 10:46:26 AM org.eclipse.main.App main
INFO: Info Message!
nov. 29, 2020 10:46:26 AM org.eclipse.main.App main
WARNING: Warning Message!
nov. 29, 2020 10:46:26 AM org.eclipse.main.App main
SEVERE: Severe Message!
```

Spring

Explication

- La console par défaut utilisée par un `logger` est configurée au niveau `Info`.
- Pour afficher les messages de niveaux inférieurs, on utilise une nouvelle instance de `ConsoleHandler`.

Créons une nouvelle console avec un niveau de log `FINEST`

```
package org.eclipse.main;

import java.util.logging.ConsoleHandler;
import java.util.logging.Level;
import java.util.logging.Logger;

public class App {

        public static void main(String[] args) {
                Logger logger = Logger.getLogger(App.class.getName());
                logger.setLevel(Level.FINEST);
                ConsoleHandler ch = new ConsoleHandler();
                ch.setLevel(Level.FINEST);
                logger.addHandler(ch);
                logger.log(Level.FINEST, "Finest Message!");
                logger.log(Level.FINER, "Finer Message!");
                logger.log(Level.FINE, "Fine Message!");
                logger.log(Level.CONFIG, "Config Message!");
                logger.log(Level.INFO, "Info Message!");
                logger.log(Level.WARNING, "Warning Message!");
                logger.log(Level.SEVERE, "Severe Message!");
        }
}
```

Logging

Résultat (Les messages `INFO`, `WARNING` et `SEVERE` sont affichés deux fois)

```
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
FINEST: Finest Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
FINER: Finer Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
FINE: Fine Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
CONFIG: Config Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
INFO: Info Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
INFO: Info Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
WARNING: Warning Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
WARNING: Warning Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
SEVERE: Severe Message!
nov. 29, 2020 12:02:37 PM org.eclipse.main.App main
SEVERE: Severe Message!
```

Spring

Explication

- Une instance de `ConsoleHandler` a été créée.
- La console par défaut existe toujours.

Spring

Explication

- Une instance de `ConsoleHandler` a été créée.
- La console par défaut existe toujours.

Solution

Désactiver la console par défaut.

Logging

Pour désactiver la console par défaut

```
public class App {

        public static void main(String[] args) {

                Logger logger = Logger.getLogger(App.class.getName());
                logger.setLevel(Level.FINEST);
                logger.setUseParentHandlers(false);
                ConsoleHandler ch = new ConsoleHandler();
                ch.setLevel(Level.FINEST);
                logger.addHandler(ch);
                logger.log(Level.FINEST, "Finest Message!");
                logger.log(Level.FINER, "Finer Message!");
                logger.log(Level.FINE, "Fine Message!");
                logger.log(Level.CONFIG, "Config Message!");
                logger.log(Level.INFO, "Info Message!");
                logger.log(Level.WARNING, "Warning Message!");
                logger.log(Level.SEVERE, "Severe Message!");
        }
}
```

Logging

Résultat

```
nov. 29, 2020 1:19:51 PM org.eclipse.main.App main
FINEST: Finest Message!
nov. 29, 2020 1:19:51 PM org.eclipse.main.App main
FINER: Finer Message!
nov. 29, 2020 1:19:51 PM org.eclipse.main.App main
FINE: Fine Message!
nov. 29, 2020 1:19:51 PM org.eclipse.main.App main
CONFIG: Config Message!
nov. 29, 2020 1:19:51 PM org.eclipse.main.App main
INFO: Info Message!
nov. 29, 2020 1:19:51 PM org.eclipse.main.App main
WARNING: Warning Message!
nov. 29, 2020 1:19:51 PM org.eclipse.main.App main
SEVERE: Severe Message!
```

Spring

Les sorties disponibles

- `ConsoleHandler`
- `FileHandler`.
- `SocketHandler`
- `MemoryHandler`.

Pour générer des données de traçage dans un fichier XML

```
package org.eclipse.main;

import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;

    public static void main(String[] args) throws SecurityException,
       IOException {

        Logger logger = Logger.getLogger(App.class.getName());
        FileHandler fh = new FileHandler("log.xml");
        logger.addHandler(fh);
        logger.log(Level.INFO, "Notre premier code de traçage");
    }
}
```

Pour générer des données de traçage dans un fichier XML

```
package org.eclipse.main;

import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;

    public static void main(String[] args) throws SecurityException,
       IOException {

        Logger logger = Logger.getLogger(App.class.getName());
        FileHandler fh = new FileHandler("log.xml");
        logger.addHandler(fh);
        logger.log(Level.INFO, "Notre premier code de traçage");
    }
}
```

Résultat

```
nov. 29, 2020 1:31:28 PM org.eclipse.main.App main
INFO: Notre premier code de traçage
```

Logging

Contenu de `log.xml` situé à la racine du projet

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2020-11-29T12:31:28.062477400Z</date>
  <millis>1606653088062</millis>
  <nanos>477400</nanos>
  <sequence>0</sequence>
  <logger>org.eclipse.main.App</logger>
  <level>INFO</level>
  <class>org.eclipse.main.App</class>
  <method>main</method>
  <thread>1</thread>
  <message>Notre premier code de traçage</message>
</record>
</log>
```

Pour générer des données de traçage dans un fichier texte

```
package org.eclipse.main;

import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class App {

    public static void main(String[] args) throws SecurityException,
       IOException {
        Logger logger = Logger.getLogger(App.class.getName());
        FileHandler fh = new FileHandler("log.txt");
        fh.setFormatter(new SimpleFormatter());
        logger.addHandler(fh);
        logger.log(Level.INFO, "Notre premier code de traçage");
    }
}
```

Pour générer des données de traçage dans un fichier texte

```
package org.eclipse.main;

import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class App {

    public static void main(String[] args) throws SecurityException,
       IOException {
         Logger logger = Logger.getLogger(App.class.getName());
         FileHandler fh = new FileHandler("log.txt");
         fh.setFormatter(new SimpleFormatter());
         logger.addHandler(fh);
         logger.log(Level.INFO, "Notre premier code de traçage");
    }
}
```

Résultat (affiché également dans un fichier `log.txt` situé à la racine du projet

```
nov. 29, 2020 1:34:27 PM org.eclipse.main.App main
INFO: Notre premier code de traçage
```

Logging

Commençons par ajouter la dépendance suivante dans `pom.xml`

```
<!-- https://mvnrepository.com/artifact/log4j/log4j -->
<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
</dependency>
```

Logging

Créons un fichier de propriétés `log4j.properties` **dans** `src/main/resources`

```
log4j.rootLogger=DEBUG, stdout

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
#log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH
   :mm:ss} %-5p %c{1}:%L - %m%n
```

Logging

Dans `App`, commençons par fabriquer une instance `Logger`

```
package org.eclipse.main;

import org.apache.log4j.Logger;

public class App {

        public static void main(String[] args)  {

                Logger logger = Logger.getLogger(App.class);

        }
}
```

Logging

Pour afficher quelques messages de journalisation, on peut utilises plusieurs méthodes

```
package org.eclipse.main;

import org.apache.log4j.Logger;

public class App {

        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.warn("john");
                logger.error("wick");
        }
}
```

Logging

Pour afficher quelques messages de journalisation, on peut utilises plusieurs méthodes

```
package org.eclipse.main;

import org.apache.log4j.Logger;

public class App {

        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.warn("john");
                logger.error("wick");
        }
}
```

Résultat

```
2020-11-24 11:43:02 WARN  App:16 - john
2020-11-24 11:43:02 ERROR App:17 - wick
```

Spring

Six niveaux de journalisation (de plus léger vers le plus sévère)

- `TRACE` : pour les messages d'information très détaillés.
- `DEBUG` : pour les messages d'information assez détaillés.
- `INFO` : pour les messages d'information.
- `WARN` : pour les erreurs potentiellement dangereuses.
- `ERROR` : pour les erreurs qui n'empêcheront pas l'application de continuer à s'exécuter.
- `FATAL` : pour les erreurs très graves qui conduiront vraisemblablement l'application à l'arrêt.

Pour afficher quelques messages de journalisation, on peut utilises plusieurs méthodes

```
package org.eclipse.main;

import org.apache.log4j.Logger;

public class App {

        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Pour afficher quelques messages de journalisation, on peut utilises plusieurs méthodes

```
package org.eclipse.main;

import org.apache.log4j.Logger;

public class App {

        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Résultat (`trace message` n'a pas été affiché)

```
2020-11-24 12:12:13 DEBUG App:20 - Debug Message!
2020-11-24 12:12:13 INFO  App:21 - Info Message!
2020-11-24 12:12:13 WARN  App:22 - Warn Message!
2020-11-24 12:12:13 ERROR App:23 - Error Message!
2020-11-24 12:12:13 FATAL App:24 - Fatal Message!
```

Pour afficher tous les messages de journalisation quel que soit le niveau, on indique le niveau le plus faible à partir duquel on commence l'affichage

```
import org.apache.log4j.Level;
import org.apache.log4j.Logger;

public class App {
        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.setLevel(Level.TRACE);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Pour afficher tous les messages de journalisation quel que soit le niveau, on indique le niveau le plus faible à partir duquel on commence l'affichage

```
import org.apache.log4j.Level;
import org.apache.log4j.Logger;

public class App {
        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.setLevel(Level.TRACE);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Résultat

```
2020-11-24 12:22:01 TRACE App:18 - Trace Message!
2020-11-24 12:22:01 DEBUG App:19 - Debug Message!
2020-11-24 12:22:01 INFO  App:20 - Info Message!
2020-11-24 12:22:01 WARN  App:21 - Warn Message!
2020-11-24 12:22:01 ERROR App:22 - Error Message!
2020-11-24 12:22:01 FATAL App:23 - Fatal Message!
```

Ou ainsi

```
import org.apache.log4j.Level;
import org.apache.log4j.Logger;

public class App {
        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.setLevel(Level.ALL);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Ou ainsi

```
import org.apache.log4j.Level;
import org.apache.log4j.Logger;

public class App {
        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.setLevel(Level.ALL);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Résultat

```
2020-11-24 12:22:01 TRACE App:18 - Trace Message!
2020-11-24 12:22:01 DEBUG App:19 - Debug Message!
2020-11-24 12:22:01 INFO  App:20 - Info Message!
2020-11-24 12:22:01 WARN  App:21 - Warn Message!
2020-11-24 12:22:01 ERROR App:22 - Error Message!
2020-11-24 12:22:01 FATAL App:23 - Fatal Message!
```

Logging

Une autre solution consiste à remplacer `DEBUG` par `TRACE` dans la première ligne de `log4j.properties`

```
log4j.rootLogger=TRACE, stdout

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
#log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH
   :mm:ss} %-5p %c{1}:%L - %m%n
```

Logging

Plus besoin d'indiquer le niveau

```
public class App {
        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Logging

Plus besoin d'indiquer le niveau

```
public class App {
        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Résultat

```
2020-11-24 12:22:01 TRACE App:18 - Trace Message!
2020-11-24 12:22:01 DEBUG App:19 - Debug Message!
2020-11-24 12:22:01 INFO  App:20 - Info Message!
2020-11-24 12:22:01 WARN  App:21 - Warn Message!
2020-11-24 12:22:01 ERROR App:22 - Error Message!
2020-11-24 12:22:01 FATAL App:23 - Fatal Message!
```

Logging

Pour générer un fichier de journalisation, on modifie la première ligne de `log4j.properties` et on ajoute une dernière partie décrivant cette nouvelle sortie

```
log4j.rootLogger=TRACE, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
#log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH
   :mm:ss} %-5p %c{1}:%L - %m%n

# Redirect log messages to a log file, support file rolling.
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=log.txt
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:
   mm:ss} %-5p %c{1}:%L - %m%n
```

Logging

Exécutez le `main` précédent et allez vérifiez la création de `log.txt` à la racine du projet

```
public class App {
        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Logging

Exécutez le `main` précédent et allez vérifiez la création de `log.txt` à la racine du projet

```
public class App {
        public static void main(String[] args)  {
                Logger logger = Logger.getLogger(App.class);
                logger.trace("Trace Message!");
                logger.debug("Debug Message!");
                logger.info("Info Message!");
                logger.warn("Warn Message!");
                logger.error("Error Message!");
                logger.fatal("Fatal Message!");
        }
}
```

Contenu de `log.txt`

```
2020-11-24 12:40:23 TRACE App:18 - Trace Message!
2020-11-24 12:40:23 DEBUG App:19 - Debug Message!
2020-11-24 12:40:23 INFO  App:20 - Info Message!
2020-11-24 12:40:23 WARN  App:21 - Warn Message!
2020-11-24 12:40:23 ERROR App:22 - Error Message!
2020-11-24 12:40:23 FATAL App:23 - Fatal Message!
```

Logging

Remarques

- Contrairement aux **Java Logging Technology** et **Log4j**, `SLF4J` n'est pas une librairie ni framework de journalisation.
- Il s'agit d'une API (interface) qui utilise les frameworks et librairies de journalisation implémentés pour le langage **Java** tels que **Java Logging Technology**, **Log4j**, **logback**.
- Ici, on va utiliser la version qui utilise **Log4j** (la version 1.2) : **slf4j-log4j12**

Logging

Commençons par ajouter les dépendances suivantes dans `pom.xml`

```
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.5</version>
</dependency>

<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.5</version>
</dependency>
```

Logging

Créons un fichier de propriétés `log4j.properties` **dans** `src/main/resources`

```
log4j.rootLogger=TRACE, stdout

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender

#log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH
   :mm:ss} %-5p %c{1}:%L - %m%n
```

Logging

Dans `App`, commençons par fabriquer une instance `Logger`

```
package org.eclipse.main;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class App {

        public static void main(String[] args)  {

                Logger logger = LoggerFactory.getLogger(App.
                   class);

        }
}
```

Logging

Pour afficher quelques messages de journalisation, on peut utilises plusieurs méthodes

```
package org.eclipse.main;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class App {

        public static void main(String[] args)  {

                Logger logger = LoggerFactory.getLogger(App.class);
                logger.warn("john");
                logger.error("wick");
        }
}
```

Logging

Pour afficher quelques messages de journalisation, on peut utilises plusieurs méthodes

```
package org.eclipse.main;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class App {

        public static void main(String[] args)  {

                Logger logger = LoggerFactory.getLogger(App.class);
                logger.warn("john");
                logger.error("wick");
        }
}
```

Résultat

```
2020-11-29 21:01:27 WARN  App:11 - john
2020-11-29 21:01:27 ERROR App:12 - wick
```

Spring

Cinq niveaux de journalisation (de plus léger vers le plus sévère)

- `TRACE` : pour les messages d'information très détaillés.
- `DEBUG` : pour les messages d'information assez détaillés.
- `INFO` : pour les messages d'information.
- `WARN` : pour les erreurs potentiellement dangereuses.
- `ERROR` : pour les erreurs qui n'empêcheront pas l'application de continuer à s'exécuter.

Pour afficher quelques messages de journalisation, on peut utiliser plusieurs méthodes

```
public class App {

    public static void main(String[] args)  {

        Logger logger = LoggerFactory.getLogger(App.class);
        logger.trace("Trace Message!");
        logger.debug("Debug Message!");
        logger.info("Info Message!");
        logger.warn("Warn Message!");
        logger.error("Error Message!");
    }
}
```

Pour afficher quelques messages de journalisation, on peut utiliser plusieurs méthodes

```
public class App {

    public static void main(String[] args)  {

        Logger logger = LoggerFactory.getLogger(App.class);
        logger.trace("Trace Message!");
        logger.debug("Debug Message!");
        logger.info("Info Message!");
        logger.warn("Warn Message!");
        logger.error("Error Message!");
    }
}
```


Résultat

```
2020-11-29 21:06:27 TRACE App:11 - Trace Message!
2020-11-29 21:06:27 DEBUG App:12 - Debug Message!
2020-11-29 21:06:27 INFO  App:13 - Info Message!
2020-11-29 21:06:27 WARN  App:14 - Warn Message!
2020-11-29 21:06:27 ERROR App:15 - Error Message!
```