LOG4J 2

Un LOGGER (enregistreur) permet d'afficher ou de stocker des informations

Exemple de Logger (trop) basique : System.out.println

LOG4J permet de créer, de paramétrer et d'utiliser des loggers

Destinations paramétrables : console , fichiers, serveurs

Mise en page paramétrable : texte, csv, xml , …

Ajout automatique d'informations comme la date

Un fichier de configuration : log4j2.xml

LOG4J : un exemple d'utilisation

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

package up5.yp.test;
public class ExempleLog4J
{
   private static Logger log = LogManager.getLogger( );

   public static void main(String [ ] args)
   {
       log.info("exécution de la fonction main");
    }
}
```

```
--> 12:58:07 exécution de la fonction main
```

Loggers et niveaux

Les 6 niveaux possibles pour un logger (le niveau choisi est indiqué dans log4J2.xml)

trace debug info warn error fatal

6 méthodes de même nom que les niveaux de logger

```
log.trace(début de la function ");
log.debug("a="+a);
log.info("Le paramètre de f vaut "+a);
log.warn("Le paramètre de f a une valeur nulle "+a);
log.error ("Le paramètre de f a une valeur négative "+a);
log.fatal  ("Le SGBD ne répond pas "+url);
```

Loggers et niveaux

trace debug **info** warn error fatal

L'information est loggée si le niveau du logger est supérieur ou égale au niveau correspondant au nom de la méthode

exemple 1:

log.**info**("Le paramètre de f vaut "+a);
loggé seulement si le niveau du logger est trace, debug ou info

exemple 2:

si le niveau du logger est warn,
seules les méthodes warn error et fatal se traduiront par un enregistrement

LOG4J : un exemple d'utilisation

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

package up5.yp.test;
public class ExempleLog4J
{
   private static Logger log = LogManager.getLogger( );

   public static void main(String [ ] args)
   {
       log.info("message info");
    }
}
```

Si le niveau du logger est trace ou debug ou info
alors le message sera pris en compte.

Logger

Un Logger a un nom, un niveau et un ou plusieurs appenders.

name

exemple : up5.pary.test.ExempleLog4J

Exception : il existe un logger racine (ROOT), il n'a pas de nom.

level

trace debug info warn error fatal

Seuls les messages de niveaux supérieurs ou égal au niveau du logger seront traités

0, 1 ou plusieurs appenders

Un appender par sortie: console, file …

LOG4J 2
Appenders et Layout

LOGGERS (enregisteur)

Chaque logger a un niveau d'alerte parmi :

trace debug info warn error fatal

APPENDERS (ajouteur …)

Différentes destinations possibles pour les messages

console, fichiers, serveurs …

LAYOUTS (mise en page)

Différents formatages possibles

Appender

Exemples d'Appender

ConsoleAppender
FileAppender
JDBCAppender
RollingFileAppender
SMTPAppender

Un Appender a un type, un nom et un layout

type — Console, File, JDBC, RollingFile, Smtp

name — Un nom qui permet de référencer l'appender dans le fichier de configuration

layout — Le format d'écriture du message

Layout

Exemples de Layout

- CsvLayout
- PatternLayout
- HtmlLayout
- XMLLayout

Attributs d'un layout :

- **type** : CsvLayout,PatternLayout,HtmlLayout,XMLLayout
- **charset** : Exemple : UTF-8

Un PatternLayout a en plus un :

- **pattern** : il définit l'affichage à effectuer

PatternLayout
pattern

Exemples de pattern …

```
log.info("HELLO");
```

```
%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n
```

17:00:16.552 [main] INFO up5.mi.pary.test.testlog4j.ExempleLog4J - HELLO

voir https://logging.apache.org/log4j/2.x/manual/layouts.html

LOG4J : un exemple d'utilisation

```
package up5.yp.test;
public class ExempleLog4J {
    private static Logger log = LogManager.getLogger( );
    public static void main(String [ ] args){
        log.info("message info");
     }}
```

Le fichier de configutation log4j2.xml

```
package up5.yp.test;
public class ExempleLog4J {
    private static Logger log = LogManager.getLogger( );
    public static void main(String [ ] args){
        log.info("message info");
     }}
```

```
<?xml version="1.0" encoding="UTF-8"?
<Configuration status="error" strict="true">
  <Appenders>
    <Appender type="Console" name="maConsole">
        <Layout type="PatternLayout" pattern="--> %d{HH:mm:ss} %msg  %n" />
    </Appender>
  </Appenders>

  <Loggers>
    <Logger name="up5.yp.test.ExempleLog4J"  level="debug">
      <AppenderRef ref="maConsole" />
    </Logger>
  </Loggers>
</Configuration>
```

log4j2.xml doit être dans le classpath.
(maven à déposer dans src/main/resources)

log4j2.xml : appenders et layouts

```
<Configuration status="ERROR" strict="true">

  <Appenders>

    <Appender type="Console" name="maConsole">
        <Layout type="PatternLayout" pattern="--> %d{HH:mm:ss} %msg  %n" />
    </Appender>

    <Appender type="File" name="MonFichierDeLog"   fileName="all.log">
      <Layout type="PatternLayout"
              pattern="%d{yyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
    </Appender>

  </Appenders>
```

log4j2.xml : les loggers

```
    <Loggers>
      <Root level="error">
            <AppenderRef  ref="MonFichierDeLog" />
            <AppenderRef  ref="maConsole" />
      </Root>
      <Logger name="up5.mi" level="info" additivity="true">
            <AppenderRef ref="maConsole" />
      </Logger>
    </Loggers>

</Configuration>
```

Nommage des loggers

exemple de noms de logger :

- unnomquelconque
- up5.mi
- up5.mi.pary.test.testlog4j.ExempleLog4J

Très souvent, le nom du logger est le nom complet de la classe où il est créé.

```
// pour initialiser un logger avec come nom celui de la classe, on peut dire :
private static Logger log = LogManager.getLogger("up5.mi.pary.test.testlog4j.ExempleLog4J");
//on peut dire aussi :
private static Logger log = LogManager.getLogger(ExempleLog4J.getClass().getName());
//on peut dire aussi :
private static Logger log = LogManager.getLogger(ExempleLog4J.class);
//ou également (et c'est le meilleur choix)
private static Logger log = LogManager.getLogger( );
```

Loggers parents

Très souvent, le nom du logger est le nom complet de la classe où il est créé.

loggers parents :

up5.mi.pary.test.testlog4j.ExempleLog4J
est le parent de up5.mi.pary.test.testlog4j
qui est le parent de up5.mi.pary.test
qui est le parent de up5.mi.pary
qui est le parent de up5.mi
qui est le parent de up5
qui est le parent de ROOT

Le niveau d'un logger est celui défini pour ce logger
ou celui du parent le plus proche pour lequel un niveau est défini.

niveau des loggers

```
<Root level="error">
        <AppenderRef  ref="MonFichierDeLog" />
</Root>
<Logger name="up5.mi" level="info">
        <AppenderRef ref=" MaConsole" />
    </Logger>
```

up5.mi.pary.test.testlog4j.ExempleLog4J	INFO
up5.mi	INFO
ROOT	ERROR

Appenders utilisés

```
<Root level="error">
        <AppenderRef  ref="MonFichierDeLog" />
</Root>
<Logger name="up5.mi" level="info">
        <AppenderRef ref=" MaConsole" />
    </Logger>
```

Par défaut :
Si un logger est utilisé, alors **les appenders du logger**
et celui des loggers parents sont utilisés (ici MaConsole et MonFichierDeLog)
(indépendamment du niveau indiqué sur les loggers parents)
Pour changer le comportement : ***additivity="false"***

```
<Logger name="up5.mi" level="info" additivity="false">
        <AppenderRef ref=" MaConsole" />
 </Logger>
```

log4J et pom.xml

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.7</version>
</dependency>

<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.7</version>
</dependency>
```