diginamic
« Votre passeport pour l'emploi numérique »

Logging avec SLF4J et LogBack

diginamic
FORMATION
« Votre passeport pour l'emploi numérique »

Objectifs Pédagogiques

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

À l'issue de cette formation, vous serez en mesure de :

- ✓ D'utiliser une API de Logging

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Programme détaillé ou sommaire

Introduction
Configuration
Les principaux concepts
TP

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Chapitre 1 Introduction

Définition

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

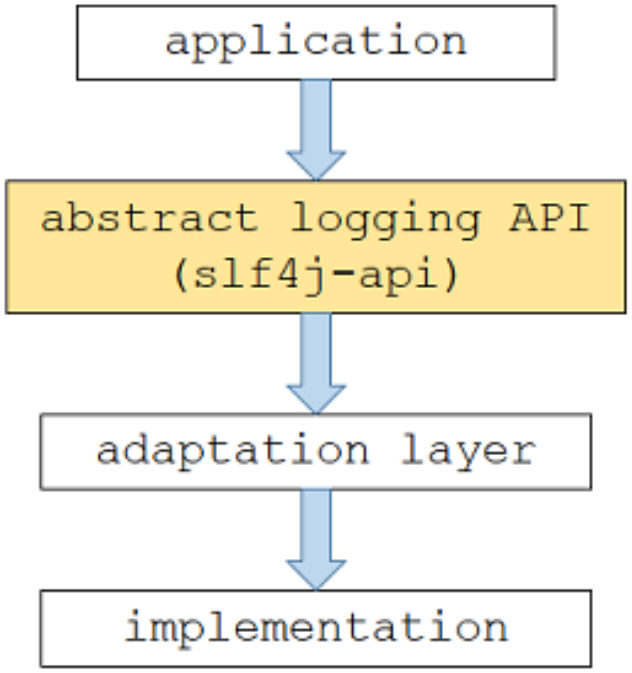
Définition

- Dans une application, le fichier de logs est le journal de bord de votre application. Il fournit des traces sur l'ensemble des traitements exécutés par votre application.

- En cas d'erreur, le fichier de logs doit vous permettre de comprendre ce qui s'est passé, de corroborer des informations fournies par l'utilisateur.

Introduction

- Pour générer ce journal de bord, les applications utilisent des librairies de logging.
- Une librairie de logging est en général associée à une API de logging.
- Votre application a donc besoin d'une API de Logging et d'une implémentation.



Introduction

- **SLF4J**
  - Simple Logging Facade For Java
  - C'est une API d'abstraction qui est compatible avec de nombreuses librairies de logging.
  - Rappelons que l'utilisation d'une API d'abstraction dans une application permet de changer d'implémentation pour passer éventuellement à une librairie plus récente sans changer tout le code.

- **Logback**
  - Implémentation parmi d'autres de SLF4J

Chapitre 2 Configuration

Installation et configuration

Configuration MAVEN

- **pom.xml**

```
<dependency>
	<groupId>ch.qos.logback</groupId>
	<artifactId>logback-classic</artifactId>
	<version>1.2.6</version>
</dependency>

<dependency>
	<groupId>org.slf4j</groupId>
	<artifactId>slf4j-api</artifactId>
	<version>1.7.32</version>
</dependency>
```

Facultatif: tiré par LogBack

Configuration de Logback

- L'implémentation **Logback** utilise un fichier de configuration appelé logback.xml et qui doit être situé à la racine de votre projet.

- Les fichiers de configuration dans un projet MAVEN sont placés dans **src/main/resources**.

- Exemple de fichier logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
	<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
		<layout class="ch.qos.logback.classic.PatternLayout">
			<Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</Pattern>
		</layout>
	</appender>

	<logger name="monLogger" level="DEBUG">
		<appender-ref ref="STDOUT" />
	</logger>

	<root level="ERROR">
		<appender-ref ref="STDOUT" />
	</root>
</configuration>
```

Chapitre 3
Les principaux concepts

Loggers, niveaux de trace, appenders, additivité

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Les principaux concepts

Une trace de log dans SLF4J est associée à un niveau de gravité:

- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- FATAL

Exemple pour écrire dans le journal de logs un message de niveau WARN

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

public class LogExemple {

	private static final Logger LOGGER = LoggerFactory.getLogger("monLogger");

	public static void main(String[] args) {
		LOGGER.warn("L'application vient de démarrer.");

		LOGGER.warn("L'application vient de s'arrêter.");
	}
}
```

Les principaux concepts

Le message de log est intercepté par un intercepteur appelé aussi logger.

Le message n'est envoyé à un media de sortie appelé appender (console, fichier, base de données, etc.) seulement si le niveau de trace du message est supérieur ou égal ou niveau de déclenchement du logger.

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LogExemple {

	private static final Logger LOGGER = LoggerFactory.getLogger("monLogger");

	public static void main(String[] args) {
		LOGGER.warn("L'application vient de démarrer.");
	}
}
```

```xml
<logger name="monLogger" level="INFO">
	<appender-ref ref="STDOUT" />
</logger>
```

Les principaux concepts

Un appender est un media (console, fichier, service distant, etc.) vers lequel vont être envoyés les messages de logs si ces messages passent le filtrage réalisé par le logger sur le niveau de gravité.

Dans l'exemple ci-dessous le media est défini via une balise appender-ref et s'appelle STDOUT

```
<logger name="monLogger" level="DEBUG">
      <appender-ref ref="STDOUT" />
</logger>
```

```
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>%d{HH:mm:ss.SSS} [%thread] %level %logger{36} - %msg%n</Pattern>
      </layout>
</appender>
```

L'appender *STDOUT* est construit avec la classe ConsoleAppender i.e. la sortie Console

Le layout permet de formater le message avec un format défini dans le Pattern

Patterns de message

Pattern:

%d	=> horodatage du message
%d{HH:mm:ss.SSS}	=> horodatage du message avec un format déterminé
%thread	=> indique le thead
%level	=> niveau de trace
%logger	=> nom du logger
%logger{n}	=> nom du logger limité à n caractères
%msg	=> message de log
%n	=> retour à la ligne

Vous pouvez ajouter des caractères de votre choix dans le pattern comme des tirets, des slashs ou même des mots.

```
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>%d{HH:mm:ss.SSS} [%thread] %level %logger{36} - Message: %msg%n</Pattern>
      </layout>
</appender>
```

Les types d'appenders

Logback propose plusieurs *appenders* (https://logback.qos.ch/manual/appenders.html) :

- OutputStreamAppender
- ConsoleAppender
- FileAppender
- RollingFileAppender
- SocketAppender et SSLSocketAppender
- SMTPAppender
- DBAppender

Additivité entre loggers

Le logger racine s'appelle root.

Tous les loggers « héritent » des appenders du logger root, comme une classe qui hérite des propriétés de sa classe mère. Pour les loggers on appelle ça « additivité ».

Attention, l'additivité ne tient pas compte des balises de filtrage (`level`) !!

Avec la configuration ci-dessous on a des messages en double dans la console.

Par additivité, monLogger a 2 appenders :

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
      (…)
       <logger name="monLogger" level="DEBUG">
             <appender-ref ref="STDOUT" />
       </logger>

       <root level="ERROR">
             <appender-ref ref="STDOUT" />
       </root>
</configuration>
```

Héritage entre loggers

Ce mode de fonctionnement n'est pas destiné à piéger les nouveaux utilisateurs mais à mettre en place des mécanismes qui seraient difficiles à mettre en place autrement.

Exemple:

- STDOUT filtre un certain type de messages qui sont redirigés vers un fichier
- Tous les autres messages (ROOT) sont redirigés vers la console

Blocage du mécanisme d'héritage:

additivity="false"

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
      (…)
      <logger name="monLogger" level="DEBUG“ additivity=“false">
            <appender-ref ref="STDOUT" />
      </logger>

      <root level="ERROR">
            <appender-ref ref="STDOUT" />
      </root>
</configuration>
```

Logger et classe

La bonne pratique est de récupérer un Logger non pas à partir de son nom mais pour la classe courante.

Dans ce cas le nom du logger correspondant est celui du package de la classe.

```
package mockito.pizzeria;

public class LogExemple {

	private static final Logger LOGGER = LoggerFactory.getLogger(LogExemple.class);

	public static void main(String[] args) {
		LOGGER.warn("L'application vient de démarrer.");
	}
}
```

```
	<logger name="mockito.pizzeria" level="DEBUG">
		<appender-ref ref="STDOUT" />
	</logger>
```

```
11:57:49.036 [main] WARN  mockito.pizzeria.LogExemple - L'application vient de démarrer.
```

Atelier (TP)

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

OBJECTIFS : mise en œuvre d'une librairie de logging.

DESCRIPTION : Dans ce TP vous allez être amené à mettre en œuvre une librairie de logging SLF4J.

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Annexe: RollingFileAppender

Voici comment créer un système de logs basé sur des fichiers limités à 1000 octets avec une rotation.


```
<appender name="SIZE_BASED_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
      <file>c:/temp/mesLogs.txt</file>

      <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
            <maxFileSize>1KB</maxFileSize>
      </triggeringPolicy>

      <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
            <fileNamePattern>c:/temp/mesLogs-%i.txt</fileNamePattern>
            <minIndex>1</minIndex>
            <maxIndex>3</maxIndex>
      </rollingPolicy>

      <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{10} - %msg%n</pattern>
      </encoder>
</appender>
```