Leveraging SLF4j for Effective Logging in IBM App Connect Enterprise

-Votarikari Shravan(912)

Table of Content

Introduction

SLF4J is a logging framework for Java that acts as a simple facade for different logging frameworks. Its main purpose is to provide a unified logging API, allowing developers to write logging code without being tied to a specific implementation. With SLF4J, developers can use its API to write logging statements, and at runtime, SLF4J routes those calls to the configured logging framework. This separation between the API and implementation allows for easy switching between logging frameworks without changing the application code. SLF4J itself is not a logging implementation, but rather a bridge between the application code and logging frameworks like Logback, Log4j, and Java Util Logging (JUL). Developers can choose the logging framework that suits their needs and integrate it seamlessly with SLF4J. In summary, SLF4J simplifies logging in Java applications by providing a common interface while offering flexibility in the choice of logging implementation. It promotes code portability and simplifies logging management across different projects and environments.

What makes SLF4j so special?

- **Flexibility:** Say goodbye to vendor lock-in! Switch between logging frameworks like Logback, Logstash, or Graylog seamlessly without changing your application code.
- **Simplicity:** Forget bulky APIs. SLF4j provides a lean and concise interface, making logging intuitive and enjoyable.
- **Community**: Backed by a vibrant community, SLF4j is constantly evolving and receiving regular updates.
- **Security:** By promoting the use of actively maintained frameworks, SLF4j helps you stay ahead of evolving security threats.

Prerequisites:

- Jar File (SLF4J implementation)
- Configuration File (xml file describes properties to supply in runtime)
- Shared Library (procedures to call and initialize slf4j)

Preparation

Pre-Setup Tasks

Task 1:

Get the jar file(slf4jimpl.jar) and Place it in the MQSI_WORKPATH/Shared-Classes folder.

Note : Ensure to restart Integration Node to take an effect from the changes made on either server/node yaml file or its Workpath folder.

Task 2:

Get the logback.xml file, store it in any location, Customize the XML Configuration File(**logback.xml**) to specify the path where logs get stored and name for the log file.

Note : remember the path for logback.xml file to specify as property value in further steps.

```
<configuration>
    <property name="LOG_ROOT" value="c:/temp/logs" />
    <property name="LOG_FILE_NAME" value="application" />

<!--    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>
-->
    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file>${LOG_ROOT}/${LOG_FILE_NAME}.log</file>

        <!-- Encoder to format log messages -->
        <layout class="ch.qos.logback.classic.PatternLayout">
            <!-- Define the layout pattern -->
            <pattern>%date{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </layout>
    </appender>

    <logger name="fileLogger" level="INFO" additivity="false">
        <appender-ref ref="FILE"/>
```

Path to store logs

Name of the log file

Task 3:

Create a Folder as given in configuration to store logs and log file.

`C:\temp\logs`

In IBM App Connect Toolkit

Shared Library

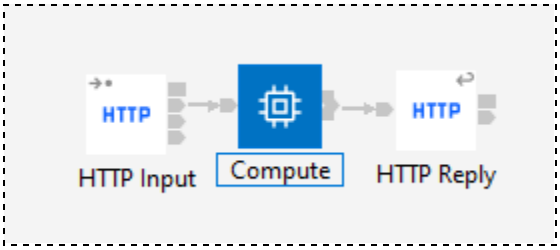
- ➔ Open App Connect Toolkit, Create a Shared Library as following
  File -> New -> Library -> (type-shared library)
- ➔ Create an Esql file in this shared library and define procedure calls for configure and initialize slf4j using java routines

```
CREATE PROCEDURE initSLF4J (IN configFilePath CHARACTER )
LANGUAGE JAVA
EXTERNAL NAME "com.slf4jxmlbased.SLF4JUtility.configLogback";

CREATE PROCEDURE insertLogData (IN logPointName CHARACTER,
                                IN logLevelName CHARACTER,
                                IN dataToLog CHARACTER)
LANGUAGE JAVA
EXTERNAL NAME "com.slf4jxmlbased.SLF4JUtility.writeLogOnFile";
```

Application

- ➔ Create an Application, Message Flow and an Esql file as following to process slf4j
  File-> New -> Application
  File -> New -> Message Flow
  File -> New -> ESQL File
- ➔ Design the message flow as following



- ➔ Write Esql code to write logs on slf4j
  - ◆ **Call initSLF4J** procedure and pass xml configuration file (**logback.xml**) path as a parameter.
  - ◆ **Call insertLogData** procedure and pass 3 values as parameters (label name, log level, logdata).

NOTE : The parameter value for initSLF4J is taken from UDP Value, can be changed by overriding BAR Properties.

```
DECLARE configFilePath EXTERNAL CHARACTER; --External Variable(UDP Value)

CREATE COMPUTE MODULE ImplementSLF4J_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL com.slf4j.initSLF4J(configFilePath); -- A Procedure to initialize SLF4J
        CALL com.slf4j.insertLogData('REQUESTDATA','INFO',InputRoot.XMLNSC); -- A Procedure to Logs the Data
        RETURN TRUE;
    END;
END MODULE;
```

Important Points

1. Log Point Name : it identifies your log data.
   Ex: Channel Request, Exception, Error, etc.
2. Log level : it is a value specified to represent the level of logging.
   Ex: INFO, WARN, TRACE, …etc.
3. Log Data : it is data to store as log information.

Deploy and Run

Now deploy the shared library and application as prepared above steps into a integration node.

Test the application to check logs using a postman testing tool.

Check the logs in C:\temp\logs\application.log file

~Thank You~