How to setup SLF4J and LOGBack in a web app - fast

Added by Cody Burleson, last edited by Cody Burleson on Feb 19, 2010

This page provides a quick guide for setting up SLF4J and LOGBack in a Maven web project in five simple steps.

Logback is intended as a successor to the popular log4j project. It was designed, in addition to many individual contributors, by Ceki Gülcü, the founder of log4j. It builds upon experience gained in building industrial-strength logging systems going back as far as 1999. Logback-classic natively implements the SLF4J API so that you can readily switch back and forth between logback and other logging frameworks such as log4j or java.util.logging (JUL).

The Simple Logging Facade for Java or (SLF4J) serves as a simple facade or abstraction for various logging frameworks, e.g. java.util.logging, log4j and logback, allowing the end user to plug in the desired logging framework at deployment time.

If your working with a Maven web-app project, this procedure will get you setup to log with LOGBack through SLF4J super fast.

Step 1 - Add LOGBack dependency to your Maven POM

Declare the following dependency in your Maven 2 pom.xml and Maven will grab the appropriate libraries for you during the build.

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>0.9.18</version>
</dependency>
```

Step 2 - Import existing (starter) XML configuration files

You will likely want to start with a base configuration file that you can build upon. In Maven you can have a logging configuration for your main source and another for your testing. You can download starter configuration files for your project by clicking the links in the hierarchy below. Put them in your project according to the position indicated by the hierarchy shown.

- src
  - main
    - resources
      - logback.xml
  - test
    - resources
      - logback-test.xml

Step 3 - Customize the XML configuration just enough to test

Open up the logback.xml file. If you used the starter provided in the link above, you'll find the following:

```
<?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="com.base22" level="TRACE"/>


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

You will notice that one logger is defined at a package level ("com.base22"). You can simply change that to match your application's package base. You can also declare additional loggers (packages and/or classes) if desired.

Related Content

SLF4J
The Simple Logging Facade for Java or (SLF4J) serves as a simple facade or abstraction for various logging frameworks.

Java Logging Standards and Guidelines
Make your logging output as consistent as possible across classes.

How to setup Log4j in a web app - fast

Logging with SLF4J

SLF4J: Now you can do logging in J
Great thread from the ServerSide with a wealth of opinions about various logging frameworks.

Step 4 - Put logging code in your classes

The last thing you need to do is drop some logging code in a class and test this whole setup.

Add the following to the imports section of your java code:

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

Add the following at the top of your class in the global section (just under the line that declares your class public class Whatever extends Whatever). Change the name of the class (MyClassName) in the getLogger method call, of course. Name it the same as the class you're dropping this code into.

```java
static final Logger LOG = LoggerFactory.getLogger(MyClassName.class);
```

Throw some logging statements in your code somewhere where you know they'll be fired right away when you run your app. For example:

```java
LOG.trace("Hello World!");
LOG.debug("How are you today?");
LOG.info("I am fine.");
LOG.warn("I love programming.");
LOG.error("I am programming.");
```

Alternatively, you can just download this simple console test app and run it as a Java app from the command line or from within your IDE:

SLF4JConsoleTest.java

This class has a main method so it runs as a Java app and it will log one statement at each level.

Step 5 - Run your app and make sure it works

Finally, run your app and make sure it works. You should see log lines in your console. If it doesn't work, just review these steps a little more carefully and fiddle with it.

Comments (0) Attachments (3) Labels