Logging with Log4J

This is a very easy-to-use Log4J functionality inside Spring applications. The following example will take you through simple steps to explain the simple integration between Log4J and Spring.

We assume you already have **log4J** installed on your machine. If you do not have it then you can download it from <https://logging.apache.org/> and simply extract the zipped file in any folder. We will use only **log4j-x.y.z.jar** in our project.

Next, let us have a working Eclipse IDE in place and take the following steps to develop a Dynamic Form-based Web Application using Spring Web Framework −

|  |  |
| --- | --- |
| **Steps** | **Description** |
| 1 | Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Add log4j library *log4j-x.y.z.jar* as well in your project using using *Add External JARs*. |
| 4 | Create Java classes *HelloWorld* and *MainApp* under the *com.tutorialspoint*package. |
| 5 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 6 | Create log4J configuration file *log4j.properties* under the **src** folder. |
| 7 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

Here is the content of **HelloWorld.java** file

package com.tutorialspoint;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage() {

System.out.println("Your Message : " + message);

}

}

Following is the content of the second file **MainApp.java**

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.apache.log4j.Logger;

public class MainApp {

static Logger log = Logger.getLogger(MainApp.class.getName());

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

log.info("Going to create HelloWord Obj");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

log.info("Exiting the program");

}

}

You can generate **debug** and **error** message in a similar way as we have generated info messages. Now let us see the content of **Beans.xml** file

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

</beans>

Following is the content of **log4j.properties** which defines the standard rules required for Log4J to produce log messages

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

# Set the name of the file

log4j.appender.FILE.File=C:\\log.out

# Set the immediate flush to true (default)

log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode

log4j.appender.FILE.Threshold=debug

# Set the append to false, overwrite

log4j.appender.FILE.Append=false

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%m%n

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message in Eclipse console −

Your Message : Hello World!

If you check your C:\\ drive, then you should find your log file **log.out** with various log messages, like something as follows −

<!-- initialization log messages -->

Going to create HelloWord Obj

Returning cached instance of singleton bean 'helloWorld'

Exiting the program

## Jakarta Commons Logging (JCL) API

Alternatively you can use **Jakarta Commons Logging (JCL)** API to generate a log in your Spring application. JCL can be downloaded from the <https://jakarta.apache.org/commons/logging/>. The only file we technically need out of this package is the *commons-logging-x.y.z.jar* file, which needs to be placed in your classpath in a similar way as you had put *log4j-x.y.z.jar* in the above example.

To use the logging functionality you need a *org.apache.commons.logging.Log*object and then you can call one of the following methods as per your requirment −

* fatal(Object message)
* error(Object message)
* warn(Object message)
* info(Object message)
* debug(Object message)
* trace(Object message)

Following is the replacement of MainApp.java, which makes use of JCL API

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.apache.commons.logging. Log;

import org.apache.commons.logging. LogFactory;

public class MainApp {

static Log log = LogFactory.getLog(MainApp.class.getName());

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

log.info("Going to create HelloWord Obj");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

log.info("Exiting the program");

}

}

You have to make sure that you have included commons-logging-x.y.z.jar file in your project, before compiling and running the program.

Now keeping the rest of the configuration and content unchanged in the above example, if you compile and run your application, you will get a similar result as what you got using Log4J API.