Java Util Logging - Customizing log format

[Last Updated: May 25, 2017]

By default Java Util logging (JUL) uses following configurations (JDK 8):

```
handlers= java.util.logging.ConsoleHandler
.level= INFO
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

The complete default configuration file can be found at the JDK/JRE installation directory: `JAVA_HOME/jre/lib/logging.properties`.

The default handler, ConsoleHandler, sends log records to `System.err` (the available command line console). This handler, by default uses `SimpleFormatter` to format logs. In this tutorial, we will learn how to modify the default log format.

The default formatter, SimpleFormatter, formats the output by using following method call:

```
String.format(format, date, source, logger, level, message, thrown);
```

The first argument 'format' can be customized in the logging.properties or by a command line option or can be set programmatically. The good thing, we don't have to learn new formatting specifiers here, as java.util.Formatter specification is fully supported.

Using logging.properties

Let's add a new line to logging.properties with property key `java.util.logging.SimpleFormatter.format` to specify our desire log format.

src/main/resources/logging.properties

```
handlers= java.util.logging.ConsoleHandler
.level= INFO
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.SimpleFormatter.format=[%1$tF %1$tT] [%4$-7s] %5$s %n
```

```
public class MyClass {
  private static Logger LOGGER = null;

  static {
      InputStream stream = MyClass.class.getClassLoader().
              getResourceAsStream("logging.properties");
      try {
          LogManager.getLogManager().readConfiguration(stream);
          LOGGER= Logger.getLogger(MyClass.class.getName());

      } catch (IOException e) {
          e.printStackTrace();
      }
  }
  public static void main(String[] args) {
      System.out.println("-- main method starts --");
      LOGGER.info("in MyClass");
      LOGGER.warning("a test warning");
  }
}
```

Output

```
-- main method starts --
[2017-05-18 14:02:49] [INFO   ] in MyClass
[2017-05-18 14:02:49] [WARNING] a test warning
```

Check this out to quickly learn different time formatting options

Using system property

```
public class MyClass2 {
  private static Logger LOGGER = null;

  static {
      System.setProperty("java.util.logging.SimpleFormatter.format",
              "[%1$tF %1$tT] [%4$-7s] %5$s %n");
      LOGGER = Logger.getLogger(MyClass2.class.getName());
  }

  public static void main(String[] args) {
      System.out.println("-- main method starts --");
      LOGGER.info("in MyClass2");
      LOGGER.warning("a test warning");
  }
}
```

Output

```
-- main method starts --
[2017-05-18 14:03:41] [INFO   ] in MyClass2
[2017-05-18 14:03:41] [WARNING] a test warning
```

Programmatically setting format

Here we have to set our own ConsoleHandler with customized SimpleFormatter instead of customizing the default one.

```
package com.logicbig.example;

import java.util.Date;
import java.util.logging.ConsoleHandler;
import java.util.logging.LogRecord;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class MyClass3 {
  private static Logger LOGGER = null;

  static {
      Logger mainLogger = Logger.getLogger("com.logicbig");
      mainLogger.setUseParentHandlers(false);
      ConsoleHandler handler = new ConsoleHandler();
      handler.setFormatter(new SimpleFormatter() {
          private static final String format = "[%1$tF %1$tT] [%2$-7s] %3$s %n";

          @Override
          public synchronized String format(LogRecord lr) {
              return String.format(format,
                      new Date(lr.getMillis()),
                      lr.getLevel().getLocalizedName(),
                      lr.getMessage()
              );
          }
      });
      mainLogger.addHandler(handler);
      LOGGER = Logger.getLogger(MyClass3.class.getName());
  }

  public static void main(String[] args) {
      System.out.println("-- main method starts --");
      LOGGER.info("in MyClass3");
      LOGGER.warning("a test warning");
  }
}
```

Output

```
-- main method starts --
[2017-05-18 14:04:07] [INFO   ] in MyClass3
[2017-05-18 14:04:07] [WARNING] a test warning
```

Hewlett Packard Enterprise | AMD

MEJORA TU AGILIDAD OPERATIVA.
Servidor HPE ProLiant DL365 Gen10 Plus

Example Project

Dependencies and Technologies Used:

- JDK 1.8
- Maven 3.3.9

Core Java Tuto

Java 15 Features
Java 14 Features
Java 13 Features
Java 12 Features
Java 11 Features
Java 10 Features
Java 9 Module Sy

Recent Tutorial

Loading Logging Props

customizing-default-log-format
src
main
java
com
logicbig
example
MyClass.java
MyClass2.java
MyClass3.java
resources
logging.properties
pom.xml

```
        Logger mainLogger = Logger.getLogger("com.logicbig");
        mainLogger.setUseParentHandlers(false);
        ConsoleHandler handler = new ConsoleHandler();
        handler.setFormatter(new SimpleFormatter() {
            private static final String format = "[%1$tF %1$tT] [%2$-7s] %3$s %n";

            @Override
            public synchronized String format(LogRecord lr) {
                return String.format(format,
                        new Date(lr.getMillis()),
                        lr.getLevel().getLocalizedName(),
                        lr.getMessage()
                );
            }
        });
        mainLogger.addHandler(handler);
        LOGGER = Logger.getLogger(MyClass3.class.getName());
    }

    public static void main(String[] args) {
        System.out.println("-- main method starts --");
        LOGGER.info("in MyClass3");
        LOGGER.warning("a test warning");
    }
}
```

dynatrace G2 reviewers agree: Dynatrace outperforms Datadog G2 satisfaction score: Dynatrace DataDog 54%

Project Structure

customizing-default-log-format
src
main
java
com
logicbig
example
MyClass.java
MyClass2.java

MyClass3.java
resources
logging.properties
pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.logicbig.example</groupId>
    <artifactId>customizing-default-log-format</artifactId>
    <version>1.0-SNAPSHOT</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>


</project>
```

UP

```
package com.logicbig.example;

import java.io.IOException;
import java.io.InputStream;
import java.util.logging.LogManager;
import java.util.logging.Logger;

public class MyClass {
    private static Logger LOGGER = null;

    static {
        InputStream stream = MyClass.class.getClassLoader().
                getResourceAsStream("logging.properties");
        try {
            LogManager.getLogManager().readConfiguration(stream);
            LOGGER= Logger.getLogger(MyClass.class.getName());

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        System.out.println("-- main method starts --");
        LOGGER.info("in MyClass");
        LOGGER.warning("a test warning");
    }
}
```

UP

```
package com.logicbig.example;

import java.util.logging.Logger;

public class MyClass2 {
    private static Logger LOGGER = null;

    static {
        System.setProperty("java.util.logging.SimpleFormatter.format",
                "[%1$tF %1$tT] [%4$-7s] %5$s %n");
        LOGGER = Logger.getLogger(MyClass2.class.getName());
    }

    public static void main(String[] args) {
        System.out.println("-- main method starts --");
        LOGGER.info("in MyClass2");
        LOGGER.warning("a test warning");
    }
}
```

UP

```
package com.logicbig.example;

import java.util.Date;
import java.util.logging.ConsoleHandler;
import java.util.logging.LogRecord;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class MyClass3 {
    private static Logger LOGGER = null;

    static {
        Logger mainLogger = Logger.getLogger("com.logicbig");
        mainLogger.setUseParentHandlers(false);
        ConsoleHandler handler = new ConsoleHandler();
        handler.setFormatter(new SimpleFormatter() {
            private static final String format = "[%1$tF %1$tT] [%2$-7s] %3$s %n";

            @Override
            public synchronized String format(LogRecord lr) {
                return String.format(format,
                        new Date(lr.getMillis()),
                        lr.getLevel().getLocalizedName(),
                        lr.getMessage()
                );
            }
        });
        mainLogger.addHandler(handler);
        LOGGER = Logger.getLogger(MyClass3.class.getName());
    }

    public static void main(String[] args) {
        System.out.println("-- main method starts --");
        LOGGER.info("in MyClass3");
        LOGGER.warning("a test warning");
    }
}
```

UP

```
handlers= java.util.logging.ConsoleHandler
.level= INFO
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
#Setting a custom log format
java.util.logging.SimpleFormatter.format=[%1$tF %1$tT] [%4$-7s] %5$s %n
```

UP

See Also

- Java String Formatting
- Loading logging.properties
- Getting started with Java Logging
- Log Levels
- Creating a custom log Handler
- Enabling JDK Internal Logging

Core Java Tutorials

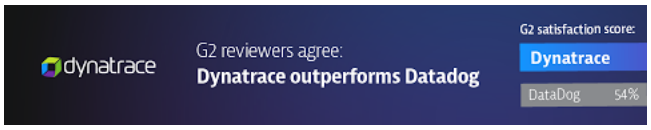
- Java 15 Features
- Java 14 Features
- Java 13 Features
- Java 12 Features
- Java 11 Features
- Java 10 Features
- Java 9 Module System
- Java 9 Misc Features
- Java 9 JShell

Recent Tutorials

- Java 16 - Records Features, Quick Walk-through
- Java 16 - Introduction to Records
- Spring - Injecting beans into Arrays/Collections, Using @Qualifiers And Specifying the Ordering
- Injecting Collections - Injecting Beans Into Arrays And Collections, ordering with Ordered Interface
- Spring - Injecting beans Into Arrays and Lists, ordering with @Ordered annotation
- Spring - Injecting beans into Arrays and Collections, selecting elements with @Qualifier annotation
- Spring - Injecting multiple Beans Into Arrays and Collections
- Spring - Arrays and Collections As Beans
- Spring - Using @ComponentScan#excludeFilters to exclude cla annotations
- Spring - Using @ComponentScan#includeFilters to scan non co annotations
- Spring - Implementing ApplicationContextAware Interface
- Spring - Using excludeFilters attribute of @ComponentScan to e
- Spring - Using @ComponentScan to scan non component classe

Share

Pr



Copyright © 2016-2021 LogicBig.