Chapter 1: Introduction

The morale effects are startling. Enthusiasm jumps when there is a running system, even a simple one. Efforts redouble when the first picture from a new graphics software system appears on the screen, even if it is only a rectangle. One always has, at every stage in the process, a working system. I find that teams can grow much more complex entities in four months than they can build.

—FREDERICK P. BROOKS, JR., *The Mythical Man-Month*

《人月神话》

Authors: Ceki Gülcü, Sébastien Pennec, Carl Harris

What is logback?

Logback is intended as a successor to the popular log4j project. It was designed by Ceki Gülcü, log4j's founder. It builds upon a decade of experience gained in designing industrial-strength logging systems. The resulting product, i.e. logback, is faster and has a smaller footprint than all existing logging systems, sometimes by a wide margin. Just as importantly, logback offers unique and rather useful features missing in other logging systems. Reasons to prefer logback over log4j

First Baby Step

Requirements

Logback-classic module requires the presence of *slf4j-api.jar* and *logback-core.jar* in addition to *logback-classic.jar* on the classpath.

The *logback-*.jar* files are part of the logback distribution whereas *slf4j-api-1.7.7.jar* ships with SLF4J, a separate project.

In order to run the examples in this chapter, you need to make sure that certain jar files are present on the classpath. Please refer to the setup page for further details.

Let us now begin experimenting with logback.

Example 1.1: Basic template for logging (*logback-examples/src/main/java/chapters/introduction/HelloWorld1.java*)

```
package chapters.introduction;

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

public class HelloWorld1 {

  public static void main(String[] args) {

    Logger logger = LoggerFactory.getLogger("chapters.introduction.HelloWorld1");
    logger.debug("Hello world.");

  }
}
```

`HelloWorld1` class is defined in the `chapters.introduction` package. It starts by importing the `Logger` and `LoggerFactory` classes defined in the SLF4J API, specifically within the `org.slf4j` package.

On the first line of the main() method, the variable named `logger` is assigned a `Logger` instance retrieved by invoking the static `getLogger` method from the `LoggerFactory` class. This logger is named "chapters.introduction.HelloWorld1". The main method proceeds to call the `debug` method of this logger passing "Hello World" as an argument. We say that the main method contains a logging statement of level DEBUG with the message "Hello world".

Note that the above example does not reference any logback classes. In most cases, as far as logging is concerned, your classes will only need to import SLF4J classes. Thus, the vast majority, if not all, of your classes will use the SLF4J API and will be oblivious to the existence of logback.

You can launch the first sample application, *chapters.introduction.HelloWorld1* with the command:

```
java chapters.introduction.HelloWorld1
```

Launching the `HelloWorld1` application will output a single line on the console. By virtue of logback's default configuration policy, when no default configuration file is found, logback will add a `ConsoleAppender` to the root logger.

```
20:49:07.962 [main] DEBUG chapters.introduction.HelloWorld1 - Hello world.
```

Logback can report information about its internal state using a built-in status system. Important events occurring during logback's lifetime can be accessed through a component called `StatusManager`. For the time being, let us instruct logback to print its internal state by invoking the static `print()` method of the `StatusPrinter` class.

Example: Printing Logger Status (logback-examples/src/main/java/chapters/introduction /HelloWorld2.java)

```
package chapters.introduction;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import ch.qos.logback.classic.LoggerContext;
import ch.qos.logback.core.util.StatusPrinter;

public class HelloWorld2 {

  public static void main(String[] args) {
    Logger logger = LoggerFactory.getLogger("chapters.introduction.HelloWorld2");
    logger.debug("Hello world.");

    // print internal state
    LoggerContext lc = (LoggerContext) LoggerFactory.getILoggerFactory();
    StatusPrinter.print(lc);
  }
}
```

Running the `HelloWorld2` application will produce the following output:

```
12:49:22.203 [main] DEBUG chapters.introduction.HelloWorld2 - Hello world.
12:49:22,076 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT fi
12:49:22,078 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT fi
12:49:22,093 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT fi
12:49:22,093 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Setting up d
```

Logback explains that having failed to find the *logback-test.xml* and *logback.xml* configuration files (discussed later), it configured itself using its default policy, which is a basic `ConsoleAppender`. An

`Appender` is a class that can be seen as an output destination. Appenders exist for many different destinations including the console, files, Syslog, TCP Sockets, JMS and many more. Users can also easily create their own Appenders as appropriate for their specific situation.

Note that in case of errors, logback will automatically print its internal state on the console.

The previous examples are rather simple. Actual logging in a larger application would not be that different. The general pattern for logging statements would not change. Only the configuration process would be different. However, you would probably want to customize or configure logback according to your needs. Logback configuration will be covered in subsequent chapters.

Note that in the above example we have instructed logback to print its internal state by invoking the `StatusPrinter.print()` method. Logback's internal status information can be very useful in diagnosing logback-related problems.

Here is a list of the three required steps in order to enable logging in your application.

1. Configure the logback environment. You can do so in several more or less sophisticated ways. More on this later.
2. In every class where you wish to perform logging, retrieve a `Logger` instance by invoking the `org.slf4j.LoggerFactory` class' `getLogger()` method, passing the current class name or the class itself as a parameter.
3. Use this logger instance by invoking its printing methods, namely the debug(), info(), warn() and error() methods. This will produce logging output on the configured appenders.

Building logback

As its build tool, logback relies on Maven, a widely-used open-source build tool.

Once you have installed Maven, building the logback project, including all its modules, should be as easy as issuing a `mvn install` command from within the directory where you unarchived the logback distribution. Maven will automatically download the required external libraries.

Logback distributions contain complete source code such that you can modify parts of logback library and build your own version of it. You may even redistribute the modified version, as long as you adhere to the conditions of the LGPL license or the EPL license.

For building logback under an IDE, please see the relevant section on the class path setup page.