Java Loggers

Java Logging

Logging in a program is important to keep track of a program's run-time behavior. Logs capture and persist the important data and make it available for analyst later on.

Logging frameworks

In Java there are multiple frameworks that you can use for logging. The framework give you the objects, methods, and configuration necessary to create and send log messages.

The built-in framework is `java.util.logging` package. However, there are many third-party frameworks like `Log4j, Logback, tinylog`. There are also something called an abstraction layer like `SLF4J` and `Apache Commons Logging` that decouples your code from the underlying logging framework. This is so that you can switch between logging frameworks on the fly.

More about abstraction layers

Each of the logging framework have different objects that is created and all have different method names that is used to do the logging. As a programmer you are not going to remember all of their names and such the abstraction layer is created to cope with that problem.

Abstraction layer like `SLF4J` decouple the logging framework from your application, so that as a programmer you won't have to worry about exactly what the method and what object to create when you want to use say `Log4j` to do your logging, or if you want to switch it after developing your application for some time to say `Logback` you can easily switch it without any pain. i.e. not having to change all the object and method name that was used with the previous logging framework.

Logging components

There are three core components to Java logging

- Loggers: They are responsible for capturing the events that you are logging and they pass it to the appropriate appender. The logger can sent it to multiple appender
- Appender (Also called handler): Responsible for recording the log to the specified destination. Appender use layouts to format the log before sending them to the destination
- Layouts (Also called formatter): Responsible for converting and formatting the data in the log. They basically determine how your log will look like when they are sent to the destination

So when you application make a logging call, the Logger will record the event to a LogRecord and then forwards it to the appropriate Appender. The Appender will format the record using the Layout before sending it to the destination like the Console, to a File, or to another Application as well.

Filters can be applied to specify which Appender should be used for what type of logs. You can sent like configuration logs to the console, but warning messages to a file.

Configuration

Each framework will be configure through configuration files. The configuration files are bundled with your executable and then are loaded by the logging framework at runtime to determine the behavior of the logging framework. Although you can also configure it in code, doing it using a configuration file is more maintainable and easier.

Loggers

The logger objects is used to trigger log events that then record the event to a LogRecord which then is forwarded to the appropriate Appender for destination. One class can have multiple independent Loggers responding to different events and you can also nest Loggers.

Create new logger

To create a new Logger you would do:

`Logger logger = Logger.getLogger(MyClass.class.getName());`

Logging event

Then to trigger a log event you would call methods on the Logger object that is created

```
logger.log(Level.WARNING, "This is a warning!");


// or a short hand
logger.warning("This is a warning!");
```

The logs have levels. The level determines the severity of the log and can be used to filter the event or send it to a different Appender.

You can also prevent a Logger from logging messages below a certain level. For example in `Log4j` the level are ordered like such ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF.

This means that if you set the log level to WARN. Then it will disable all log levels below WARN, so it will show all messages except DEBUG and INFO.

Default logger and handler

So if you are using the global logger initialized by `java.util.Logger`, it is defaulted to `Level.INFO`. So you want to change the root logger's level if you want finer logs if you are using the root logger.

In addition, each logger also have a default Console handler. It also have it's level set to `Level.INFO` by default as well.

So if you want your own logger to show finer logs then you want to disable the default console handler then add your own console handler with it's own custom levels.

```
log.setUseParentHandlers(false);
```

Appender / Handler

Append formats the log then sent the log to the correct output destination.

A logger can have multiple Appender. The two common appender (handler) for `java.util.logging` are `ConsoleHandler` and `FileHandler`.

You can then format the

Specifying the java.util.logging.config.file

You can specify where the `logging.properties` files are by passing it as a system property value

Layout / Formatter

For each appender you can also specify a layout or formatter.

By default the Console handler have the SimpleFormatter which just outputs the log in simple text.

However, FileHandler outputs it in XML format.

Revision #2
Created 2023-03-17 16:52:02 UTC by Tamarine
Updated 2023-03-17 20:53:29 UTC by Tamarine