Logging Service

Summary

The e-government framework adopts Log4j that can leave log using Log4j as an open source.

Logging records the matters that may occur during operation or development of system in the external storage of the system, helping easy identification of the system situation. The common method that many developers use for displaying Log is System.out.println(). However, this method is not recommended due to the following while it is simple.

- Even if the console log is redirected to an output file, the file may not be overwritten when the application server restarts.
- It is not a good method to use System.out.println() just when developing and testing only and to delete it before transferring to operation.
- Since System.out.println() calling is synchronized during disk I/O, throughput of system decreases.
- By default, the stack trace results remain in the console. However, it is not desirable to track the Exception through console during system operation.

It is because the code during operation may operate different from that during test. Accordingly, a mechanism is required that can manage the logging declaratively and minimize the performance overhead during operation in order to have testing code and operation code identically.

Description

How to Set Log4j Preference

How to set Log4j preference can be divided into following 2 types as shown below.

- Directly Set in Programming
- Using Setting File

Description of Important Component

1. **Logger**: the entity of log (class creating the log file) – Almost all logging functions except setting are processed through this. Define the logger(based on logger name) to use for each application and designate log level and appender for this.

The logger has log level and whether to display log or not is determined by level of log sentence and level of logger.

- Determine which logger to use before creating application.

ex) static Logger logger = Logger.getLogger(SimpleLog.class);

[Note] Commons-Logging also has wrapper class.

2. **Appender**: location to print out log

- It means the position to print out the log. The position of output can be guessed more or less by checking the name of classes that end with XXXAppender in Log4J API document.

http://logging.apache.org/log4j/docs/api/index.html

3. **Layout**: output format of Appender – select several information such as date, time and class name, and can be designated as log information contents.

Detailed pattern can be found in the following information.
http://logging.apache.org/log4j/docs/api/org/apache/log4j/PatternLayout.html

Type of Layout

1) DateLayout,
2) HTMLLayout,
3) PatternLayout, (In general, using PatternLayout is the most suitable for debugging)
4) SimpleLayout,
5) XMLLayout

Description on Pattern Layout

ex) ”[%d{yyyy-MM-ddHH:mm:ss}] %-5p [%l] - %m%n

Pattern Layout	Description
C	Displays class name. Can set to display the class name or over specific package by adding the setting such as DateLayout.
d	Displays log time. Can designate the output format suitable injava.text.SimpleDateFormat.
F	Displays the file name. Methods executed at the time of log and line number are displayed together.
L	Displays line number only.
m	Displays the message delivered to log.
M	Displays the name of method executing log.
n	Line feed.
p	Log event name (DEBUG, etc.)
r	Log processing time (milliseconds)

Designate Log Level

The log4j has fivelog levels such as debug, info, warn, error and fatal.

Each can leave log using five methods such as method debug(), info(), warn(), error(), and fatal().

Log level is as follows. (FATAL > ERROR > WARN > INFO > DEBUG > TRACE)

Log Level	Description
fatal	Indicating that serious error occurs. A systematically serious problem occurs so that application cannot be operated. In general, it is not used in application.
error	Indicating that a problem may occur during processing of request.
warn	Indicating the warning message that can be the cause of system error while this problem can be processed.
info	Indicating the informative message such as login or status change.
debug	Indicating the message used for debug purpose at the time of development.
trace	Level newly added at log4j1.2.12. Indicating more detailed status to solve the debug level is too wide.

Provided, however, that, logging events under the designated log level are ignored if there is a log level designated at setLevel of Logger.

Accordingly, no log remains. That is, as shown below:

logger.setLevel(Level.INFO);

if designated in the code, among three codes shown below:

```
logger.debug("debug log");
logger.info("info log");
logger.warn("warning log");
```

A debug log does not remain but only info and warn log remains. Since there is no function of pre-processer like C in Java, debugging codes cannot be created separately when it is debugging and when it is releasing like the form of #ifdef DEBUG. Accordingly, such functions of log4j are very convenient in log management.

Appender

log4j supports various log output targets and methods such as console, file, DB, socket, message and mail, and can be diversely defined with Appender of log4j.

ConsoleAppender: appender to print out to console screen. org.apache.log4j.ConsoleAppender: Appender to display in the Console screen. The following is the contents for definition of property for ConsoleAppender in log4j.xml file.

```
<appender name="console" class="org.apache.log4j.ConsoleAppender">
	<!-- ref.) attr: Encoding, ImmediateFlush, Target, Threshold -->
	<layout class="org.apache.log4j.PatternLayout">
		<param name="ConversionPattern" value="%d %5p [%c] %m%n" />
	</layout>
</appender>
```

FileAppender: FileAppender is used to log in the file.

```
<appender name="file" class="org.apache.log4j.FileAppender">
	<!--
		ref.) attr: Append, Encoding, BufferedIO, BufferSize, File,
		ImmediateFlush, Threshold
	-->
	<param name="File" value="./logs/file/sample.log" />
	<!—Set to Append false to overwrite the file every time for convenience of test -->
	<param name="Append" value="false" />
	<layout class="org.apache.log4j.PatternLayout">
		<param name="ConversionPattern" value="%d %5p [%c] %m%n" />
	</layout>
</appender>

<appender name="mdcFile" class="org.apache.log4j.FileAppender">
	<param name="File" value="./logs/file/mdcSample.log" />
	<param name="Append" value="false" />
	<layout class="org.apache.log4j.PatternLayout">
		<!—IncludingMDC related pattern-->
		<param name="ConversionPattern"
			value="%d %5p [%c] [%X{class} %X{method} %X{testKey}] %m%n" />
	</layout>
</appender>
```

RollingFileAppender: Since FileAppender leaves the log in the file continuously, the size of file may be too big, making it impossible to manage log systematically. If the size of file gets bigger than specific size through designation of file backup index or size of file, change the existing file to backup file and start logging from the start.

```
<!-- log4j-1.3alpha-8 style – structure changed with package change and policy processing -->
<appender name="rollingFile" class="org.apache.log4j.rolling.RollingFileAppender">
	<rollingPolicy class="org.apache.log4j.rolling.FixedWindowRollingPolicy">
		<param name="FileNamePattern" value="./logs/rolling/rollingSample.%i.log" />
		<param name="MaxIndex" value="3" />
	</rollingPolicy>
```

```
        <triggeringPolicy class="org.apache.log4j.rolling.SizeBasedTriggeringPolicy">
                <param name="MaxFileSize" value="1000" />
        </triggeringPolicy>
        <layout class="org.apache.log4j.PatternLayout">
                <param name="ConversionPattern" value="%d %5p [%c] %m%n" />
        </layout>
</appender>
```

DailyRollingFileAppender: perform logging according to the date or condition set earlier. Can designate the time when log is rolled as well as the time when the object is created using the creators DailyRollingFileAppender(Layout layout, String filename, String datePattern); or using setDatePattern() after creating in default creators. See Apache api documents for more details.

```
<appender name="dailyRollingFile" class="org.apache.log4j.DailyRollingFileAppender">
        <!-- ref.) attr: FileAppender + DatePattern -->
        <param name="File" value="./logs/daily/dailyRollingSample.log" />
        <param name="Append" value="true" />
        <!--
                Follow SimpleDateFormat. ex.) .yyyy-ww: every week from the first day, .yyyy-MM-dd-HH-mm: every minute
        -->
        <param name="DatePattern" value=".yyyy-MM-dd-HH-mm-ss" />
        <layout class="org.apache.log4j.PatternLayout">
                <param name="ConversionPattern" value="%d %5p [%c] %m%n" />
        </layout>
</appender>
```

JDBCAppender: Appender to print out the log in DB. Parameters such as Driver, URL, User, Password, Sql can be defined under it. The following is the contents of property definitions for JDBCAppender with log4j.xml file.

```
<appender name="db" class="org.apache.log4j.jdbc.JDBCAppender">
        <!-- parameter for defining JDBC Driver-->
        <param name="Driver" value="oracle.jdbc.driver.OracleDriver"/>
        <!—parameter for defining DB URL-->
        <param name="URL" value="jdbc:oracle:thin:@107.108.150.108:1521:ora10"/>
        <!-- parameter for defining DB User-->
        <param name="User" value="egovframe"/>
        <!-- parameter for defining DB Password-->
        <param name="Password" value="egovframe"/>
        <!-- parameter for defining the query to execute when leaving log-->
        <param name="Sql" value="insert into STMR_LOG (msg)
        values('%d %p [%c] - &lt;%m&gt;%n')"/>
</appender>
```

E-government Expansion Appender

EgovDBAppender:

- In case of Oracle(ojdbc-14.jar jdbc type 4 thin), a problem may occur when executing getGeneratedKeys. In this regard, according to useSupportsGetGeneratedKeys flag, process getGeneratedKeys of JDBC3.0 with EgovDBAppender that can process it as unavailable option(TEST DB: 10g r2 version)
- DB Appender of log4j-1.3alpha-8 is extended. According to useSupportsGetGeneratedKeys flag, this appender is added with option not to use getGeneratedKeys of JDBC3.0.

In case of Oracle(ojdbc-14.jar jdbc type 4 thin) if you execute getGeneratedKeys, java.sql.SQLException: operation not allowed) error occurs. Support not to cause any problem in processing of DBAppender for Oracle by setting not to use useSupportsGetGeneratedKeysthrough EgovDBAppender.

```
<appender name="egovDB" class="org.apache.log4j.db.EgovDBAppender">
        <!--caller_filename, caller_class, caller_method, caller_line -->
        <param name="locationInfo" value="true" />
        <!—In case of Oracle, following is set to false or following option line is deleted(false by
default) -->
        <param name="useSupportsGetGeneratedKeys" value="false" />
        <connectionSource class="org.apache.log4j.db.DriverManagerConnectionSource">
                <param name="driverClass" value="${driver}" />
                <param name="url" value="${dburl}" />
                <param name="user" value="${username}" />
                <param name="password" value="${password}" />
        </connectionSource>
</appender>
```

EgovJDBCAppender: Singleton is implemented and should be set as Annotation Bean so as to inject the dataSource bean of spring. It extends JDBCAppenderof log4j and JDBCAppenderdirectly creates connection every time depending on connection setting of log4j , but EgovJDBCAppender is the Appender expanded to use by injecting dataSource of spring in the form of Annotation.

```
<appender name="pooledDB"
        class="egovframework.rte.fdl.logging.db.EgovJDBCAppender">
        <!--caller_filename, caller_class, caller_method, caller_line -->
        <param name="locationInfo" value="true" />
        <param name="sql"
                value="INSERT INTO logging_event (
                                                        sequence_number, timestamp,
rendered_message,
                                                       logger_name, level_string, ndc,
                                                        thread_name, reference_flag,
                                                        caller_filename, caller_class,
                                                        caller_method, caller_line)
                                                VALUES
                                                        ('%X{sequence_number}',
'%X{timestamp}', '%X{rendered_message}',
                                                        '%X{logger_name}',
'%X{level_string}', '%X{ndc}',
                                                        '%X{thread_name}',
'%X{reference_flag}',
                                                        '%X{caller_filename}',
'%X{caller_class}',
                                                        '%X{caller_method}',
'%X{caller_line}')" />
</appender>
```

Layout

What kinds of format you will leave the log in? In addition to simple message, many information can be combined including the thread name of target currently logging and logging time. The layout includes HTMLLayout, PatternLayout, SimpleLayout andXMLLayout. While SimpleLayout andXMLLayoutcan be used much, it may be inconvenient to leave log message of the style you want. The layout that I use frequently is PatternLayout and can create various log message combination like printf of C function.

%p: priority such as debug, info, warn, error, fatal is displayed.
%m: contents designated with function such as debug(), info(), warn(), error(), fatal() are displayed.
%d: record the time when logging event occurs. Output format follows the form designated in the brace after %d. It can be used in the form of %d{HH:mm:ss, SSS} or
    %d{yyyy MMM ddHH:mm:ss, SSS}. It can be used in the form of SimpleDateFormat of Java.
%t: display the name of thread from which log event occurs.
%%: used to display %.
%n: platform dependent line feed characteristics are displayed.

Reference

Apache Logging Services Project