{gavatar mode=header}

Required	Version
JSF 2 or later	
Tomcat	7

How JSF 2 is structured in term of logging categories

Find below a list of logger namespaces related to JSF . FacesLogger in com.sun.faces.util package contains an enum of all application loggers

Category	Description
javax.enterprise.resource.webcontainer.jsf.managedbean	Managed bean related logging
javax.enterprise.resource.webcontainer.jsf.context	Faces context related logging
javax.enterprise.resource.webcontainer.jsf.renderkit	Logging in the jsf rendering kit
javax.enterprise.resource.webcontainer.jsf.facelets	Facelets related logging
javax.enterprise.resource.webcontainer.jsf.taglib	Taglib logging
javax.enterprise.resource.webcontainer.jsf.lifecycle	Lifecycle related logging
javax.enterprise.resource.webcontainer.jsf.config	Configuration related logging
javax.enterprise.resource.webcontainer.jsf.timing	Used by Timer class (com.sun.faces.util.Timer)
javax.enterprise.resource.webcontainer.jsf.resource	Resource related logging
javax.enterprise.resource.webcontainer.jsf.application	Application related logging

How to create an application specific logging configuration for JSF

If you want to define your own logging configuration for your web application in Tomcat you need to create a *logging.properties* file in your *src/main/resources* folder for Maven users.

3 Steps to create configure :

1. Define the handlers
2. Configure the handlers
3. Define category levels

Define handlers

Lets define six file handlers :

Handler	Description
0error	All errors
1managedbean	Managed bean related logging
2config	Configuration related logging
3facelets	Facelets related logging
4resource	Resource related logging
5lifecycle	**Lifecycle** related logging

prefix have been added to handler names, so that multiple handlers of a single class may be instantiated. A prefix is a String which starts with a digit, and ends with '.'.
For example, 22foobar. is a valid prefix.

{codecitation class="brush:plain"}
handlers = org.apache.juli.FileHandler, java.util.logging.ConsoleHandler,
 0error.org.apache.juli.FileHandler,
 1managedbean.org.apache.juli.FileHandler,
 2config.org.apache.juli.FileHandler,
 3facelets.org.apache.juli.FileHandler,
 4resource.org.apache.juli.FileHandler,
 5lifecycle.org.apache.juli.FileHandler

.handlers = java.util.logging.ConsoleHandle

{/codecitation}

Configure handlers

Let's define handlers specific configuration for the five file handlers we have defined.As recommended by Tomcat documentation we are using

Juli
file handler. We will store each categories in a separate file located in the
logs
folder of
Tomcat
in a sub folder called Ubiteck according to the path
${catalina.base}/logs/ubiteck.

{codecitation}

```
 0error.org.apache.juli.FileHandler.level = FINEST
0error.org.apache.juli.FileHandler.directory = ${catalina.base}/logs/ubiteck
0error.org.apache.juli.FileHandler.prefix = 00_error.
0error.org.apache.juli.FileHandler.formatter= com.ubiteck.jsf.util.WebAppCustomFormatter

1managedbean.org.apache.juli.FileHandler.level = FINEST
1managedbean.org.apache.juli.FileHandler.directory = ${catalina.base}/logs/ubiteck
1managedbean.org.apache.juli.FileHandler.prefix = 01_managedbean.
1managedbean.org.apache.juli.FileHandler.formatter=
com.ubiteck.jsf.util.WebAppCustomFormatter

2config.org.apache.juli.FileHandler.level = FINEST
2config.org.apache.juli.FileHandler.directory = ${catalina.base}/logs/ubiteck
2config.org.apache.juli.FileHandler.prefix = 02_Config.
2config.org.apache.juli.FileHandler.formatter= com.ubiteck.jsf.util.WebAppCustomFormatter

3facelets.org.apache.juli.FileHandler.level = FINEST
3facelets.org.apache.juli.FileHandler.directory = ${catalina.base}/logs/ubiteck
3facelets.org.apache.juli.FileHandler.prefix = 03_Facelets.
3facelets.org.apache.juli.FileHandler.formatter= com.ubiteck.jsf.util.WebAppCustomFormatter

4resource.org.apache.juli.FileHandler.level = FINEST
4resource.org.apache.juli.FileHandler.directory = ${catalina.base}/logs/ubiteck
4resource.org.apache.juli.FileHandler.prefix = 04_Resource.
4resource.org.apache.juli.FileHandler.formatter= com.ubiteck.jsf.util.WebAppCustomFormatter

5lifecycle.org.apache.juli.FileHandler.level = FINEST
5lifecycle.org.apache.juli.FileHandler.directory = ${catalina.base}/logs/ubiteck
5lifecycle.org.apache.juli.FileHandler.prefix = 05_Lifecycle.
```

5lifecycle.org.apache.juli.FileHandler.formatter= com.ubiteck.jsf.util.WebAppCustomFormatter

{/codecitation}

Define category levels

Let's define what we do we want to see and at which level.

{codecitation}

```
  javax.enterprise.resource.webcontainer.jsf.level=SEVERE
javax.enterprise.resource.webcontainer.jsf.handlers = 0error.org.apache.juli.FileHandler

com.ubiteck.level=SEVERE
com.ubiteck.handlers = 0error.org.apache.juli.FileHandler

##
# Catalina listener start error logging
#

org.apache.catalina.core.level=SEVERE
org.apache.catalina.core.handlers = 0error.org.apache.juli.FileHandler

org.apache.catalina.level=SEVERE
org.apache.catalina.handlers = 0error.org.apache.juli.FileHandler

#
# Managed Bean
#
javax.enterprise.resource.webcontainer.jsf.managedbean.level=FINEST
javax.enterprise.resource.webcontainer.jsf.managedbean.handlers =
1managedbean.org.apache.juli.FileHandler

#
# Config
#
javax.enterprise.resource.webcontainer.jsf.config.level=FINEST
javax.enterprise.resource.webcontainer.jsf.config.handlers = 2config.org.apache.juli.FileHandler
```

```
#
# Facelets
#
javax.enterprise.resource.webcontainer.jsf.facelets.level=FINEST
javax.enterprise.resource.webcontainer.jsf.facelets.handlers =
3facelets.org.apache.juli.FileHandler

#
# Resource
#
javax.enterprise.resource.webcontainer.jsf.resource.level=FINEST
javax.enterprise.resource.webcontainer.jsf.resource.handlers =
4resource.org.apache.juli.FileHandler

#
# Lifecycle
#
javax.enterprise.resource.webcontainer.jsf.lifecycle.level=FINEST
javax.enterprise.resource.webcontainer.jsf.lifecycle.handlers =
5lifecycle.org.apache.juli.FileHandler
```

{/codecitation}

Both the **logger** and its **handlers** have L

Explicitly specify the logging.properties location

If you are running **Tomcat** inside Eclipse or another IDE the JDK logging.properties will be used. The JDK default **logging** file is located in your JDK at *<installation folder>jdk1.6.0_38jrelib*

If you want to explicitly specify the location of your **logging.properties** use the following VM argument :

{codecitation}

-Djava.util.logging.config.file=/<tomcat path>/logging.properties

{/codecitation}

Conclusion

You can now restart your application. The fives logging file and the folder containing the files will be created at start time. You are ready to debug the most complex issues you will face using J2SE and Tomcat.

Logging Levels

The log levels define the severity of a message. The class Level is used to define which messages should be written to the log.

Log Levels in descending order are:

- SEVERE (highest)

- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST

In addition to that you have also the levels OFF and ALL to turn the logging of or to log everything.

For example the following will set the logger to level info which means all messages with severe, warning and info will be logged.

{codecitation} LOGGER.setLevel(Level.INFO); {/codecitation}

Logging Handlers

Each logger can have access to several handler. The handler receives the log message from the logger and exports it to a certain target A handler can be turn off with *setLevel(Level.OFF)* and turned on with
setLevel(...)

You have several standard handler provided by J2SE :

Handler

Description

StreamHandler

A simple handler for writing formatted records to an OutputStream.

ConsoleHandler

A simple handler for writing formatted records to System.err

FileHandler

A handler that writes formatted log records either to a single file, or to a set of rotating log files.

SocketHandler

A handler that writes formatted log records to remote TCP ports.

MemoryHandler

A handler that buffers log records in memory.

Log Levels INFO and higher will be automatically written to the console.

Formatter

Each handlers output can be configured with a formatter

Available formatter

- SimpleFormatter Generate all messages as text
- XMLFormatter Generates XML output for the log messages

You can also build your own formatter.

Tomcat Juli

Although all JDKs ship with logging functionality provided by *java.util.logging*, this default implementation is not designed for container-based environments. To get around this limitation, Tomcat replaces the default LogManager with JULI, a modified implementation with a number of additional features for handling containers.

The most notable of JULI's extended functionalities is the ability to set property files on a per-classloader basis. This feature makes redeployment of web applications significantly less labor-intensive.

Java logging related articles

- Sample log4j.properties file

Tags: related , logging , tomcat , file , define , handler , handlers , levels