Logging Tutorial

Primer On java.util.logging and JULI

The goal of this primer is to demonstrate how to log with java.util.logging, as implemented by JULI, as a backdrop to the rest of the tutorial, in particular the sections related to configuring Tomcat logging with the configuration file. One may think it's wise to skip this section and proceed directly to the section on the configuration file, but all the concepts talked about here are necessary to understand the configuration file. The examples used in this section show programmatically how logging is configured. The section on Tomcat's configuration file discusses how to accomplish declaratively what is done programmatically in this section.

The following concepts (Classes) are important:

- LogManager
- Loggers
- Handlers
- Levels
- Root Logger
- Root Handler

The official Tomcat logging documentation refers to the above concepts / classes extensively.

Lets start with Loggers. What's the purpose of a Logger? A Logger is what a developer uses to write log statements to the console, to a file, to the network, etc. If you wanted to log something from your web application's class CriticalComponent using java.util.logging you would first create a logger like this:

```
private String nameOfLogger = 'com.example.myapp.CriticalComponent';

or

private String nameOfLogger = CriticalComponent.class.getName();

private static Logger myLogger = Logger.getLogger(nameOfLogger);
```

Pay attention to how we defined the name of the Logger. This is important to the material explaining Tomcat's logging configuration. You may also want to think about why myLogger is a static field (Hint myLogger is shared among all instances of CriticalComponent).

Now you have a logger to create logging messages from your class CriticalComponent. For example you could now try something like this:

```
public void wasssup()
{
myLogger.info("Ah Yeah Baby - that's the end of System.out.println");
}
```

If you ran this code with a deployed web application you would see this statement on the console or in catalina.out:

```
INFO; 322125105255ms; 4407662;# 1; com.example.myapp.CriticalComponent; wasssup; Ah Yeah Baby - that's the end of System.out.println
```

Notice that both the name of the Logger and the method that logged the message are mentioned in the log statement. This is important to know when you want to alter the configuration of the Logger. For example you might want to turn this logger off, because it's not that useful.

What if you wanted the output to appear in a file and on the console? For that you need to define 2 Handlers. Create the two like this:

```
Handler fileHandler = new FileHandler("/var/log/tomcat/myapp.log");

Handler consoleHandler = new ConsoleHandler();

myLogger.addHandler(fileHandler);

myLogger.addHandler(consoleHandler);
```

Now myLogger will log to both the console and the file /var/log/tomcat/myapp.log.

So now we understand Loggers and Handlers, but we have not touched on Levels, Root Loggers, and Root Handlers yet. What are those?

Lets start with a root Logger. A root logger is a logger whose name is "". What's the purpose of it? Suppose you tried to do some logging with myLogger like this:

```
myLogger.finest("Sooooo Fine");
```

And nothing shows up in your log, but you know that this statement is being called. What's going on? The answer is that the JULI root logger's Level is set to INFO by default. The level INFO has a corresponding integer assigned to it, which is 3.
When myLogger attempts to log it first checks its level. If myLogger's level is greater than the level intrinsic to the method doing the logging (finest), then the record will be logged. In this case the logging method finest corresponds to Level zero. Zero is less than 3, hence the logger does not log the message. My logger will only log messages with a level that is greater than or equal to 3. So for instance if

```
myLogger.severe("Oohhh %#$@#$!!!")
```

is called, it will get logged because the level intrinsic to the method severe is greater than 3. So you are following this, but wondering how myLogger's level got set to 3 (INFO), since we never explicitly set it. The answer is that it comes from the root Logger.

Now suppose you created a logger named 'com.example.myapp' and set its level. Would myLogger still get its level from the root Logger. It would not.

The reason is that the logger named 'com.example.myapp' is now a parent logger to myLogger, and myLogger gets its level from it instead. How did the 'com.example.myapp' Logger become a parent? It's because of its name 'com.example.myapp'. If it were named 'org.charity.generous', it would not be a parent logger. Do you see the pattern (myLogger = Logger.getLogger('com.example.myapp.CriticalComponent') includes 'com.example.myapp' in its name)?

Now that we mulled that over you have an idea of what a level is as well. The Level determines what gets logged. You can set the level directly on myLogger like this:

myLogger.setLevel(Level.WARNING);

In this case only messages with a level of WARNING or SEVERE will get logged by myLogger.

Now remember that once something gets logged by myLogger, it's handed over to myLogger's Handlers. But what if we never assigned any Handlers to myLogger. How would it get its Handler?

The answer is the root Handlers. The root logger has root Handlers. So if you don't explicitly define any Handlers for myLogger, it will use the Handlers attached to the root Logger. Now if you did attach Handler(s) to the logger named 'com.example.myapp', but not to myLogger, then myLogger would use the Handler(s) attached to 'com.example.myapp', instead of the Handlers attached to the root Logger. In other words myLogger uses the Handlers of the closest Logger ancestor, which in this case is either the 'com.example.myapp' Logger or the root Logger.

So all of the above is important to understand in order to be able to configure Tomcat logging with JULI. Now we can talk about doing so via the configuration file.

Configuring Tomcat JULI

Configuring Loggers

Suppose we wanted to configure myLogger via a logging.properties file, rather than programmatically as we did earlier, and simply set its Level to SEVERE.
We would put the following in the configuration file:

```
com.example.myapp.CriticalComponent.level = SEVERE
```

That's how Loggers are configured. We specified the name of the Logger followed by a period and the name of the property we wanted to set, which is 'level' in this case. If we had several classes with corresponding loggers, and we did not want to set the logging level for each class individually, we could set the logging level on the root Logger like this:

```
.level = SEVERE
```

This will set the level on all loggers to SEVERE, unless the Logger uses a non-default level (Either programmatically or via configuration).

So if you wished to turn logging off all together simply configure the root logger like this and make sure that there are no child loggers that override this setting:

```
.level = OFF
```

On the flip side to log everything set it like this:

```
.level = ALL
```

Configuring Handlers

Suppose that we also wanted to add a Handler to the Logger named com.example.myapp.CriticalComponent.

We could define our handler like this:

handlers = org.apache.juli.FileHandler

And configure it like this:
org.apache.juli.FileHandler.level = WARNING org.apache.juli.FileHandler.directory = /var/log/tomcat/
org.apache.juli.FileHandler.prefix = mywebapp

Then assign it to our Logger like this:

com.example.myapp.CriticalComponent.handlers = org.apache.juli.FileHandler

Now we've done the same thing declaratively in our logging.properties file that we did programmatically earlier.

What if we wanted to make the Handler we just configured the root Handler? We could configure it as the root Handler like this:
.handlers = org.apache.juli.FileHandler.

Now all Tomcat Loggers that do not have a Handler assigned to them directly, either programmatically or declaratively via the logging.properties file, will delegate to this Handler.

Configuring Context Loggers

So now you've had a look at the example logging.properties file in the official documentation again, and you understand most of it.

Then you get to these lines:

---

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].level = INFO org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[
/manager].handlers = \
3manager.org.apache.juli.FileHandler

These don't follow the same syntax we've been using thus far to configure Loggers and Handlers.

These are for configuring Context loggers. What's a context logger? First you need to know a little about the ServletContext.

For a description of what a Servlet Context is see:

---

http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/ServletContext.html

---

You'll notice specifically that it mentions that there is one ServletContext per webapp, and that this ServletContext can be used to perform logging.

Looking down a little further in the ServletContext.html document we see that the ServletContext has a log method that takes a String.

If a developer working on the manager application were to use the ServletContext.log method, without the corresponding configuration lines shown above, output would go to standard out, and end up on the console or catalina.out. However now that the ServletContext Logger has been configured for the manager application, ServletContext.log("...") statements made within the /manager application will end up in manager.dd.mm.yyy.log.

CategoryFAQ