SolrLogging

Solr's Logging Mechanism

Starting with Solr 1.4, the Solr Code base compiles against the SLF4J API.

The Solr Admin console has a screen for changing the logging level globally. This is a transient setting good for doing diagnostic work, but does not persist after restart.

Solr 4.3 and above

What changed

These versions do not include any logging jars in the WAR file. They must be provided separately. The Solr example for these versions includes jars (in the jetty lib/ext directory) that set up SLF4J with a binding to the Apache log4j library.

Why did it change?

The logging setup was changed for increased flexibility. With older versions, changing your logging mechanism required either building a special target from the source code or doing surgery on the WAR file. Now anyone can change to another logging mechanism or upgrade to newer component versions simply by changing jar files.

A note about SLF4J intercept jars

Some of the third-party software components in Solr do not use the SLF4J library for their logging, they use one of the other logging frameworks directly. SLF4J includes various 'intercept' jars that grab the output of these direct calls and send them through SLF4J so that all your logs end up handled by the same configuration. The logging frameworks that must be dealt with are java.util.logging (JUL), java.commons.logging (JCL), and log4j.

The default SLF4J logging destination for the 4.3 example is log4j, so the example does not need to intercept calls for that framework. They will be handled by the real log4j jar. It only needs to intercept JUL and JCL, as described above. This is why the jul-to-slf4j and jcl-over-slf4j jars are included in the 4.3 example. If you change your logging framework from log4j to JUL, then you must remove the intercept jar for JUL and add the intercept jar for log4j, which is named log4j-over-slf4j. If you use an entirely different logging mechanism like logback, then you must have intercepts for all three - JUL, JCL, and log4j.

Using the example logging setup in containers other than Jetty

To get the same logging setup in another container (Tomcat for example) as with the example Jetty server, you need to do the following

1. Copy the jars from `solr/example/lib/ext` into your container's main lib directory. These jars will set up SLF4J and log4j.
   a. Exactly where this lib directory lives is highly variable. For a Debian or Ubuntu server using the Tomcat package available from the OS vendor, this is likely to be `/usr/share/tomcat6/lib` or `/usr/share/tomcat7/lib`.
2. Copy the logging config from `solr/example/resources/log4j.properties` into a location on the classpath. Usually you can use the same location as the jar files above. Edit the configuration file for your preferred log destination.
3. Optionally, if you did not place `log4j.properties` on the classpath, set the "log4j.properties" system property. Examples for this:
   - Linux/UNIX: `-Dlog4j.configuration=file:///path/to/log4j.properties`
   - Windows: `-Dlog4j.configuration=file:///c:/solr/resources/log4j.properties`

If the system can not find the jars for logging, you may get an error which prevents Solr from deploying. For instance in Tomcat 6, it might look like this:

```
INFO: Deploying web application archive solr.war
May 29, 2013 3:59:40 PM org.apache.catalina.core.StandardContext start
SEVERE: Error filterStart
May 29, 2013 3:59:40 PM org.apache.catalina.core.StandardContext start
SEVERE: Context [/solr] startup failed due to previous errors
```

If the system cannot find your logging configuration, you may get errors like this:

```
log4j:WARN No appenders could be found for logger (org.apache.solr.servlet.SolrDispatchFilter).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

It might help to start the server with `-Dlog4j.debug=true` to see more details.

Switching from Log4J back to JUL (java.util.logging)

The example logging setup takes over the configuration of Solr logging, which prevents the container from controlling where logs go. Users of containers other than the included Jetty (Tomcat in particular) may be accustomed to doing the logging configuration in the container. If you want to switch back to java.util.logging so this is once again possible, here's what to do. These steps apply to the example/lib/ext directory in the Solr example, or to your container's lib directory as mentioned in the previous section. These steps also assume that the slf4j version is 1.6.6, which comes with Solr4.3. Newer versions may use a different slf4j version. As of May 2013, you can use a newer SLF4J version with no trouble, but be aware that all slf4j components in your classpath must be the same version.

1. Download slf4j version 1.6.6 (the version used in Solr4.3.x). http://www.slf4j.org/dist/slf4j-1.6.6.zip 2. Unpack the slf4j archive. 3. Delete these JARs from your lib folder: slf4j-log4j12-1.6.6.jar, jul-to-slf4j-1.6.6.jar, log4j-1.2.16.jar 4. Add these JARs to your lib folder (from slf4j zip): slf4j-jdk14-1.6.6.jar, log4j-over-slf4j-1.6.6.jar 5. Use your old logging.properties

Solr 1.4 to 4.2.1

These versions include SLF4J jars in the WAR File that bind SLF4J to JDK standard logging (JUL, the java.util.logging classes). An overview of how JUL can be configured at the JVM level can be found here:

http://java.sun.com/j2se/1.5.0/docs/guide/logging/overview.html

Many servlet containers also provide alternate log configuration options in their configuration files. You should consult your servlet container documentation to see what options are available.

Users who want an alternate logging implementation (log4j, logback etc) will need to repackage the .war file to remove the existing SLF4J jar files and replace them with the correct jar files. You can do this by extracting and repacking the .war file, or by building the war file from source without the jar files.

⚠ From Solr3.6 there is a build target `ant dist-war-excl-slf4j` which will automatically package the WAR file with only the slf4j-api JAR and no bindings. Using this war file you can place your slf4j-XXX.jar binding file of choice in your servlet container external lib folder. Note that the slf4j jar(s) that you add must match the version number of the slf4j-api jar file that is contained within the war. For Solr 3.6, that version is 1.6.1. For Solr 4.0, it is 1.6.4.

⚠ From Solr4.1 there are two build targets related to choosing your own slf4j binding. These are `ant dist-excl-slf4j` and `ant dist-war-excl-slf4j`. The first target builds the usual dist files in addition to the WAR file, the second just builds the WAR file. These build targets will package the WAR without any slf4j jars at all, so you must include all relevant slf4j jars in your container external lib directory. These build targets were removed from Solr4.3 and later.

Here are the slf4j jars that are in the WAR file when you build it without one of the special targets:

- slf4j-api-1.6.1.jar
- slf4j-jdk14-1.6.1.jar
- jcl-over-slf4j-1.6.1.jar
- log4j-over-slf4j-1.6.1.jar

The first file is the primary slf4j library. The second is the slf4j binding - in this case . Some of Solr's dependent components use other logging methods - jcl and log4j. The remaining jar files above intercept calls to these other logging methods and inject them into slf4j.

When you change the slf4j binding, you must include the primary library (the -api jar), the binding for the logging method that you wish to use, the jar(s) for that logging method, and relevant "-over-slf4j" jars. Note that if you choose either jcl or log4j for your binding, you must leave out the matching "-over-slf4j" jar.

More Info: http://www.slf4j.org/

Solr 1.0 to 1.3

These versions use JDK standard logging directly, not through SLF4J.

Customizing Logging

With Example Jetty Instance

If you're using the example from Solr 4.3.0 and later, you can simply edit the log4j.properties file in the resources directory to customize the logging. There is a lot of information on the Internet regarding how to configure log4j, including the introduction. The included config file logs to a file as well as the console.

Earlier versions do not come with a logging config file. See LoggingInDefaultJettySetup for an example of how you can configure logging with 4.2.1 and earlier.

Using log4j with Solr from source, 4.2.1 or earlier

Prior to nightly version], using a different SLF4J binding (such as log4j) meant either repackaging the war file or building it yourself from source. To build it yourself, you need to obtain a [NightlyBuilds or get it from svn. Once it's downloaded, extract it, cd to the solr directory in the extracted directory, then compile it using `ant dist-war-excl-slf4j` or `and dist-excl-slf4j` depending on the version.

Here are the jars you will need to put in your container external lib folder, assuming that you get the 1.7 version of slf4j. The last one comes from the log4j website, not slf4j:

- slf4j-api-1.7.5.jar
- slf4j-log4j12-1.7.5.jar
- jcl-over-slf4j-1.7.5.jar
- log4j-1.2.17.jar

http://www.slf4j.org/download.html http://logging.apache.org/log4j/1.2/

Note that log4j must be configured before it will work. One way to do this is to pass an argument like the following to java:

```
-Dlog4j.configuration=file:etc/log4j.properties
```

http://logging.apache.org/log4j/1.2/manual.html#defaultInit

Sample log4j.properties file:

```
#  Logging level

log4j.rootLogger=WARN, file


#- size rotation with log cleanup.

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.MaxFileSize=4MB

log4j.appender.file.MaxBackupIndex=9


#- File to log to and log format

log4j.appender.file.File=logs/solr.log

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%-5p - %d{yyyy-MM-dd HH:mm:ss.SSS}; %C; %m\n
```

Using Logback with the Solr 3.5 binary distribution

Here are some details for implementing logback (http://logback.qos.ch/). As you may know, Logback was created by the original author of Log4j. It has several enhancements including Filters and Custom Appenders.

1. Get Solr 3.5 http://www.apache.org/dyn/closer.cgi/lucene/solr/3.5.0 and extract the tar/zip file on your system.

Note: `$ORIG` is your top level directory for Solr.

```
# cd $ORIG

# tar xzvf apache-solr-3.5.0.tgz
```

2. Copy the solr.war file to a new location

```
# mkdir -p /tmp/solr

# cd ./example/webapps

# cp solr.war /tmp/solr

# cd /tmp/solr

# jar xvf solr.war

# rm -f solr.war
```

3. Remove the slf4j-jdk14-1.6.1.jar file

```
# cd /tmp/solr

# rm -f ./WEB-INF/lib/slf4j-jdk14-1.6.1.jar
```

4. Copy logback classic files

Get the file: http://logback.qos.ch/dist/logback-1.0.1.tar.gz

```
# tar xzvf logback-1.0.1.tar.gz

# cd logback-1.0.1

# cp logback-classic-1.0.1.jar /tmp/solr/WEB-INF/lib

# cp logback-core-1.0.1.jar /tmp/solr/WEB-INF/lib
```

Note: If you wanted to upgrade SLF4J from 1.6.1 to 1.6.4 you would do the following...

1. Get the SLF4J from http://www.slf4j.org/dist/slf4j-1.6.4.tar.gz
   b. Extract the tar into a temporary directory
   c. delete the old versions:

```
# rm /tmp/solr/WEB-INF/lib/jcl-over-slf4j-1.6.1.jar

# rm /tmp/solr/WEB-INF/lib/log4j-over-slf4j-1.6.1.jar

# rm /tmp/solr/WEB-INF/lib/slf4j-api-1.6.1.jar
```

d. Copy the new ones

```
# cp jcl-over-slf4j-1.6.4.jar /tmp/solr/WEB-INF/lib

# cp log4j-over-slf4j-1.6.4.jar /tmp/solr/WEB-INF/lib

# cp slf4j-api-1.6.4.jar /tmp/solr/WEB-INF/lib
```

5. Rebuild the war

```
# cd /tmp/solr

# jar cvf solr.war *
```

6. Copy the war back to your example/webapps directory

```
# cp solr.war $ORIG/apache-solr-3.5.0/example/webapps
```

7. Create resources/logback.xml file

```
# cd $ORIG/apache-solr-3.5.0/example

# mkdir resources
```

copy logback.xml into this directory.

8. Restart Solr 3.5

Example logback.xml.

```
<configuration debug="true">

  <appender name="FILE" class="ch.qos.logback.core.FileAppender">

    <file>/var/log/solr.log</file>

    <encoder>

        <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

    </encoder>

  </appender>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

    <encoder>

      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

    </encoder>

  </appender>

  <logger name="org.apache.solr.handler.dataimport.DocBuilder.level" level="SEVERE" additivity="false">

    <appender-ref ref="FILE" />

  </logger>

  <logger name="org.apache.solr.handler.dataimport.ThreadedEntityProcessorWrapper.level" level="SEVERE"
additivity="false">

    <appender-ref ref="FILE" />

  </logger>

  <logger name="org.apache.solr.core.SolrCore" level="INFO" additivity="false">

        <appender-ref ref="FILE" />

  </logger>

  <logger name="org.apache.solr" level="INFO" additivity="false">

    <appender-ref ref="FILE" />

  </logger>

  <root level="OFF">

    <appender-ref ref="STDOUT" />

  </root>

</configuration>
```

SolrJ and Logging Jars

SolrJ also requires additional jars for logging, just like Solr does. These jars are not included in dist/solrj-lib along with the rest of SolrJ's dependencies. The reason is similar to the situation with Solr – the Solr project does not dictate what logging framework the user must use. By choosing which logging jars to include, you can make the decision about logging framework yourself.

If you just want to get SolrJ working quickly, you can copy the jars from lib/ext, the log4j.properties file from resources, and the SolrJ dependencies from dist/solrj-lib into your project's classpath. SolrJ will have all the dependencies that it needs.