http://tomcat.apache.org/tomcat-5.5-doc/logging.html

・jajakarta ドキュメントには、2007-06-07 現在のところ tomcat-5.5 のドキュメントはない。

The Apache Tomcat 5.5 Servlet/JSP ContainerApache Tomcat

Logging in Tomcat

■ Introduction

■概要

Tomcat 5.5 uses Commons Logging throughout its internal code allowing the developer to choose a logging configuration that suits their needs, e.g java.util.logging or Log4J.

Commons Logging provides Tomcat the ability to log hierarchially across various log levels without needing to rely on a particular logging implementation.

Tomcat 5.5 では､開発者が必要に応じて (例 : java.util.logging や Log4J) ログ取得の設定を選択できるように、内部コードでも Commons Logging を使用しています。Commons Logging は、特定のログ実装に依存しなくても良いようなやり方で Tomcat にログの階層化管理能力を与えます。

An important consequence for Tomcat 5.5 is that the <Logger> element found in previous versions to create a localhost_log is no longer a valid nested element of <Context>.

Instead, the default Tomcat configuration will use java.util.logging. If the developer wishes to collect detailed internal Tomcat logging (i.e what is happening within the Tomcat

engine), then they should configure a logging system such as java.util.logging or log4j as detailed next.

Tomcat 5.5 への大きな影響としては、前のバージョンで <Context> 要素に含まれた <Logger> 要素は機能しなくなったことです。代わりに、デフォルトの Tomcat 設定は java.util.logging を使用します。開発中に Tomcat 内部の詳細なログを採りたい (例 :Tomcat engine で何が起きているか) なら、次のように

java.util.logging や log4j を設定すればよいでしょう。

■ log4j

Tomcat 5.5 has done away with localhost_log which you may be familiar with as the runtime exception/stack trace log. These types of error are usually thrown by uncaught

exceptions, but are still valuable to the developer. They can now be found in the stdout log.

Tomcat 5.5 では localhost_log としていつものランタイムの例外やスタックトレースのログを設定してあります。捕捉されなかった例外として発生するエラーですが、重要な情報です。 stdout のログに記録されます。

If you need to setup cross-context detailed logging from within Tomcat's code, then you can use a simple log4j configuration. Note that this logging van be very verbose depending

on the log level you chose to use. Note also that a log4j logging configuration is not going to produce stack trace type logging: those stack traces are output to stdout as

discussed above.

Tomcat から利用されるコンテキストをまたがって詳細なログを設定する必要があるなら、簡単な log4j 設定を使いましょう。このログ方法は設定によっては非常に大量の情報を出力することを覚え

ておいてください。また、 log4j のログ設定はスタックとレースのログ出力は生成しないことにも気をつけてください。スタックとレースは stdout に出力されます。

Follow the following steps to setup a file named tomcat.log that has internal Tomcat logging output to it:

下記の手順で Tomcat 内部のログを tomcat.log ファイルに出力することができます。

1.Create a file called log4j.properties with the following content and save it into common/classes.

・log4j.properties の名前で 下記の内容のファイルを common/classes に作成してください。

```
log4j.rootLogger=DEBUG, R
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=${catalina.home}/logs/tomcat.log
log4j.appender.R.MaxFileSize=10MB
log4j.appender.R.MaxBackupIndex=10
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

1.Download Log4J (v1.2 or later) and place the log4j jar in $CATALINA_HOME/common/lib.
・Log4J (v1.2 以降) をダウンロードし、 log4j の jar ファイルを $CATALINA_HOME/common/lib
2.Download Commons Logging and place the commons-logging.jar (not commons-logging-api.jar) in $CATALINA_HOME/common/lib with the log4j jar.
・Commons Loging をダウンロードし、commons-loggig.jar ファイルを log4j. $CATALINA_HOME/common/lib
3.Start Tomcat
・Tomcat を起動

This log4j configuration sets up a file called tomcat.log in your Tomcat logs folder with a maximum file size of 10MB and up to 10 backups. DEBUG level is specified which will

result in the most verbose output from Tomcat. この log4j 設定によって、Tomcat の logs フォルダに最大サイズ 10M で 10 回分まで保存される tomcat.log というファイルが生成されます。 DEBUG レ

ベルを指定することで Tomcat のログ出力を最

も詳細に行います。

You can (and should) be more picky about which packages to include in the logging. Tomcat 5.5 uses defines loggers by Engine and Host names. For example, for a default Catalina

localhost log, add this to the end of the log4j.properties above. Note that there are known issues with using this naming convention (with square brackets) in log4j XML based

configuration files, so we recommend you use a properties file as described until a future version of log4j allows this convention. ログを取得する場合は、パッケージを詳しく指定して行えますし、行うべきです。Tomcat 5.5 では Engine と Host の名前で logger を定義し使用しています。例えば、上記の log4j.properties ロ

グファイルに下記の設定を追加することでデフォルトの Catalina localhost のログを取得できます。角括弧を使用した名前付けルールでは、log4j の XML での設定ファイルに既知の問題が生じるの

で、log4j の将来のバージョンで問題が解決されるまでは properties ファイルでの設定を利用することをお勧めします。

```
log4j.logger.org.apache.catalina.core.ContainerBase.[Catalina].[localhost]=DEBUG, R
log4j.logger.org.apache.catalina.core=DEBUG, R
log4j.logger.org.apache.catalina.session=DEBUG, R
```

Be warned a level of DEBUG will produce megabytes of logging and slow startup of Tomcat. This level should be used sparingly when debugging of internal Tomcat operations is

required. DEBUG レベルでは数メガバイトのログファイルが出力されたり、Tomcat の起動が遅くなるので注意してください。このレベルを使うのは Tomcat 内部の動作をデバッグしなければならないときなど控え

めに使いましょう。

Your web applications should certainly use their own log4j configuration. This is valid with the above configuration. You would place a similar log4j.properties file in your web

application's WEB-INF/classes folder, and log4j1.2.8.jar into WEB-INF/lib. Then specify your package level logging. This is a basic setup of log4j which does *not* require

Commons-Logging, and you should consult the log4j documentation for more options. This page is intended only as a bootstrapping guide. あなたの web アプリケーションは独自の log4j 設定を使用するべきです。上述の設定を同時に行っても有効です。似たような log4j.properties ファイルを web アプリケーションの WEB-INF/clases

フォルダに配置し、 log4j1.2.8.jar は WEB-INF/lib に配置すると良いでしょう。設定ファイル内でパッケージのログのレベルを指定してください。

java.util.logging

- In order to configure JDK logging you should have JDK 1.4+. Tomcat 5.5 is intended for JDK 5.0 or later, but can be run on JDK 1.4 using a compatibility package.
- JDK でのログ記録を利用するなら、JDK は 1.4 以降が必要です。Tomcat 5.5 は JDK 5.0 以降を想定していますが、互換パッケージを利用すると JDK 1.4 ででも利用できます。

- The default implemenatation of java.util.logging provided in the JDK is too limited to be useful. A limitation of JDK Logging appears to be the inability to have per-web

application logging, as the configuration is per-VM. As a result, Tomcat will, in the default configuration, replace the default LogManager implementation with a container

friendly implementation called JULI, which addresses these shortcomings. It supports the same configuration mechanisms as the standard JDK java.util.logging, using either a

programmatic approach, or properties files. The main difference is that per-classloader properties files can be set (which enables easy redeployment friendly webapp

configuration), and the properties files support slightly extended constructs which allows more freedom for defining handlers and assigning them to loggers.

- JULI is enabled by default in Tomcat 5.5, and supports per classloader configuration, in addition to the regular global java.util.logging configuration. This means that logging

can be configured at the following layers:

- Tomcat 5.5 では JULI が有効になっていて、一般的な全体で利用する java.util.logging での設定だけでなく、クラスローダ別の設定が可能になっています。これは下記のような階層で設定が可能

だということです。

In the JDK's logging.properties file. Check your JAVA_HOME environment setting to see which JDK Tomcat is using (or maybe JRE 5.0 as Tomcat can now run on a JRE from version 5.5).

The file will be in $JAVA_HOME/jre/lib. Alternately, it can also use a global configuration file located elsewhere by using the system property java.util.logging.config.file, or

programmatic configuration using java.util.logging.config.class. In each classloader using a logging.properties file. This means that it is possible to have a configuration for the Tomcat core, as well as separate configurations for each

webapps which will have the same lifecycle as the webapps. JDK での logging.properties ファイルでの設定。JAVA_HOME 環境変数を調べて Tomcat がどの JDK を使用しているか調べてください。(Tomcat は 5.5 から JRE で動作するようになったので、JRE

5.0 かもしれません。) logging.properties ファイルは JAVA_HOME/jre/lib に入っているでしょう。代わりに java.util.logging.config.file システムプロパティや

java.util.logging.config.class を使って設定することもできます。クラスローダ別に logging.properties ファイルが使用できます。Tomcat コアの設定を使ったり、web アプリケーション別の設定ファイル (web アプリケーションと同じライフサイクル) を利用できます

。

The default logging.properties specifies a ConsoleHandler for routing logging to stdout and also a FileHandler. A handler's log level threshold can be set using SEVERE, CONFIG,

INFO, WARN, FINE, FINEST or ALL. The logging.properties shipped with JDK is set to INFO. You can also target specific packages to collect logging from and specify a level. Here is

how you would set debugging from Tomcat. You would need to ensure the ConsoleHandler's level is also set to collect this threshold, so FINEST or ALL should be set. Please refer to

Sun's java.util.logging documentation for the complete details.

デフォルトの logging.properties ファイルでは、標準出力と FileHandler の出力先に ConsoleHandler を使うよう指定されています。ハンドラのログ出力レベルには SEVERE, CONFIG, INFO, WARN,

FINE, FINEST や ALL が指定できます。JDK に含まれる logging.properties では INFO に設定されています。どのパッケージからログを取得するかを指定したり、ログ出力レベルを指定できます。

下記は Tomcat でのデバッグ時の設定方法です。

```
org.apache.catalina.level=FINEST
```

The configuration used by JULI is extremely similar, but uses a few extensions to allow better flexibility in assigning loggers. The main differences are:

A prefix may be 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. As in Java 5.0, loggers can define a list of handlers using the loggerName.handlers property. By default, loggers will not delegate to their parent if they have associated

handlers. This may be changed per logger using the loggerName.useParentHandlers property, which

accepts a boolean value. The root logger can define its set of handlers using a .handlers property. System property replacement for property values which start with ${sytstemPropertyName}.

Example logging.properties file to be placed in common/classes:

```
handlers = 1catalina.org.apache.juli.FileHandler, 2localhost.org.apache.juli.FileHandler, \
          3manager.org.apache.juli.FileHandler, 4admin.org.apache.juli.FileHandler, \
          java.util.logging.ConsoleHandler
```

.handlers = 1catalina.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler

Handler specific properties.# Describes specific configuration info for Handlers.##

1 catalina.org.apache.juli.FileHandler.level = FINE1 catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs1catalina.org.apache.juli.FileHandler.prefix = catalina.

2localhost.org.apache.juli.FileHandler.level = FINE2localhost.org.apache.juli.FileHandler.directory = ${catalina.base}/logs2localhost.org.apache.juli.FileHandler.prefix = localhost.

3manager.org.apache.juli.FileHandler.level = FINE3manager.org.apache.juli.FileHandler.directory = ${catalina.base}/logs3manager.org.apache.juli.FileHandler.prefix = manager.

4 admin.org.apache.juli.FileHandler.level = FINE4 admin.org.apache.juli.FileHandler.directory = ${catalina.base}/logs4admin.org.apache.juli.FileHandler.prefix = admin.

java.util.logging.ConsoleHandler.level = FINEjava.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

Facility specific properties.# Provides extra control for each logger.##

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFOorg.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers = \

```
2localhost.org.apache.juli.FileHandler
```

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].level = INFOorg.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].handlers = \

```
3manager.org.apache.juli.FileHandler
```

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/admin].level = INFOorg.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/admin].handlers = \

```
4admin.org.apache.juli.FileHandler
```

For example, set the com.xyz.foo logger to only log SEVERE# messages:#org.apache.catalina.startup.ContextConfig.level = FINE#org.apache.catalina.startup.HostConfig.level = FINE#org.apache.catalina.session.ManagerBase.level = FINE

Example logging.properties for the servlet-examples web application to be placed in WEB-INF/classes inside the web application:

```
handlers = org.apache.juli.FileHandler, java.util.logging.ConsoleHandler
```

Handler specific properties.# Describes specific configuration info for Handlers.##

org.apache.juli.FileHandler.level = FINEorg.apache.juli.FileHandler.directory = ${catalina.base}/logsorg.apache.juli.FileHandler.prefix = servlet-examples.

java.util.logging.ConsoleHandler.level = FINEjava.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

Handler Properties

Tomcat's JULI implementation is not intended to be a fully-featured logging libary, only a simple bridge to those libraries. However, JULI does provide several properties for

configuring the its handlers. These are listed below.

FileHandler

■ Attribute Description

■属性の説明

・directory The directory where the log file will be written. The Tomcat server account should have write permissions to this directory. The default value of this property is logs.
・directory: ログファイルを出力するディレクトリを指定します。 Tomcat サーバの実行アカウントはこのディレクトリへの書き込み権限が必要です。デフォルト値は logs です。

・prefix The log file name prefix. This is the portion of the log file name before the date. The default value of this property is juli..
・prefix: ログファイルの接頭名です。ログファイル名のうち、日付の前に付く部分です。デフォルト値は juli. です。

・suffix The log file name suffix. This is the portion of the log file name after the date. The default value of this property is .log.

・level The threshold level for this handler. It must be one of the levels in the java.util.logging.Level class. The default value of this property is ALL. Messages whose level is below the specified level will not be written to the file.

・filter The fully-qualified class name of a class that implements the java.util.logging.Filter interface. JULI will load this class and associate it with this handler to filter its messages. By default, there is no Filter associated with the handler.

・formatter The fully-qualified class name of a class that implements the java.util.logging.Formatter interface. JULI will load this class and associate it with this handler to format its messages. By default, there is no Formatter associated with the handler.