Enhydra API's

Igor Smirnov

Table of Contents

Chapter 1. Logging API

Introduction

Enhydra Enterprise supports *three* different ways of logging. Logger interface defines abstract level of Enhydra logging (Logger and LogChannel interfaces) in eaf_api.jar (EAFApi module).

Old **Standard** logging system (to file and to console) is default and it is defined in application conf file by LogFile, LogToFile and LogToStderr parameters.

Also new Enhydra supports **log4j-logging**. Implementations of Logger and LogChannel classes with log4j support are included in EAF.

Monolog is used in Jonas for logging. So, EAF contains Monolog implementation of Enhydra Logger. Implementations of Logger and LogChannel classes with log4j support are included in eafmonolog.jar.

Classes and Interfaces

LogChannel class is an interface of a channel associated with a logging facility. All messages for the facility are written using a channel.

Logger is a general-purpose logging facility. It is modeled after *syslogd*. This is a base class from which an actual implementation is derived.

LogWriter is a class used to write log output to a particular LogChannel and level. This class is PrintWriter, with println() causing a write.

Implementations of *LogChannel* interface:

- com.lutris.logging.StandardLogChannel (default value - for standard logging system)
- com.lutris.logging.Log4jLogChannel (for log4j logging)
- com.lutris.logging.MonologLogChannel (for monolog logging)

Extensions of *Logger* class:

- com.lutris.logging.StandardLogger (default value - for standard logging system)
- com.lutris.logging.Log4jLogger (for log4j logging)
- com.lutris.logging.MonologLogger (for monolog logging)

Extensions of *LogWriter* class:

- com.lutris.logging.Log4jLogWriter (for log4j logging)
- com.lutris.logging.MonologWriter (for monolog logging)

Standard Logger Configuration

Standard logging initialization

Standard logging in is initialized by default, in case no LogClassName parameter is specified in the application's configuration file. The following is the list of parameters one can specify for a standard logger:

Server.LogFile -- This is the file where the server log is written

Server.LogToFile[] -- This is a comma separated list of message types to send to the log file specified in server.logFile

Server.LogToStderr[] -- This is a comma separated list of message types to send to standard error. Possible logger levels:

- EMERGENCY -- Panic condition.
- ALERT -- A condition that should be corrected immediately, such as database corruption.
- CRITICAL -- Critical conditions such as had device errors.
- ERROR -- General errors that are not usually fatal, but must be resolved.
- WARNING -- Warning condition that may need attention, although the need is not immediate.
- NOTICE -- Conditions that are not error conditions, but may require special handling such as infrequent conditions.
- INFO -- General informational conditions, knowledge of which will help to keep the server in good order.
- DEBUG -- Messages that contain information normally of use only when debugging an application.
- CLASSLOAD -- Information about the loading of application classes. Very useful debugging class path problems.
- REQUEST -- The StandardLoggingFilter logs hits to this facility if this is specified (normally it writes to it's own file).
- XMLC -- Information about auto-compiling XMLC pages.
- XMLC_DEBUG -- Debug information about auto-compiling XMLC pages.

An example:

```
Server.LogFile = "<enhydra_enterprise_root>/multiserver/logs/enhydra.log"
Server.LogToFile[] = EMERGENCY, ALERT, CRITICAL, ERROR, WARNING, INFO
Server.LogToStderr[] = EMERGENCY, ALERT, CRITICAL, ERROR, WARNING, INFO
```

Log4j Logger Configuration

In order to initialize log4j it is mandatory to specify the following parameter in the configuration file:

```
LogClassName=com.lutris.logging.Log4jLogger
```

In addition, a log4j-config paramer should be defined and point to a log4j.xml configuration file. Detailed specification can be found at: http://jakarta.apache.org/log4j/docs/documentation.html.

Log4j supports following logging levels: ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF.

More information about logging and logging configuration at: Enhydra logging [../logging/enhydra_logging.html]

Monolog Logger Configuration

In order to initialize Monolog the following paramer should be defined in the configuration file:

```
LogClassName=com.lutris.logging.MonologLogger
```

The application's logging configuration can be performed either in the trace.properties file in the enhydra/conf folder or in a user-specific file in which case a Monolog property should be defined in such a way that it points to the configuration file. An example:

```
Monolof=trace.properties
```

Monolog supports following logging levels: INHERIT, DEBUG, INFO, WARN, ERROR, FATAL.

More details about Monolog logging can be found at: http://monolog.objectweb.org/index.html.

More information about logging and logging configuration at: Enhydra logging [../logging/enhydra_logging.html]

Chapter 2. Configuration API

Introduction

Enhydra Enterprise supports *two* different ways of application configuration. The first one is trough <appName>.conf and the second with web.xml.

Classes and Interfaces

The API consists of:

org.enhydra.util.ConfigFileInterface - an interface that defines methods that every configuration file should have.

org.enhydra.util.AbsConfigFile - an Abstract class that implements org.enhydra.util.ConfigFileInterface and contains non specific code for all configuration files.

Implementations of *ConfigFileInterface*:

- com.lutris.util.ConfigFile - This class implements reading parameters from <appName.conf> config file and defines methods that are specific for this type of configuration file.
- org.enhydra.util.XMLConfigFile - This class implements reading parameters from web.xml file and defines methods that are specific for this type of configuration file.

The configuration object *com.lutris.util.Config* (which contains application configuration datails) now works with org.enhydra.util.ConfigFileInterface interface. For adding new type of configuration file, the class *org.enhydra.util.AbsConfigFile* should be extended (or directly interface *org.enhydra.util.ConfigFileInterface* implemented), and in web.xml file parameters ConfFile (name of the configuration file with the path relative to directory where is web.xml file) and ConfFileClass (name of the class with full package that will be used for configuration file) should be set to new values.

More details about configuring applications at Configuration in Enhydra Enterprise Server [../configuration/configuration.html]

Configuration example

Example of various settings in the web.xml:

```
<param-name>ConfFile</param-name>
<!--      <param-value>../airSent.conf</param-value>   -->
<!--      <param-value>web.xml</param-value>   -->
<param-value>web.xml</param-value>

<param-name>ConfFileClass</param-name>
<!--      <param-value>com.lutris.util.ConfigFile</param-value> -->
<!--      <param-value>org.enhydra.util.XMLConfigFile</param-value> -->
<param-value>org.enhydra.util.XMLConfigFile</param-value>
```

Chapter 3. Session API

Introduction

Sessions in Enhydra Enterprise are container managed. The session data is stored in the HttpSession object. It is possible to specify a Session Manager class in the configuration file. For example:

```
SessionManager.Class=com.lutris.appserver.server.sessionEnhydra.SimpleServletSessionManager
```

If the SessionManager.Class is not specified, the old *com.lutris.appserver.server.sessionEnhydra.StandardSessionManager* is used (it is kept for compatibility reasons, but it's use is not encouraged).

Classes and Interfaces

The configurable interface of the Session API:

package com.lutris.appserver.server.session:

- SessionManager - allocates sessions and maintains the mapping between session keys and Session objects.

Implementations and extensions (NOTE: for the first two containers, session data must be serializable in order to fully utilize the persistence obtained by the session container):

- JmxContainerAdapterSessionManager - extends ContainerAdapterSessionManager. An implementation of ContainerAdapterSessionManager specific to the Tomcat Servlet container.
- ContainerAdapterSessionManager - Simple session manager to be used with servlet container capable of managing their sessions. It uses HttpSession to keep the session data.
- SimpleServletSessionManager - extends StandardSessionManager. It obtains SessionId from the servletContainer and uses it to create a new Enhydra session
- StandardSessionManager - implements SessionManager, StandardSessionIdleHandler. This session manager maintains the mapping between session keys and sessions. It generates secure session keys that are safe to use as cookie values in web applications. It also manages the life of a session.

Session configuration examples

Example of various settings in the web.xml:

```
<env-entry>
    <env-entry-name>SessionManager/Class</env-entry-name>
 /
 env-en-
<env-entry-value>com.lutris.appserver.server.sessionEnhydra.StandardSessionManager<try-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Chapter 4. Database manager API

Classes and Interfaces

package com.lutris.appserver.server.sql

- DatabaseManager - he database management object interface. This class implementing this interface manages operations with database.

Implementations of DatabaseManager:

- StandardDatabaseManager - implements DatabaseManager interface. DODS project is Standard implementation of DatabaseManager interface.

Configuration

To set DatabaseManager implementation class, use the following syntax in configuration file: DatabaseManager.Class parameter defines class name of DatabaseManager implementation (full package name). Default value is *com.lutris.appserver.server.StandardDatabaseManager*.