JMX Guide

About Terracotta Documentation

This documentation is about Terracotta DSO, an advanced distributed-computing technology aimed at meeting special clustering requirements.

Terracotta products without the overhead and complexity of DSO meet the needs of almost all use cases and clustering requirements. To learn how to migrate from Terracotta DSO to standard Terracotta products, see Migrating From Terracotta DSO. To find documentation on non-DSO (standard) Terracotta products, see Terracotta Documentation. Terracotta release information, such as release notes and platform compatibility, is found in Product Information.

Release: 3.6
Publish Date: November, 2011 **Documentation Archive »**

JMX Guide

The Terracotta server publishes a number of MBeans to provide information on the operation of the cluster. The Terracotta Developer Console and the Terracotta Operations Center use the JMX interface to provide real-time information, but the same interface can be used by any JMX client. As of Terracotta 3.0, there is also a new API to report platform cluster events such as client nodes joining or leaving the cluster.

JMX Authentication

It is possible to configure the Terracotta Server to require authentication credentials before JMX communications take place. This will secure server shutdown, the Terracotta Developer Console, the Terracotta Operations Center, JConsole, and any other processes attempting to connect via JMX. Server authentication will not provide secure message transmission once valid credentials are provided by a listening client. The JMX messages will not be encrypted. For this reason, we recommend that the Terracotta server be placed in a secure location on a private network and only be queried remotely via an encrypted tunnel such as those provided by stunnel or SSH.

Configuring JMX authentication with Terracotta requires that an authentication element be placed in the Terracotta configuration XML file, read by the Terracotta server instance that requires security. Terracotta relies on the standard Java security mechanisms for JMX which involve the creation of a . access and .password file with correct permissions set. The default location for these files is in (for JDK 1.5+) $JAVA_HOME/jre/lib/management.

Once configured properly, the Terracotta Developer Console (or the Terracotta Operations Center) connection dialog will prompt you for credentials before completing a server connection. The stop-tc-server script located in the bin directory will also prompt for a username and password.

Configuration

See the Configuration Guide and Reference for information on configuring the Terracotta server to use JMX authentication.

Permissions

Setting permissions is done with the files **jmxremote.password** and **jmxremote.access.** Java security is very sensitive to the permission settings of these files so you will need to verify that they are properly set. One caveat to pay attention to is that the files must be owned by the system user who will be executing the Terracotta server. If the Terracotta server runs under the user "tcuser" then you must chown tcuser to change the ownership. The password file is most sensitive and requires:

```
# chmod 500 jmxremote.password

---------------------------------------------------------------------------
-rw-r--r-- 1 tcuser tcuser 2375 Feb 12 18:19 jmxremote.access
-r-x------ 1 tcuser tcuser 2873 Feb 12 18:19 jmxremote.password
---------------------------------------------------------------------------
```

NOTE: Setting these permissions on Windows is possible but requires a special DOS command. This permission is only available on Windows drives with NTFS formatting. See http://java.sun.com/j2se/1.5.0/docs/guide/management/security-windows.html for more information.

Users/Groups

You may append to the default files located in $JAVA_HOME/jre/lib/management using the following commands:

```
# echo "myusername mypassword" >> jmxremote.password
# echo "myusername readwrite" >> jmxremote.access
```

'myusername' defines the desired username and 'mypassword' defines the desired password. In jmxremote.access, myusername is associated with the username defined in jmxremote.password and must declare readwrite permissions for that group.

Sample excerpt from jmxremote.password

```
...
# Following are two commented-out entries. The "measureRole" role has
# password "QED". The "controlRole" role has password "R&D".
#
# monitorRole QED
# controlRole R&D
myusername mypassword
...
```

Sample excerpt from jmxremote.access

```
...
# Default access control entries:
# o The "monitorRole" role has readonly access.
# o The "controlRole" role has readwrite access.

monitorRole readonly
controlRole readwrite
myusername readwrite
...
```

Platform Cluster Events and Cluster Node Data

A platform cluster-events API that allows events to be communicated using a standard Java programming approach is available as part of the Terracotta API introduced with Terracotta 3.0.0. This Java-based events API can be more easily integrated into a Java applications environment.

Previous versions of Terracotta relied on JMX-based cluster events. This has caused issues for applications attempting to integrate with Terracotta because most Java applications do not use JMX in this way. Beginning with Terracotta 3.0.0, these JMX-based cluster events are no longer supported.

Applications that rely on the obsoleted JMX API for cluster events must be rewritten to use the new Java API. Support for the JMX cluster events will be removed without further notice. Application code using the TerracottaCluster mbean or registering notifications with it will receive an UnsupportedOperationException.

Rogue/Slow/Unhealthy Client Detection and monitoring

A rogue client is a client which is for some reason not healthy e.g its having a lot of pending transactions or its slow.Terracotta server exposes a list of MBeans to monitor the performance of these clients (like pending transactions) so that one can determine the health of a client and can kill any particular unhealthy client.

Here is some sample code for getting client MBeans

```
MBeanServerConnection mbsc = null;
JMXConnectorProxy jmxc = new JMXConnectorProxy("localhost", Integer.valueOf(appConfig.getAttribute(JMX_PORT)));
try {
  mbsc = jmxc.getMBeanServerConnection();
} catch (IOException e) {
  throw new AssertionError(e);
}
DSOMBean dsoMBean = (DSOMBean) MBeanServerInvocationHandler
          .newProxyInstance(mbsc, L2MBeanNames.DSO, DSOMBean.class, false);

ObjectName[] clientObjectNames = dsoMBean.getClients();
DSOClientMBean[] clients = new DSOClientMBean[clientObjectNames.length];
for (int i = 0; i < clients.length; i++) {
  clients[i] = (DSOClientMBean) MBeanServerInvocationHandler.newProxyInstance(mbsc, clientObjectNames[i],
DSOClientMBean.class, false);
}
```

This is a partial list of MBeans exposed to monitor these clients:

- **com.tc.stats.isTunneledBeansRegistered**—Is the bean registered
- **com.tc.stats.getL1InfoBeanName**—Get the name(class ObjectName) of this bean
- **com.tc.stats.getL1InfoBean**—Get the MBean
- **com.tc.stats.getInstrumentationLoggingBeanName**—Get the name(class ObjectName) of the instrumentation logging MBean)
- **com.tc.stats.getInstrumentationLoggingBean**—Get the Instrumentation logging MBean
- **com.tc.stats.getRuntimeLoggingBeanName**—Get the name(Class ObjectName) of te runtime logging MBean
- **com.tc.stats.getRuntimeLoggingBean**—Get the run time logging MBean
- **com.tc.stats.getRuntimeOutputOptionsBeanName**—Get the name(class ObjectName) of the runtime output options MBean
- **com.tc.stats.getRuntimeOutputOptionsBean**—Get the runtime output options MBean
- **com.tc.stats.getChannelID**—Get the channel ID
- **com.tc.stats.getRemoteAddress**—Get the remote address of the channel
- **com.tc.stats.getTransactionRate**—Get the transaction rate for the client

- **com.tc.stats.getObjectFaultRate**—Get the object fault rate for the client
- **com.tc.stats.getObjectFlushRate**—Get the object flush rate for the client
- **com.tc.stats.getPendingTransactionsCount**—Get the pending transactions count for the client
- **com.tc.stats.getStatistics**—Get the statistics for the given properties
- **com.tc.stats.killClient**—Kill this client
- *com.tc.stats.

Hibernate Statistics From Terracotta Clients

If you are running Terracotta clustered caches as hibernate second-level cache provider, it is possible to access the hibernate statistics + ehcache stats etc via jmx. EhcacheHibernateMBean is the main interface that exposes all the API's via jmx. It basically extends two interfaces – EhcacheStats and HibernateStats. And as the name implies EhcacheStats contains methods related with Ehcache and HibernateStats related with Hibernate. You can see cache hit/miss/put rates, change config element values dynamically – like maxElementInMemory, TTI, TTL, enable/disable statistics collection etc and various other things. Please look into the specific interface for more details.
