Apache Tomcat

MBeans Descriptors How To

Introduction

Tomcat uses JMX MBeans as the technology for implementing manageability of Tomcat.

The descriptions of JMX MBeans for Catalina are in the mbeans-descriptors.xml file in each package.

You will need to add MBean descriptions for your custom components in order to avoid a "ManagedBean is not found" exception.

Adding MBean descriptions

You may also add MBean descriptions for custom components in an mbeans-descriptors.xml file, located in the same package as the class files it describes.

The permitted syntax for the mbeans-descriptors.xml is defined by the DTD file.

The entries for a custom LDAP authentication Realm may look like this:

```
  <mbean         name="LDAPRealm"
            className="org.apache.catalina.mbeans.ClassNameMBean"
          description="Custom LDAPRealm"
               domain="Catalina"
                group="Realm"
                 type="com.myfirm.mypackage.LDAPRealm">


    <attribute   name="className"
          description="Fully qualified class name of the managed object"
                 type="java.lang.String"
```

```
                    writeable="false"/>


        <attribute      name="debug"
                  description="The debugging detail level for this component"
                         type="int"/>
        .
        .
        .

      </mbean>
```

Default Servlet Reference

What is the DefaultServlet

The default servlet is the servlet which serves static resources as well as serves the directory listings (if directory listings are enabled).

Where is it declared?

It is declared globally in *$CATALINA_BASE/conf/web.xml*. By default here is it's declaration:

```
    <servlet>
        <servlet-name>default</servlet-name>
        <servlet-class>
          org.apache.catalina.servlets.DefaultServlet
        </servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
        </init-param>
        <init-param>
            <param-name>listings</param-name>
            <param-value>false</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

...

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

So by default, the default servlet is loaded at webapp startup and directory listings are disabled and debugging is turned off.

If you need to change the DefaultServlet settings for an application you can override the default configuration by re-defining the DefaultServlet in /WEB-INF/web.xml. However, this will cause problems if you attempt to deploy the application on another container as the DefaultServlet class will not be recognised. You can work-around this problem by using the Tomcat specific /WEB-INF/tomcat-web.xml deployment descriptor. The format is identical to /WEB-INF/web.xml. It will override any default settings but not those in /WEB-INF/web.xml. Since it is Tomcat specific, it will only be processed when the application is deployed on Tomcat.

What can I change?

The DefaultServlet allows the following initParameters:

Property	**Description**
debug	Debugging level. It is not very useful unless you are a tomcat developer. As of this writing, useful values are 0, 1, 11. [0]
listings	If no welcome file is present, can a directory listing be shown? value may be **true** or **false** [false] Welcome files are part of the servlet api. **WARNING:** Listings of directories containing many entries are expensive. Multiple requests for large directory listings can consume significant proportions of server resources.
precompressed	If a precompressed version of a file exists (a file with .br or .gz appended to the file name located alongside the original file), Tomcat will serve the precompressed file if the user agent supports the matching content encoding (br or gzip) and this option is enabled. [false] The precompressed file with the with .br or .gz extension will be accessible if requested directly so if the original resource is protected with a security constraint, the precompressed versions must be similarly protected. It is also possible to configure the list of precompressed formats. The syntax is comma separated list of [content-encoding]=[file-extension] pairs. For example: br=.br,gzip=.gz,bzip2=.bz2. If

	multiple formats are specified, the client supports more than one and the client does not express a preference, the order of the list of formats will be treated as the server preference order and used to select the format returned.
readmeFile	If a directory listing is presented, a readme file may also be presented with the listing. This file is inserted as is so it may contain HTML.
globalXsltFile	If you wish to customize your directory listing, you can use an XSL transformation. This value is a relative file name (to either $CATALINA_BASE/conf/ or $CATALINA_HOME/conf/) which will be used for all directory listings. This can be overridden per context and/or per directory. See **contextXsltFile** and **localXsltFile** below. The format of the xml is shown below.
contextXsltFile	You may also customize your directory listing by context by configuring contextXsltFile. This must be a context relative path (e.g.: /path/to/context.xslt) to a file with a .xsl or .xslt extension. This overrides globalXsltFile. If this value is present but a file does not exist, then globalXsltFile will be used. If globalXsltFile does not exist, then the default directory listing will be shown.
localXsltFile	You may also customize your directory listing by directory by configuring localXsltFile. This must be a file in the directory where the listing

	will take place to with a .xsl or .xslt extension. This overrides globalXsltFile and contextXsltFile. If this value is present but a file does not exist, then contextXsltFile will be used. If contextXsltFile does not exist, then globalXsltFile will be used. If globalXsltFile does not exist, then the default directory listing will be shown.
input	Input buffer size (in bytes) when reading resources to be served. [2048]
output	Output buffer size (in bytes) when writing resources to be served. [2048]
readonly	Is this context "read only", so HTTP commands like PUT and DELETE are rejected? [true]
fileEncoding	File encoding to be used when reading static resources. [platform default]
useBomIfPresent	If a static file contains a byte order mark (BOM), should this be used to determine the file encoding in preference to fileEncoding. This setting must be one of true (remove the BOM and use it in preference to fileEncoding), false (remove the BOM but do not use it) or pass-through (do not use the BOM and do not remove it). [true]
sendfileSize	If the connector used supports sendfile, this represents the minimal file size in KiB for which sendfile will be used. Use a negative

	value to always disable sendfile. [48]
useAcceptRanges	If true, the Accept-Ranges header will be set when appropriate for the response. [true]
showServerInfo	Should server information be presented in the response sent to clients when directory listing is enabled. [true]
sortListings	Should the server sort the listings in a directory. [false]
sortDirectoriesFirst	Should the server list all directories before all files. [false]
allowPartialPut	Should the server treat an HTTP PUT request with a Range header as a partial PUT? Note that while RFC 7233 clarified that Range headers only valid for GET requests, RFC 9110 (which obsoletes RFC 7233) now allows partial puts. [true]
directoryRedirectStatusCode	When a directory redirect (trailing slash missing) is made, use this as the the HTTP response code. [302]

How do I customize directory listings?

You can override DefaultServlet with you own implementation and use that in your web.xml declaration. If you can understand what was just said, we will assume you can read the code to DefaultServlet servlet and make the appropriate adjustments. (If not, then that method isn't for you)

You can use either localXsltFile, contextXsltFile or globalXsltFile and DefaultServlet will create an xml document and run it through an xsl transformation based on the values provided in the XSLT file. localXsltFile is first checked, then contextXsltFile, followed by globalXsltFile. If no XSLT files are configured, default behavior is used.

Format:

```
    <listing>
     <entries>
      <entry type='file|dir' urlPath='aPath' size='###' date='gmt date'>
        fileName1
      </entry>
      <entry type='file|dir' urlPath='aPath' size='###' date='gmt date'>
        fileName2
      </entry>
      ...
     </entries>
     <readme></readme>
    </listing>
```

- size will be missing if type='dir'
- Readme is a CDATA entry

The following is a sample xsl file which mimics the default tomcat behavior:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="3.0">

  <xsl:output method="html" html-version="5.0"
```

```
    encoding="UTF-8" indent="no"
    doctype-system="about:legacy-compat"/>

<xsl:template match="listing">
  <html>
    <head>
        <title>
            Sample Directory Listing For
            <xsl:value-of select="@directory"/>
        </title>
        <style>
            h1 {color : white;background-color : #0086b2;}
            h3 {color : white;background-color : #0086b2;}
            body {font-family : sans-serif,Arial,Tahoma;
                      color : black;background-color : white;}
            b {color : white;background-color : #0086b2;}
            a {color : black;} HR{color : #0086b2;}
            table td { padding: 5px; }
        </style>
    </head>
    <body>
        <h1>Sample Directory Listing For
                    <xsl:value-of select="@directory"/>
        </h1>
        <hr style="height: 1px;" />
        <table style="width: 100%;">
          <tr>
            <th style="text-align: left;">Filename</th>
            <th style="text-align: center;">Size</th>
            <th style="text-align: right;">Last Modified</th>
          </tr>
          <xsl:apply-templates select="entries"/>
          </table>
        <xsl:apply-templates select="readme"/>
        <hr style="height: 1px;" />
        <h3>Apache Tomcat/11.0</h3>
    </body>
  </html>
</xsl:template>



<xsl:template match="entries">
    <xsl:apply-templates select="entry"/>
</xsl:template>

<xsl:template match="readme">
    <hr style="height: 1px;" />
    <pre><xsl:apply-templates/></pre>
    </xsl:template>


    <xsl:template match="entry">
        <tr>
            <td style="text-align: left;">
                <xsl:variable name="urlPath" select="@urlPath"/>
                <a href="{$urlPath}">
                    <pre><xsl:apply-templates/></pre>
                </a>
            </td>
            <td style="text-align: right;">
                <pre><xsl:value-of select="@size"/></pre>
            </td>
            <td style="text-align: right;">
                <pre><xsl:value-of select="@date"/></pre>
            </td>
        </tr>
    </xsl:template>


</xsl:stylesheet>
```

How do I secure directory listings?

Use web.xml in each individual webapp. See the security section of the Servlet specification.

Clustering/Session Replication How-To

For the impatient

Simply add

<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>

to your <Engine> or your <Host> element to enable clustering.

Using the above configuration will enable all-to-all session replication using the DeltaManager to replicate session deltas. By all-to-all, we mean that *every* session gets replicated to *all the other nodes* in the cluster. This works great for smaller clusters, but we don't recommend it for larger clusters — more than 4 nodes or so. Also, when using the DeltaManager, Tomcat will replicate sessions to *all* nodes, *even nodes that don't have the application deployed*. To get around these problem, you'll want to use the BackupManager. The BackupManager only replicates the session data to *one* backup node, and only to nodes that have the application deployed. Once you have a simple cluster running with the DeltaManager, you will probably want to migrate to the BackupManager as you increase the number of nodes in your cluster.

Here are some of the important default values:

1. Multicast address is 228.0.0.4
2. Multicast port is 45564 (the port and the address together determine cluster membership.
3. The IP broadcasted is java.net.InetAddress.getLocalHost().getHostAddress() (make sure you don't broadcast 127.0.0.1, this is a common error)
4. The TCP port listening for replication messages is the first available server socket in range 4000-4100
5. Listener is configured ClusterSessionListener
6. Two interceptors are configured TcpFailureDetector and MessageDispatchInterceptor

The following is the default cluster configuration:

```
        <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
                 channelSendOptions="8">

          <Manager
className="org.apache.catalina.ha.session.DeltaManager"
                   expireSessionsOnShutdown="false"
                   notifyListenersOnReplication="true"/>

          <Channel
className="org.apache.catalina.tribes.group.GroupChannel">
            <Membership
className="org.apache.catalina.tribes.membership.McastService"
                        address="228.0.0.4"
                        port="45564"
                        frequency="500"
                        dropTime="3000"/>
            <Receiver
className="org.apache.catalina.tribes.transport.nio.NioReceiver"
                      address="auto"
                      port="4000"
                      autoBind="100"
                      selectorTimeout="5000"
                      maxThreads="6"/>
            <Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
              <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
            </Sender>
            <Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/
>
            <Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatchInt
erceptor"/>
          </Channel>

          <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
                 filter=""/>
          <Valve
className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

          <Deployer
className="org.apache.catalina.ha.deploy.FarmWarDeployer"
                    tempDir="/tmp/war-temp/"
                    deployDir="/tmp/war-deploy/"
                    watchDir="/tmp/war-listen/"
                    watchEnabled="false"/>

          <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>
```

```
        </Cluster>
```

We will cover this section in more detail later in this document.

Security

The cluster implementation is written on the basis that a secure, trusted network is used for all of the cluster related network traffic. It is not safe to run a cluster on a insecure, untrusted network.

There are many options for providing a secure, trusted network for use by a Tomcat cluster. These include:

- private LAN
- a Virtual Private Network (VPN)
- IPSEC

The EncryptInterceptor provides confidentiality and integrity protection but it does not protect against all risks associated with running a Tomcat cluster on an untrusted network, particularly DoS attacks.

Cluster Basics

To run session replication in your Tomcat 11 container, the following steps should be completed:

- All your session attributes must implement java.io.Serializable
- Uncomment the Cluster element in server.xml
- If you have defined custom cluster valves, make sure you have the ReplicationValve defined as well under the Cluster element in server.xml
- If your Tomcat instances are running on the same machine, make sure the Receiver.port attribute is unique for each instance, in most cases Tomcat is smart enough to resolve this on it's own by autodetecting available ports in the range 4000-4100
- Make sure your web.xml has the <distributable/> element

- If you are using mod_jk, make sure that jvmRoute attribute is set at your Engine <Engine name="Catalina" jvmRoute="node01" > and that the jvmRoute attribute value matches your worker name in workers.properties
- Make sure that all nodes have the same time and sync with NTP service!
- Make sure that your loadbalancer is configured for sticky session mode.

Load balancing can be achieved through many techniques, as seen in the Load Balancing chapter.

Note: Remember that your session state is tracked by a cookie, so your URL must look the same from the out side otherwise, a new session will be created.

The Cluster module uses the Tomcat JULI logging framework, so you can configure logging through the regular logging.properties file. To track messages, you can enable logging on the key: org.apache.catalina.tribes.MESSAGES

Overview

To enable session replication in Tomcat, three different paths can be followed to achieve the exact same thing:

1. Using session persistence, and saving the session to a shared file system (PersistenceManager + FileStore)
2. Using session persistence, and saving the session to a shared database (PersistenceManager + JDBCStore)
3. Using in-memory-replication, using the SimpleTcpCluster that ships with Tomcat (lib/catalina-tribes.jar + lib/catalina-ha.jar)

Tomcat can perform an all-to-all replication of session state using the DeltaManager or perform backup replication to only one node using the BackupManager. The all-to-all replication is an algorithm that is only efficient when the clusters are small. For larger clusters, you should use the BackupManager to use a primary-secondary session replication strategy where the session will only be stored at one backup node.
Currently you can use the domain worker attribute (mod_jk > 1.2.8) to build

cluster partitions with the potential of having a more scalable cluster solution with the DeltaManager (you'll need to configure the domain interceptor for this). In order to keep the network traffic down in an all-to-all environment, you can split your cluster into smaller groups. This can be easily achieved by using different multicast addresses for the different groups. A very simple setup would look like this:

```
           DNS Round Robin

                     |

             Load Balancer

              /                \

        Cluster1        Cluster2

        /        \           /        \

  Tomcat1 Tomcat2   Tomcat3 Tomcat4
```

What is important to mention here, is that session replication is only the beginning of clustering. Another popular concept used to implement clusters is farming, i.e., you deploy your apps only to one server, and the cluster will distribute the deployments across the entire cluster. This is all capabilities that can go into with the FarmWarDeployer (s. cluster example at server.xml)

In the next section will go deeper into how session replication works and how to configure it.

Cluster Information

Membership is established using multicast heartbeats. Hence, if you wish to subdivide your clusters, you can do this by changing the multicast IP address or port in the <Membership> element.

The heartbeat contains the IP address of the Tomcat node and the TCP port that Tomcat listens to for replication traffic. All data communication happens over TCP.

The ReplicationValve is used to find out when the request has been completed and initiate the replication, if any. Data is only replicated if the session has changed (by calling setAttribute or removeAttribute on the session).

One of the most important performance considerations is the synchronous versus asynchronous replication. In a synchronous replication mode the request doesn't return until the replicated session has been sent over the wire and reinstantiated on all the other cluster nodes. Synchronous vs. asynchronous is configured using the channelSendOptions flag and is an integer value. The default value for the SimpleTcpCluster/DeltaManager combo is 8, which is asynchronous. See the configuration reference for more discussion on the various channelSendOptions values.

For convenience, channelSendOptions can be set by name(s) rather than integer, which are then translated to their integer value upon startup. The valid option names are: "asynchronous" (alias "async"), "byte_message" (alias "byte"), "multicast", "secure", "synchronized_ack" (alias "sync"), "udp", "use_ack". Use comma to separate multiple names, e.g. pass "async, multicast" for the options SEND_OPTIONS_ASYNCHRONOUS | SEND_OPTIONS_MULTICAST.

You can read more on the send flag(overview) or the send flag(javadoc). During async replication, the request is returned before the data has been replicated. async replication yields shorter request times, and synchronous replication guarantees the session to be replicated before the request returns.

Bind session after crash to failover node

If you are using mod_jk and not using sticky sessions or for some reasons sticky session don't work, or you are simply failing over, the session id will need to be modified as it previously contained the worker id of the previous tomcat (as defined by jvmRoute in the Engine element). To solve this, we will use the JvmRouteBinderValve.

The JvmRouteBinderValve rewrites the session id to ensure that the next request will remain sticky (and not fall back to go to random nodes since the worker is no longer available) after a fail over. The valve rewrites the JSESSIONID value in the cookie with the same name. Not having this valve in place, will make it harder to ensure stickiness in case of a failure for the mod_jk module.

Remember, if you are adding your own valves in server.xml then the defaults are no longer valid, make sure that you add in all the appropriate valves as defined by the default.

Hint:
With attribute *sessionIdAttribute* you can change the request attribute name that included the old session id. Default attribute name is *org.apache.catalina.ha.session.JvmRouteOriginalSessionID*.

Trick:
You can enable this mod_jk turnover mode via JMX before you drop a node to all backup nodes! Set enable true on all JvmRouteBinderValve backups, disable worker at mod_jk and then drop node and restart it! Then enable mod_jk Worker and disable JvmRouteBinderValves again. This use case means that only requested session are migrated.

Configuration Example

```
        <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
                 channelSendOptions="6">


          <Manager
className="org.apache.catalina.ha.session.BackupManager"
                   expireSessionsOnShutdown="false"
                   notifyListenersOnReplication="true"
                   mapSendOptions="6"/>
          <!--
          <Manager
className="org.apache.catalina.ha.session.DeltaManager"
                   expireSessionsOnShutdown="false"
                   notifyListenersOnReplication="true"/>
          -->
          <Channel
className="org.apache.catalina.tribes.group.GroupChannel">
            <Membership
className="org.apache.catalina.tribes.membership.McastService"
                        address="228.0.0.4"
                        port="45564"
                        frequency="500"
                        dropTime="3000"/>
            <Receiver
className="org.apache.catalina.tribes.transport.nio.NioReceiver"
                      address="auto"
                      port="5000"
                      selectorTimeout="100"
                      maxThreads="6"/>

            <Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
              <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
            </Sender>
            <Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/
>
            <Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatchInt
erceptor"/>
```

```
                <Interceptor
className="org.apache.catalina.tribes.group.interceptors.ThroughputIntercept
or"/>

            </Channel>


            <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"


filter=".*\.gif|.*\.js|.*\.jpeg|.*\.jpg|.*\.png|.*\.htm|.*\.html|.*\.css|.*\.txt"/>


            <Deployer
className="org.apache.catalina.ha.deploy.FarmWarDeployer"

                        tempDir="/tmp/war-temp/"

                        deployDir="/tmp/war-deploy/"

                        watchDir="/tmp/war-listen/"

                        watchEnabled="false"/>



            <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>

          </Cluster>
```

Break it down!!

```
          <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"

                      channelSendOptions="6">
```

The main element, inside this element all cluster details can be configured. The channelSendOptions is the flag that is attached to each message sent by the SimpleTcpCluster class or any objects that are invoking the SimpleTcpCluster.send method. The description of the send flags is available at our javadoc site The DeltaManager sends information using the

SimpleTcpCluster.send method, while the backup manager sends it itself directly through the channel.
For more info, Please visit the reference documentation

```
          <Manager
className="org.apache.catalina.ha.session.BackupManager"

                   expireSessionsOnShutdown="false"

                   notifyListenersOnReplication="true"

                   mapSendOptions="6"/>

          <!--

          <Manager
className="org.apache.catalina.ha.session.DeltaManager"

                   expireSessionsOnShutdown="false"

                   notifyListenersOnReplication="true"/>

          -->
```

This is a template for the manager configuration that will be used if no manager is defined in the <Context> element. In Tomcat 5.x each webapp marked distributable had to use the same manager, this is no longer the case since Tomcat you can define a manager class for each webapp, so that you can mix managers in your cluster. Obviously the managers on one node's application has to correspond with the same manager on the same application on the other node. If no manager has been specified for the webapp, and the webapp is marked <distributable/> Tomcat will take this manager configuration and create a manager instance cloning this configuration.
For more info, Please visit the reference documentation

```
          <Channel
className="org.apache.catalina.tribes.group.GroupChannel">
```

The channel element is Tribes, the group communication framework used inside Tomcat. This element encapsulates everything that has to do with communication and membership logic.

For more info, Please visit the reference documentation

```
            <Membership
className="org.apache.catalina.tribes.membership.McastService"

                        address="228.0.0.4"

                        port="45564"

                        frequency="500"

                        dropTime="3000"/>
```

Membership is done using multicasting. Please note that Tribes also supports static memberships using the StaticMembershipInterceptor if you want to extend your membership to points beyond multicasting. The address attribute is the multicast address used and the port is the multicast port. These two together create the cluster separation. If you want a QA cluster and a production cluster, the easiest config is to have the QA cluster be on a separate multicast address/port combination than the production cluster.
The membership component broadcasts TCP address/port of itself to the other nodes so that communication between nodes can be done over TCP. Please note that the address being broadcasted is the one of the Receiver.address attribute.
For more info, Please visit the reference documentation

```
            <Receiver
className="org.apache.catalina.tribes.transport.nio.NioReceiver"

                      address="auto"

                      port="5000"

                      selectorTimeout="100"

                      maxThreads="6"/>
```

In tribes the logic of sending and receiving data has been broken into two functional components. The Receiver, as the name suggests is responsible for receiving messages. Since the Tribes stack is thread less, (a popular improvement now adopted by other frameworks as well), there is a thread pool in this component that has a maxThreads and minThreads setting.

The address attribute is the host address that will be broadcasted by the membership component to the other nodes.
For more info, Please visit the reference documentation

```
          <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
            <Transport className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
          </Sender>
```

The sender component, as the name indicates is responsible for sending messages to other nodes. The sender has a shell component, the ReplicationTransmitter but the real stuff done is done in the sub component, Transport. Tribes support having a pool of senders, so that messages can be sent in parallel and if using the NIO sender, you can send messages concurrently as well.
Concurrently means one message to multiple senders at the same time and Parallel means multiple messages to multiple senders at the same time.
For more info, Please visit the reference documentation

```
          <Interceptor className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>

          <Interceptor className="org.apache.catalina.tribes.group.interceptors.MessageDispatchInterceptor"/>

          <Interceptor className="org.apache.catalina.tribes.group.interceptors.ThroughputInterceptor"/>

        </Channel>
```

Tribes uses a stack to send messages through. Each element in the stack is called an interceptor, and works much like the valves do in the Tomcat servlet container. Using interceptors, logic can be broken into more manageable pieces

of code. The interceptors configured above are:
TcpFailureDetector - verifies crashed members through TCP, if multicast packets get dropped, this interceptor protects against false positives, ie the node marked as crashed even though it still is alive and running.
MessageDispatchInterceptor - dispatches messages to a thread (thread pool) to send message asynchronously.
ThroughputInterceptor - prints out simple stats on message traffic.
Please note that the order of interceptors is important. The way they are defined in server.xml is the way they are represented in the channel stack. Think of it as a linked list, with the head being the first most interceptor and the tail the last.
For more info, Please visit the reference documentation

```
            <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"


filter=".*\.gif|.*\.js|.*\.jpeg|.*\.jpg|.*\.png|.*\.htm|.*\.html|.*\.css|.*\.txt"/>
```

The cluster uses valves to track requests to web applications, we've mentioned the ReplicationValve and the JvmRouteBinderValve above. The <Cluster> element itself is not part of the pipeline in Tomcat, instead the cluster adds the valve to its parent container. If the <Cluster> elements is configured in the <Engine> element, the valves get added to the engine and so on.
For more info, Please visit the reference documentation

```
            <Deployer
className="org.apache.catalina.ha.deploy.FarmWarDeployer"

                        tempDir="/tmp/war-temp/"

                        deployDir="/tmp/war-deploy/"

                        watchDir="/tmp/war-listen/"

                        watchEnabled="false"/>
```

The default tomcat cluster supports farmed deployment, ie, the cluster can deploy and undeploy applications on the other nodes. The state of this component is currently in flux but will be addressed soon. There was a change in the deployment algorithm between Tomcat 5.0 and 5.5 and at that point, the

logic of this component changed to where the deploy dir has to match the webapps directory.
For more info, Please visit the reference documentation

```
          <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>

        </Cluster>
```

Since the SimpleTcpCluster itself is a sender and receiver of the Channel object, components can register themselves as listeners to the SimpleTcpCluster. The listener above ClusterSessionListener listens for DeltaManager replication messages and applies the deltas to the manager that in turn applies it to the session.
For more info, Please visit the reference documentation

Cluster Architecture

Component Levels:

```
         Server

           |

         Service

           |

         Engine

           |   \

           |    --- Cluster --*

           |

         Host

           |

         ------

        /      \
     Cluster      Context(1-N)
        |               \
        |               -- Manager
        |                     \
        |                     -- DeltaManager
        |                     -- BackupManager
        |
     ---------------------------
        |                          \
       Channel                       \
    ----------------------------- \
        |                              \
     Interceptor_1 ..                 \
        |                                \
     Interceptor_N                        \
    -----------------------------        \
     |           |          |               \
   Receiver    Sender    Membership          \
                                             -- Valve
                                             |      \
                                             |       -- ReplicationValve
                                             |       -- JvmRouteBinderValve
                                             |
                                             -- LifecycleListener
```

```
                                                  |

                                                  -- ClusterListener

                                                  |           \

                                                  |             --
ClusterSessionListener

                                                  |

                                                  -- Deployer

                                                              \

                                                                -- FarmWarDeployer
```

How it Works

To make it easy to understand how clustering works, we are gonna to take you through a series of scenarios. In this scenario we only plan to use two tomcat instances TomcatA and TomcatB. We will cover the following sequence of events:

1. TomcatA starts up
2. TomcatB starts up (Wait the TomcatA start is complete)
3. TomcatA receives a request, a session S1 is created.
4. TomcatA crashes
5. TomcatB receives a request for session S1
6. TomcatA starts up
7. TomcatA receives a request, invalidate is called on the session (S1)
8. TomcatB receives a request, for a new session (S2)
9. TomcatA The session S2 expires due to inactivity.

Ok, now that we have a good sequence, we will take you through exactly what happens in the session replication code

1. **TomcatA starts up**

Tomcat starts up using the standard start up sequence. When the Host object is created, a cluster object is associated with it. When the contexts are parsed, if the distributable element is in place in the web.xml file, Tomcat asks the Cluster class (in this case SimpleTcpCluster) to create a manager for the replicated context. So with clustering enabled, distributable set in web.xml Tomcat will create a DeltaManager for that context instead of a StandardManager. The cluster class will start up a membership service (multicast) and a replication service (tcp unicast). More on the architecture further down in this document.

2. **TomcatB starts up**

When TomcatB starts up, it follows the same sequence as TomcatA did with one exception. The cluster is started and will establish a membership (TomcatA, TomcatB). TomcatB will now request the session state from a server that already exists in the cluster, in this case TomcatA. TomcatA responds to the request, and before TomcatB starts listening for HTTP requests, the state has been transferred from TomcatA to TomcatB. In case TomcatA doesn't respond, TomcatB will time out after 60 seconds, issue a log entry, and continue starting. The session state gets transferred for each web application that has distributable in its web.xml. (Note: To use session replication efficiently, all your tomcat instances should be configured the same.)

3. **TomcatA receives a request, a session S1 is created.**

The request coming in to TomcatA is handled exactly the same way as without session replication, until the request is completed, at which time the ReplicationValve will intercept the request before the response is returned to the user. At this point it finds that the session has been modified, and it uses TCP to replicate the session to TomcatB. Once the serialized data has been handed off to the operating system's TCP logic, the request returns to the user, back through the valve pipeline. For each request the entire session is replicated, this allows code that modifies attributes in the session without calling setAttribute or removeAttribute to be replicated. A useDirtyFlag configuration parameter can be used to optimize the number of times a session is replicated.

4. **TomcatA crashes**

When TomcatA crashes, TomcatB receives a notification that TomcatA has dropped out of the cluster. TomcatB removes TomcatA from its membership list, and TomcatA will no longer be notified of any changes that occurs in TomcatB. The load balancer will redirect the requests from TomcatA to TomcatB and all the sessions are current.

5. **TomcatB receives a request for session S1**

Nothing exciting, TomcatB will process the request as any other request.

6. **TomcatA starts up**

Upon start up, before TomcatA starts taking new request and making itself available to it will follow the start up sequence described above 1) 2). It will join the cluster, contact TomcatB for the current state of all the sessions. And once it receives the session state, it finishes loading and opens its HTTP/mod_jk ports. So no requests will make it to TomcatA until it has received the session state from TomcatB.

7. **TomcatA receives a request, invalidate is called on the session (S1)**

The invalidate call is intercepted, and the session is queued with invalidated sessions. When the request is complete, instead of sending out the session that has changed, it sends out an "expire" message to TomcatB and TomcatB will invalidate the session as well.

8. **TomcatB receives a request, for a new session (S2)**

Same scenario as in step 3)

9. TomcatA The session S2 expires due to inactivity.

The invalidate call is intercepted the same way as when a session is invalidated by the user, and the session is queued with invalidated sessions. At this point, the invalidated session will not be replicated across until another request comes through the system and checks the invalid queue.

Phuuuhh! :)

Membership Clustering membership is established using very simple multicast pings. Each Tomcat instance will periodically send out a multicast ping, in the ping message the instance will broadcast its IP and TCP listen port for replication. If an instance has not received such a ping within a given timeframe, the member is considered dead. Very simple, and very effective! Of course, you need to enable multicasting on your system.

TCP Replication Once a multicast ping has been received, the member is added to the cluster Upon the next replication request, the sending instance will use the host and port info and establish a TCP socket. Using this socket it sends over the serialized data. The reason I chose TCP sockets is because it has built in flow control and guaranteed delivery. So I know, when I send some data, it will make it there :)

Distributed locking and pages using frames Tomcat does not keep session instances in sync across the cluster. The implementation of such logic would be to much overhead and cause all kinds of problems. If your client accesses the same session simultaneously using multiple requests, then the last request will override the other sessions in the cluster.

Monitoring your Cluster with JMX

Monitoring is a very important question when you use a cluster. Some of the cluster objects are JMX MBeans

Add the following parameter to your startup script:

```
set CATALINA_OPTS=\

-Dcom.sun.management.jmxremote \

-Dcom.sun.management.jmxremote.port=%my.jmx.port% \

-Dcom.sun.management.jmxremote.ssl=false \

-Dcom.sun.management.jmxremote.authenticate=false
```

List of Cluster Mbeans

Name	Description	MBean ObjectName - Engine	MBean ObjectName - Host
Cluster	The complete cluster element	type=Cluster	type=Cluster,host=${HOST}
DeltaManager	This manager control the sessions and handle session replication	type=Manager,context=${APP.CONTEXT.PATH}, host=${HOST}	type=Manager,context=${APP.CONTEXT.PATH}, host=${HOST}
FarmWarDeployer	Manages the process of deploying	Not supported	type=Cluster, host=${HOST}, component=deployer

	an application to all nodes in the cluster		
Member	Represents a node in the cluster	type=Cluster, component=member, name=${NODE_NAME}	type=Cluster, host=${HOST}, component=member, name=${NODE_NAME}
ReplicationValve	This valve control the replication to the backup nodes	type=Valve,name=ReplicationValve	type=Valve,name=ReplicationValve,host=${HOST}
JvmRouteBinderValve	This is a cluster fallba	type=Valve,name=JvmRouteBinderValve, context=${APP.CONTEXT.PATH}	type=Valve,name=JvmRouteBinderValve,host=${HOST}, context=${APP.CONTEXT.PATH}

	ck valve to chang e the Sessi on ID to the curre nt tomca t jvmro ute.	

Monitoring and Managing Tomcat

Introduction

Monitoring is a key aspect of system administration. Looking inside a running server, obtaining some statistics or reconfiguring some aspects of an application are all daily administration tasks.

Enabling JMX Remote

Note: This configuration is needed only if you are going to monitor Tomcat remotely. It is not needed if you are going to monitor it locally, using the same user that Tomcat runs with.

The Oracle website includes the list of options and how to configure JMX Remote on Java 11: http://docs.oracle.com/javase/6/docs/technotes/guides/management/agent.html.

The following is a quick configuration guide for Java 11:

Add the following parameters to setenv.bat script of your Tomcat (see RUNNING.txt for details).

Note: This syntax is for Microsoft Windows. The command has to be on the same line. It is wrapped to be more readable. If Tomcat is running as a Windows service, use its configuration dialog to set java options for the service. For Linux, MacOS, etc, remove "set " from beginning of the line.

set CATALINA_OPTS=-Dcom.sun.management.jmxremote.port=%my.jmx.port%

-Dcom.sun.management.jmxremote.rmi.port=%my.rmi.port%

-Dcom.sun.management.jmxremote.ssl=false

-Dcom.sun.management.jmxremote.authenticate=false

If you don't set com.sun.management.jmxremote.rmi.port then the JSR 160 JMX-Adaptor will select a port at random which will may it difficult to configure a firewall to allow access.

If you require TLS:

1. change and add this:
2. -Dcom.sun.management.jmxremote.ssl=true
3. -Dcom.sun.management.jmxremote.registry.ssl=true
4. to configure the protocols and/or cipher suites use:
5. -Dcom.sun.management.jmxremote.ssl.enabled.protocols=%my.jmx.ssl.protocols%
6. -Dcom.sun.management.jmxremote.ssl.enabled.cipher.suites=%my.jmx.cipher.suites%
7. to client certificate authentication use:

-Dcom.sun.management.jmxremote.ssl.need.client.auth=%my.jmx.ssl.clientauth%

If you require authorization (it is strongly recommended that TLS is always used with authentication):

1. change and add this:
2.   -Dcom.sun.management.jmxremote.authenticate=true
3.   -
   Dcom.sun.management.jmxremote.password.file=../conf/jmxremote.pas
   sword

   -Dcom.sun.management.jmxremote.access.file=../conf/jmxremote.access

4. edit the access authorization
   file *$CATALINA_BASE/conf/jmxremote.access*:
5. monitorRole readonly

controlRole readwrite

6. edit the password file *$CATALINA_BASE/conf/jmxremote.password*:
7. monitorRole tomcat

controlRole tomcat

Tip: The password file should be read-only and only accessible by the operating system user Tomcat is running as.

8. Alternatively, you can configure a JAAS login module with:

   -Dcom.sun.management.jmxremote.login.config=%login.module.name%

If you need to specify a host name to be used in the RMI stubs sent to the client (e.g. because the public host name that must be used to connect is not the same as the local host name) then you can set:

set CATALINA_OPTS=-Djava.rmi.server.hostname

If you need to specify a specific interface for the JMX service to bind to then you can set:

set CATALINA_OPTS=-Dcom.sun.management.jmxremote.host

Manage Tomcat with JMX remote Ant Tasks

To simplify JMX usage with Ant, a set of tasks is provided that may be used with antlib.

antlib: Copy your catalina-ant.jar from $CATALINA_HOME/lib to $ANT_HOME/lib.

The following example shows the JMX Accessor usage:
Note: The name attribute value was wrapped here to be more readable. It has to be all on the same line, without spaces.

```
<project name="Catalina Ant JMX"

        xmlns:jmx="antlib:org.apache.catalina.ant.jmx"

        default="state"

        basedir=".">

  <property name="jmx.server.name" value="localhost" />

  <property name="jmx.server.port" value="9012" />

  <property name="cluster.server.address" value="192.168.1.75" />

  <property name="cluster.server.port" value="9025" />


  <target name="state" description="Show JMX Cluster state">

    <jmx:open

        host="${jmx.server.name}"

        port="${jmx.server.port}"

        username="controlRole"

        password="tomcat"/>

    <jmx:get

        name=
"Catalina:type=IDataSender,host=localhost,
senderAddress=${cluster.server.address},senderPort=${cluster.server.port}"
      attribute="connected"
      resultproperty="IDataSender.backup.connected"
      echo="false"
    />
    <jmx:get
      name="Catalina:type=ClusterSender,host=localhost"
      attribute="senderObjectNames"
      resultproperty="senderObjectNames"
      echo="false"
    />
    <!-- get current maxActiveSession from ClusterTest application
       echo it to Ant output and store at
       property <em>clustertest.maxActiveSessions.original</em>
    -->
    <jmx:get
      name="Catalina:type=Manager,context=/ClusterTest,host=localhost"
      attribute="maxActiveSessions"
      resultproperty="clustertest.maxActiveSessions.original"
      echo="true"
    />
    <!-- set maxActiveSession to 100
    -->
```

```
<jmx:set
   name="Catalina:type=Manager,context=/ClusterTest,host=localhost"
   attribute="maxActiveSessions"
   value="100"
   type="int"
/>
<!-- get all sessions and split result as delimiter <em>SPACE</em> for easy
     access all session ids directly with Ant property sessions.[0..n].
-->
<jmx:invoke
   name="Catalina:type=Manager,context=/ClusterTest,host=localhost"
   operation="listSessionIds"
   resultproperty="sessions"
   echo="false"
   delimiter=" "
/>
<!-- Access session attribute <em>Hello</em> from first session.
-->
<jmx:invoke
   name="Catalina:type=Manager,context=/ClusterTest,host=localhost"
   operation="getSessionAttribute"
   resultproperty="Hello"
   echo="false"
>
```

```
        <arg value="${sessions.0}"/>

        <arg value="Hello"/>

      </jmx:invoke>

      <!-- Query for all application manager.of the server from all hosts

          and bind all attributes from all found manager MBeans.

      -->

      <jmx:query

        name="Catalina:type=Manager,*"

        resultproperty="manager"

        echo="true"

        attributebinding="true"

      />

      <!-- echo the create properties -->

  <echo>

  senderObjectNames: ${senderObjectNames.0}

  IDataSender.backup.connected: ${IDataSender.backup.connected}

  session: ${sessions.0}

  manager.length: ${manager.length}

  manager.0.name: ${manager.0.name}

  manager.1.name: ${manager.1.name}

  hello: ${Hello}

  manager.ClusterTest.0.name: ${manager.ClusterTest.0.name}

  manager.ClusterTest.0.activeSessions: ${manager.ClusterTest.0.activeSessions}

  manager.ClusterTest.0.counterSend_EVT_SESSION_EXPIRED:
```

```
  ${manager.ClusterTest.0.counterSend_EVT_SESSION_EXPIRED}

manager.ClusterTest.0.counterSend_EVT_GET_ALL_SESSIONS:

  ${manager.ClusterTest.0.counterSend_EVT_GET_ALL_SESSIONS}

</echo>


    </target>


</project>
```

import: Import the JMX Accessor Project with *<import file="${CATALINA.HOME}/bin/catalina-tasks.xml" />* and reference the tasks with *jmxOpen*, *jmxSet*, *jmxGet*, *jmxQuery*, *jmxInvoke*, *jmxEquals* and *jmxCondition*.

JMXAccessorOpenTask - JMX open connection task

List of Attributes

Attribute	Description	Default value
url	Set JMX connection URL - *service:jmx:rmi:///jndi/rmi://localhost:8050/jmxrmi*	
host	Set the host, shortcut the very long URL syntax.	localhost
port	Set the remote connection port	8050
username	remote JMX connection user name.	
password	remote JMX connection password.	
ref	Name of the internal connection reference. With this attribute you can configure more the one connection	jmx.server

	inside the same Ant project.	
echo	Echo the command usage (for access analysis or debugging)	false
if	Only execute if a property of the given name **exists** in the current project.	
unless	Only execute if a property of the given name **not exists** in the current project.	

Example to open a new JMX connection

```
<jmx:open
    host="${jmx.server.name}"
    port="${jmx.server.port}"
/>
```

Example to open a JMX connection from URL, with authorization and store at other reference

```
<jmx:open
    url="service:jmx:rmi:///jndi/rmi://localhost:9024/jmxrmi"
    ref="jmx.server.9024"
    username="controlRole"
    password="tomcat"
/>
```

Example to open a JMX connection from URL, with authorization and store at other reference, but only when property *jmx.if* exists and *jmx.unless* not exists

```
<jmx:open
    url="service:jmx:rmi:///jndi/rmi://localhost:9024/jmxrmi"
```

```
        ref="jmx.server.9024"

        username="controlRole"

        password="tomcat"

        if="jmx.if"

        unless="jmx.unless"

    />
```

Note: All properties from *jmxOpen* task also exists at all other tasks and conditions.

JMXAccessorGetTask: get attribute value Ant task

List of Attributes

Attribute	**Description**	**Default value**
name	Full qualified JMX ObjectName - - *Catalina:type=Server*	
attribute	Existing MBean attribute (see Tomcat MBean description above)	
ref	JMX Connection reference	jmx.serv er
echo	Echo command usage (access and result)	false
resultproperty	Save result at this project property	
delimiter	Split result with delimiter (java.util.StringTokenizer) and use resultproperty as prefix to store tokens.	
separatearrayresul	When return value is an array, save result as	true

ts	property list (*$resultproperty.[0..N]* and *$resultproperty.length*)

Example to get remote MBean attribute from default JMX connection

```
<jmx:get
    name="Catalina:type=Manager,context=/servlets-examples,host=localhost"
    attribute="maxActiveSessions"
    resultproperty="servlets-examples.maxActiveSessions"
/>
```

Example to get and result array and split it at separate properties

```
<jmx:get
      name="Catalina:type=ClusterSender,host=localhost"
      attribute="senderObjectNames"
      resultproperty="senderObjectNames"
/>
```

Access the senderObjectNames properties with:

${senderObjectNames.length} give the number of returned sender list.

${senderObjectNames.[0..N]} found all sender object names

Example to get IDataSender attribute connected only when cluster is configured. *Note:* The name attribute value was wrapped here to be more readable. It has to be all on the same line, without spaces.

```
<jmx:query
    failonerror="false"
```

```
        name="Catalina:type=Cluster,host=${tomcat.application.host}"

        resultproperty="cluster"

    />

    <jmx:get

        name=

"Catalina:type=IDataSender,host=${tomcat.application.host},

senderAddress=${cluster.backup.address},senderPort=${cluster.backup.port}"

        attribute="connected"

        resultproperty="datasender.connected"

        if="cluster.0.name" />
```

JMXAccessorSetTask: set attribute value Ant task

List of Attributes

Attribute	Description	Default value
name	Full qualified JMX ObjectName - - *Catalina:type=Server*	
attribute	Existing MBean attribute (see Tomcat MBean description above)	
value	value that set to attribute	
type	type of the attribute.	java.lang.String
ref	JMX Connection reference	jmx.server
echo	Echo command usage (access and result)	false

Example to set remote MBean attribute value

```
    <jmx:set
```

```
        name="Catalina:type=Manager,context=/servlets-
examples,host=localhost"
        attribute="maxActiveSessions"
        value="500"
        type="int"
    />
```

JMXAccessorInvokeTask: invoke MBean operation Ant task

List of Attributes

Attribute	Description	Default value
name	Full qualified JMX ObjectName -- *Catalina:type=Server*	
operation	Existing MBean operation	
ref	JMX Connection reference	jmx.server
echo	Echo command usage (access and result)	false
resultproperty	Save result at this project property	
delimiter	Split result with delimiter (java.util.StringTokenizer) and use resultproperty as prefix to store tokens.	
separatearrayresults	When return value is an array, save result as property list (*$resultproperty.[0..N]* and *$resultproperty.length*)	true

stop an application

```
<jmx:invoke
    name="Catalina:type=Manager,context=/servlets-
examples,host=localhost"
    operation="stop"/>
```

Now you can find the sessionid at *${sessions.[0..N}* properties and access the count with ${sessions.length} property.

Example to get all sessionids

```
<jmx:invoke
    name="Catalina:type=Manager,context=/servlets-
examples,host=localhost"
    operation="listSessionIds"
    resultproperty="sessions"
    delimiter=" "
/>
```

Now you can find the sessionid at *${sessions.[0..N}* properties and access the count with ${sessions.length} property.

Example to get remote MBean session attribute from session ${sessionid.0}

```
<jmx:invoke
    name="Catalina:type=Manager,context=/ClusterTest,host=localhost"
    operation="getSessionAttribute"
    resultproperty="hello">
      <arg value="${sessionid.0}"/>
      <arg value="Hello" />
</jmx:invoke>
```

Example to create a new access logger valve at vhost *localhost*

```
 <jmx:invoke

           name="Catalina:type=MBeanFactory"

           operation="createAccessLoggerValve"

           resultproperty="accessLoggerObjectName"

 >

      <arg value="Catalina:type=Host,host=localhost"/>

 </jmx:invoke>
```

Now you can find new MBean with name stored at *${accessLoggerObjectName}* property.

JMXAccessorQueryTask: query MBean Ant task

List of Attributes

Attribute	Description	Default value
name	JMX ObjectName query string - - *Catalina:type=Manager,**	
ref	JMX Connection reference	jmx.server
echo	Echo command usage (access and result)	false
resultproperty	Prefix project property name to all founded MBeans (*mbeans.[0..N].objectname*)	
attributebinding	bind ALL MBean attributes in addition to *name*	false
delimiter	Split result with delimiter	

	(java.util.StringTokenizer) and use resultproperty as prefix to store tokens.	
separatearrayresults	When return value is an array, save result as property list (*$resultproperty.[0..N]* and *$resultproperty.length*)	true

Get all Manager ObjectNames from all services and Hosts

```
  <jmx:query
    name="Catalina:type=Manager,*
    resultproperty="manager" />
```

Now you can find the Session Manager at *${manager.[0..N].name}* properties and access the result object counter with ${manager.length} property.

Example to get the Manager from *servlet-examples* application an bind all MBean properties

```
  <jmx:query
    name="Catalina:type=Manager,context=/servlet-examples,host=localhost*"
    attributebinding="true"
    resultproperty="manager.servletExamples" />
```

Now you can find the manager at *${manager.servletExamples.0.name}* property and can access all properties from this manager with *${manager.servletExamples.0.[manager attribute names]}*. The result object counter from MBeans is stored ad ${manager.length} property.

Example to get all MBeans from a server and store inside an external XML property file

```
<project name="jmx.query"
            xmlns:jmx="antlib:org.apache.catalina.ant.jmx"
            default="query-all" basedir=".">

<property name="jmx.host" value="localhost"/>

<property name="jmx.port" value="8050"/>

<property name="jmx.username" value="controlRole"/>

<property name="jmx.password" value="tomcat"/>


<target name="query-all" description="Query all MBeans of a server">

  <!-- Configure connection -->

  <jmx:open

    host="${jmx.host}"

    port="${jmx.port}"

    ref="jmx.server"

    username="${jmx.username}"

    password="${jmx.password}"/>


  <!-- Query MBean list -->

  <jmx:query

    name="*:*"

    resultproperty="mbeans"

    attributebinding="false"/>


  <echoproperties

    destfile="mbeans.properties"

    prefix="mbeans."
```

```
        format="xml"/>


    <!-- Print results -->
    <echo message=
        "Number of MBeans in server ${jmx.host}:${jmx.port} is ${mbeans.length}"/>
</target>
</project>
```

Now you can find all MBeans inside the file *mbeans.properties*.

JMXAccessorCreateTask: remote create MBean Ant task

List of Attributes

Attribute	Description	Default value
name	Full qualified JMX ObjectName -- *Catalina:type=MBeanFactory*	
className	Existing MBean full qualified class name (see Tomcat MBean description above)	
classLoader	ObjectName of server or web application classloader (*Catalina:type=ServerClassLoader,name=[server,common,shared]* or *Catalina:type=WebappClassLoader,context=/myapps,host=localhost*)	
ref	JMX Connection reference	jmx.server
echo	Echo command usage (access and result)	false

Example to create remote MBean

```
<jmx:create
    ref="${jmx.reference}"
    name="Catalina:type=MBeanFactory"
    className="org.apache.commons.modeler.BaseModelMBean"
    classLoader="Catalina:type=ServerClassLoader,name=server">
    <arg value="org.apache.catalina.mbeans.MBeanFactory" />
</jmx:create>
```

Warning: Many Tomcat MBeans can't be linked to their parent once created. The Valve, Cluster and Realm MBeans are not automatically connected with their parent. Use the *MBeanFactory* create operation instead.

JMXAccessorUnregisterTask: remote unregister MBean Ant task

List of Attributes

Attribute	**Description**	**Default value**
name	Full qualified JMX ObjectName - - *Catalina:type=MBeanFactory*	
ref	JMX Connection reference	jmx.server
echo	Echo command usage (access and result)	false

Example to unregister remote MBean

```
<jmx:unregister
    name="Catalina:type=MBeanFactory"
/>
```

Warning: A lot of Tomcat MBeans can't be unregister.

The MBeans are not unlinked from their parent. Use *MBeanFactory* remove operation instead.

JMXAccessorCondition: express condition

List of Attributes

Attribute	Description	Default value
url	Set JMX connection URL - *service:jmx:rmi:///jndi/rmi://localhost:8050/jmxrmi*	
host	Set the host, shortcut the very long URL syntax.	localhost
port	Set the remote connection port	8050
username	remote JMX connection user name.	
password	remote JMX connection password.	
ref	Name of the internal connection reference. With this attribute you can configure more the one connection inside the same Ant project.	jmx.server
name	Full qualified JMX ObjectName - - *Catalina:type=Server*	
echo	Echo condition usage (access and result)	false
if	Only execute if a property of the given name **exists** in the current project.	
unless	Only execute if a property of the given name **not exists** in the current project.	
value (required)	Second arg for operation	

type	Value type to express operation (support *long* and *double*)	long
operation	express one • == equals • != not equals • > greater than (&gt;) • >= greater than or equals (&gt;=) • < lesser than (&lt;) • <= lesser than or equals (&lt;=)	==

Wait for server connection and that cluster backup node is accessible

```
<target name="wait">
    <waitfor maxwait="${maxwait}" maxwaitunit="second"
timeoutproperty="server.timeout" >
        <and>
            <socket server="${server.name}" port="${server.port}"/>
            <http url="${url}"/>
            <jmx:condition
                operation="=="
                host="localhost"
                port="9014"
                username="controlRole"
                password="tomcat"
                name=
"Catalina:type=IDataSender,host=localhost,senderAddress=192.168.111.1,sen
```

```
derPort=9025"
            attribute="connected"
            value="true"
         />
      </and>
   </waitfor>
   <fail if="server.timeout" message="Server ${url} don't answer inside
${maxwait} sec" />
   <echo message="Server ${url} alive" />
</target>
```

JMXAccessorEqualsCondition: equals MBean Ant condition

List of Attributes

Attribute	Description	Default value
url	Set JMX connection URL - *service:jmx:rmi:///jndi/rmi://localhost:8050/jmxrmi*	
host	Set the host, shortcut the very long URL syntax.	localhost
port	Set the remote connection port	8050
username	remote JMX connection user name.	
password	remote JMX connection password.	
ref	Name of the internal connection reference. With this attribute you can configure more the one connection inside the same Ant project.	jmx.server

name	Full qualified JMX ObjectName - - *Catalina:type=Server*	
echo	Echo condition usage (access and result)	false

Wait for server connection and that cluster backup node is accessible

```
<target name="wait">

   <waitfor maxwait="${maxwait}" maxwaitunit="second"
timeoutproperty="server.timeout" >

      <and>

         <socket server="${server.name}" port="${server.port}"/>

         <http url="${url}"/>

         <jmx:equals

            host="localhost"

            port="9014"

            username="controlRole"

            password="tomcat"

            name=

"Catalina:type=IDataSender,host=localhost,senderAddress=192.168.111.1,sen
derPort=9025"

            attribute="connected"

            value="true"

         />

      </and>

   </waitfor>

   <fail if="server.timeout" message="Server ${url} don't answer inside
${maxwait} sec" />
```

```
    <echo message="Server ${url} alive" />
</target>
```

Using the JMXProxyServlet

Tomcat offers an alternative to using remote (or even local) JMX connections while still giving you access to everything JMX has to offer: Tomcat's JMXProxyServlet.

The JMXProxyServlet allows a client to issue JMX queries via an HTTP interface. This technique offers the following advantages over using JMX directly from a client program:

- You don't have to launch a full JVM and make a remote JMX connection just to ask for one small piece of data from a running server
- You don't have to know how to work with JMX connections
- You don't need any of the complex configuration covered in the rest of this page
- Your client program does not have to be written in Java

A perfect example of JMX overkill can be seen in the case of popular server-monitoring software such as Nagios or Icinga: if you want to monitor 10 items via JMX, you will have to launch 10 JVMs, make 10 JMX connections, and then shut them all down every few minutes. With the JMXProxyServlet, you can make 10 HTTP connections and be done with it.

You can find out more information about the JMXProxyServlet in the documentation for the Tomcat manager.