Tomcat Performance Tuning and Troubleshooting
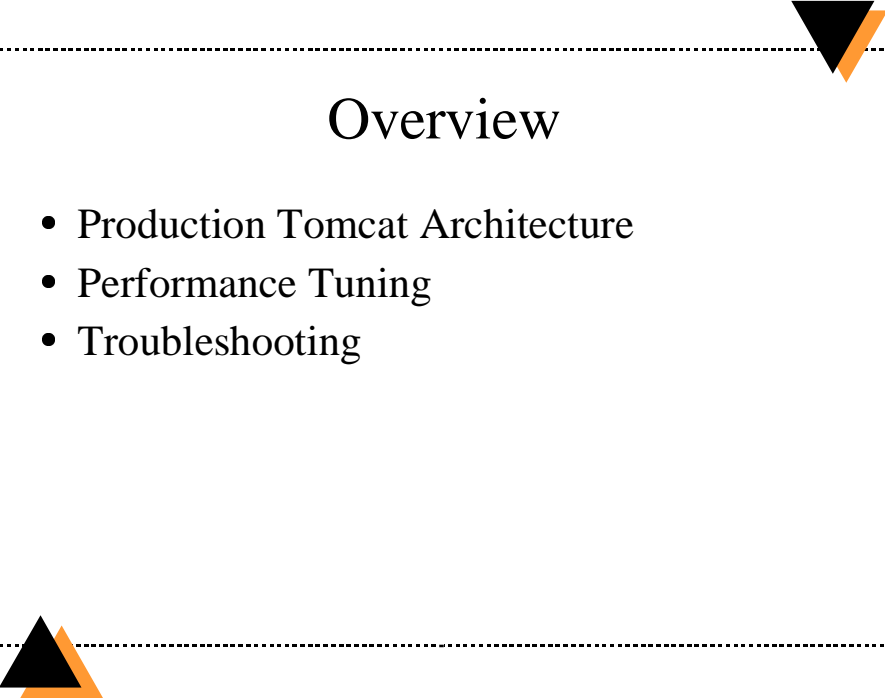
Glenn L. Nielsen

UNIX Programming Coordinator

Missouri Research and Education Network (MOREnet)

`http://kinetic.more.net/web/javaserver/performance.shtml`

`http://www.more.net/`

ApacheCon November 17, 2003

Tuning Tomcat for performance and resolving problems which affect availability are critical for a production instance of Tomcat. This session addresses how to collect and analyze data to trouble shoot problems and configure Tomcat for performance.

Overview

- Production Tomcat Architecture
- Performance Tuning
- Troubleshooting
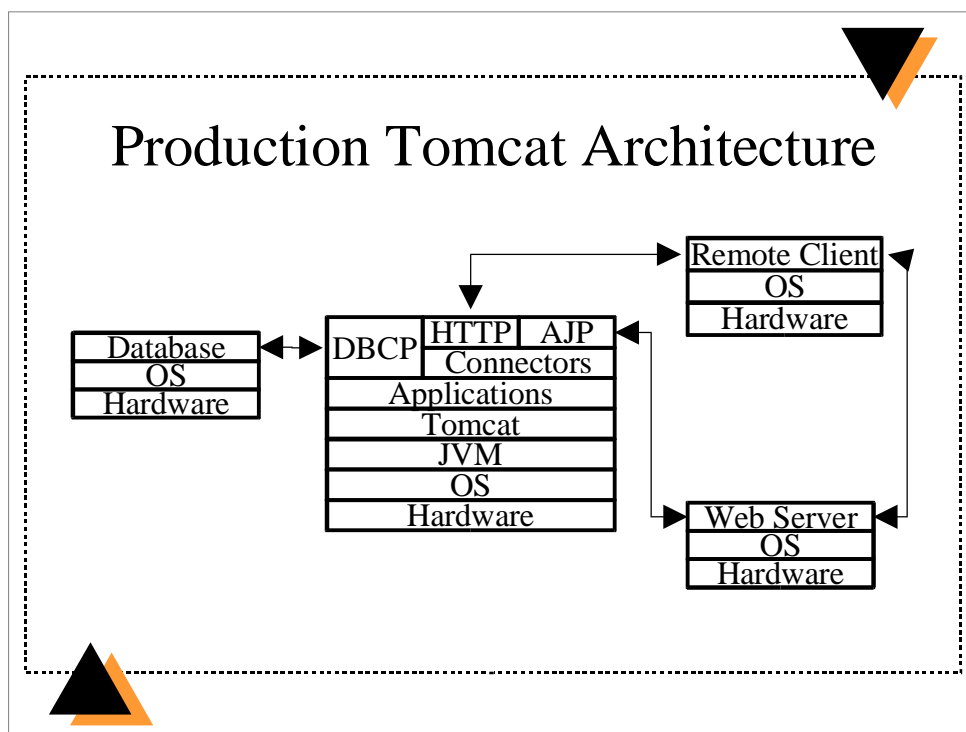
Notes

Production Tomcat Architecture

In production Tomcat relies on a number of resources which can all impact its overall performance. Understanding the overall system architecture is key to tuning performance and troubleshooting problems.

Performance Tuning

•How to measure and test performance

•JVM version, memory usage, and garbage collection

•Tomcat version and configuration

•Database connection pools

•Application Design and Profiling

Troubleshooting

•Collecting and analyzing log data

•Common problems

Production Tomcat Architecture

```
                                          ┌─────────────┐
                                     ┌───▶│Remote Client│
                                     │    ├─────────────┤
                                     │    │     OS      │◀─┐
                                     │    ├─────────────┤  │
            ┌──────────┐      ┌──────┴──┬──┤  Hardware  │  │
            │ Database │◀────▶│ DBCP │HTTP│ AJP │◀──┐    │
            ├──────────┤      ├──────┴────┴─────┤   │    │
            │    OS    │      │   Connectors    │   │    │
            ├──────────┤      ├─────────────────┤   │    │
            │ Hardware │      │  Applications   │   │    │
            └──────────┘      ├─────────────────┤   │    │
                              │     Tomcat      │   │    │
                              ├─────────────────┤   │    │
                              │      JVM        │   │    │
                              ├─────────────────┤   │    │
                              │      OS         │   │    │
                              ├─────────────────┤   │    │
                              │   Hardware      │   │    │
                              └─────────────────┘   │    │
                                        ┌───────────┘    │
                                        ▼    ┌─────────────┐
                                             │ Web Server  │◀─┘
                                             ├─────────────┤
                                             │     OS      │
                                             ├─────────────┤
                                             │  Hardware   │
                                             └─────────────┘
```

Notes

Hardware. CPU(s), memory, network IO, and file IO

Operating System. Symmetric multiprocessing (SMP) and thread support

JVM. Version, tuning memory usage, and tuning garbage collection are important.

Tomcat. Later releases are more optimized for performance. If you use JavaServer Pages, Jasper 2 available in Tomcat 4.1 significantly boosts performance.

Application. Application design can have the largest impact on overall performance.

Database. Conncurrent db connections allowed, db connection pooling, and object caching.

Web Server. Can be used in front of Tomcat to serve static content removing this load from Tomcat. This allows Tomcat to do what it does best, dynamic web application content. Allows load balancing using multiple instances of Tomcat.

Network. Network delays. Nothing we can do about this.

Remote Client. Speed of remote client network connection. Nothing we can do about this except compress the content using Apache mod_gzip or Tomcat compression filter.

Performance Tuning-
Measuring Performance

- Before you can tune your server for optimal performance you must be able to measure how well it is performing.

- Request Latency: The time it takes from when a remote client requests a page until that page has been completely rendered in their browser.

- Request latency mean and standard deviation trend graph

Notes

Without testing and measuring performance how can you know if any changes made have improved performance?

Request latency is the most important performance measurement because it reflects the responsiveness of your site for visitors.

You can only control the portion of the overall request latency which occurs while processing the request on your servers.

There is nothing you can do about the speed of the remote clients network connection or any network delays between your servers and the remote client.

http://kinetic.more.net/ar/kinetic/tomcat_deviation.png

Performance Tuning - Measuring Performance

- Test systems should match production as closely as possible.
  - Same hardware, OS, and software.
  - Populate databases with the number of records expected in production.
  - Test HTTP requests with different request parameters.
  - Simulate expected traffic over time including short term spikes.
  - Final tests should be over longer periods (days).

Notes

Databases

An application may perform well with a small number of records in the database. Performance could degrade a great deal as the number of records in the table(s) increases.

Request parameters

Test with the extremes. A page which performs a search may perform well when the search criteria returns a small result set, but perform poorly when the search criteria returns a large result set.

Simulated request traffic

A page which performs well with low request volume can cause the server to fail under higher request volume.

Longer tests

The JVM performance changes over time and can actually improve if using HotSpot. Memory leaks, db temporarily unavailable, etc. can only be found when running longer tests.

Performance Tuning - Measuring Performance

- Server load testing tools
  - Apache Jakarta JMeter
  - ab - Apache HTTP server benchmark tool
  - Many other free and commercial testing tools.
- Production
  - Apache 1.3/2.0 mod_jk (1.2)
  - Java Management Extensions (JMX)
  - Introscope - Application Server Monitoring
  - Example shell script

Notes

Apache Jakarta Jmeter

http://jakarta.apache.org/jmeter

Apache HTTP server benchmarking tool

http://httpd.apache.org/docs/programs/ab.html

mod_jk

If you are using Apache 1.3 and mod_jk 1.2 you can use JkRequestLogFormat to log individual Tomcat request latency in nanoseconds. The latest mod_jk 1.2 distribution comes with some perl scripts for analyzing mod_jk logs. The request latency graph we viewed in slide 4 was generated with these scripts.

http://jakarta.apache.org/tomcat/tomcat-4.1-doc/jk2/jk/aphowto.html

Java Management Extensions (JMX)

Tomcat currently uses JMX for configuration management. Using JMX to instrument Tomcat for production monitoring is being considered by Tomcat developers.

Open-source JMX API (MX4J) http://mx4j.sourceforge.net/

Open-source JMX Management Console (MC4J) http://mc4j.sourceforge.net/

Introscope--Application Server Monitoring

Introscope can instrument any Java application and also supports JMX.

http://www.wilytech.com/

Example Shell Script

The example shell script in Appendix D can be used to detect major Tomcat problems. It forces the JVM to do a Thread Stacktrace dump and send e-mail notifications.

Performance Tuning - JVM

- Use the most recent stable JVM release.
- Use HotSpot profiling optimizer "-server".
- Tune JVM memory usage and garbage collection (GC).
- If possible dedicate a server for use as app server.

Notes

Sun Java 1.3 and later releases include the HotSpot profiling optimizer customized for long running server applications.

Sun Java 1.4 improves performance for servlet containers by approximately 35%.

Tomcat will freeze processing of all requests while the JVM is performing GC. On a poorly tuned JVM this can last 10's of seconds. Most GC's should take < 1 second and never exceed 10 seconds.

Performance Tuning - JVM memory and GC

- You must test and tune the JVM for your hardware, OS, and application.
  - Tune the -Xms (minimum) and -Xmx (maximum) java stack memory
  - Use -Xincgc to enable incremental garbage collection
  - Try reducing -Xss thread stack memory usage
  - Use -verbose:gc to capture GC performance data
  - JavaWorld Garbage Collection Article
  - Sun HotSpot Performance Documentation

Notes

Setting the minimum and maximum java stack memory to the same value can improve GC performance.

Make sure the java process always keeps the memory it uses resident in physical memory and not swapped out to virtual memory.

JVM garbage collection performance can degrade significantly if the JVM stack gets swapped out to disk as virtual memory.

JavaWorld Garbage Collection Article

http://www.javaworld.com/javaworld/jw-01-2002/jw-0111-hotspotgc.html

Sun HotSpot Performance Documenation

http://java.sun.com/docs/hotspot/index.html

Performance Tuning - Tomcat

- Each major new Tomcat release has been better optimized for performance.
- JavaServer Page performance was signficantly improved in Tomcat 4.1 which includes version 2 of the Jasper JSP Engine.
- For medium to high volume sites use a web server in front of Tomcat to serve static content.

Notes

Jasper 2

Can perform JSP page recompile in the background. Implements JSP custom tag pooling.

Web server

Let the web server do what it does best, serve static content. And let Tomcat do what it does best, generate dynamic content.

Apache 2.0

Apache mod_jk has better performance characteristics when you use Apache 2.0 with the threaded worker MPM.

Performance Tuning - Tomcat Configuration

- Example Tomcat 4.1Configuration
- Set reloadable to false
- Set liveDeploy to false
- Set debug to 0
- Set swallowOutput to true

Notes

See Appendix A -- Example Tomcat 4.1 Configuration

reloadable false

When reloadable is true Tomcat tries to detect web application class changes and automatically reload any classes which change. Setting this to false removes a lot of unnecessary overhead in production.

liveDeploy false

liveDeploy controls whether your webapps directory is periodically checked for new web application directories and war files. This is done using a background thread. Set this to false and use the manager application or ant deploy tasks instead.

debug 0

Disable all debug output, unnecessary logging just adds overhead.

swallowOutput true

This makes sure all output to stdout or stderr for a web application gets directed to the web application log rather than the console or catalina.out. This makes it easier to troubleshoot web application problems.

Performance Tuning - Connector Configuration

- Example Tomcat 4.1 Configuration
- minProcessors
- maxProcessors
- acceptCount
- enableLookups

Notes

See Appendix A -- Example Tomcat 4.1 Configuration

minProcessors

Set to number of processors required for normal request volume.

maxProcessors

Set to twice the max number of concurrent requests expected.

acceptCount

Don't set too high, this sets the number of pending requests awaiting processing. In my opinion it is better to deny a few requests than overload Tomcat and cause problems for all requests.

enableLookups

DNS lookups can add significant delays.

Performance Tuning -
Jasper 2 Configuration

- Example Jasper 2 Configuration
- Set development to false
- Use JSP custom tag pooling
- Configure Jasper to fork javac compiles

Notes

See Appendix B -- Example Jasper 2 Configuration

development false

Disables JSP page out of date checks on each request and enables JSP background recompiles. development is set to true by default.

JSP custom tag pooling

Object pooling of classes which implement custom tags significantly improves performance of JSP pages which use custom tag libraries. JSP custom tag pooling is enabled by default.

Use fork for JSP page compiles

The JVM compiler javac has known memory leaks. Eliminates JVM memory usage and GC overhead of javac by configuring Jasper to fork javac when compiling JSP pages.

Performance Tuning - Databases

- Example Tomcat 4.1 Configuration
- Jakarta-Commons DBCP
- Tomcat 4.1 JNDI DataSource How To
- Database object caching

Notes

See Appendix A -- Example Tomcat 4.1 Configuration

Jakarta-Commons DBCP

http://jakarta.apache.org/commons/dbcp.html

Tomcat 4.1 JNDI DataSource How To

http://jakarta.apache.org/tomcat/tomcat-4.1-doc/jndi-datasource-examples-howto.html

Database object caching

- Using middleware to persist and cache objects from your database can significantly improve performance.
- Fewer delays due to db query latency.
- Less thrashing of the JVM for creation and subsequent GC of objects created for result sets.
- Castor JDO http://castor.exolba.org/
- Torque http://db.apache.org/torque/
- Object Relational Bridge http://db.apache.org/

Performance Tuning - Application Design

- Switch dynamic pages to static pages
- Cache dynamic page output
- Database connection pooling
- Database object caching
- Reduce the amount of HTML generated for large pages

Notes

Switch dynamic pages to static pages

If the data used to generate a dynamic page rarely changes, modify it to a static page which you regenerate periodically.

Cache dynamic page output

If each request for a page doesn't generate unique output but changes too frequently to make it static, consider temporary caching of the generated output.

•Servlet 2.3 Cache Filter
http://www.servletsuite.com/servlets/cacheflt.htm

•Jakarta-Taglibs Cache JSP tag library
http://jakarta.apache.org/taglibs/doc/cache-doc/intro.html

•OSCache Servlet Filter
http://www.opensymphony.com/oscache/

Performance Tuning - Application Profiling

- Use tools like JProbe or OptimizeIt to profile your web applications during the development phase.
- Look for possible thread synchronization bottlenecks.
- Date and Time java class thread synchronization bottlenecks.

Notes

JProbe

http://www.sitraka.com/software/jprobe/

OptimizeIt

http://www.borland.com/optimizeit/

Date/Time synchronization bottleneck

The following java classes have a number of thread synchronization bottlenecks:

- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.text.SimpleDateFormat
- java.util.Calendar
- java.util.Date

Troubleshooting - Log Review

- Review logs often
- Intermittent problems
- Application bugs
- Connector Broken Pipe

Notes

Review logs often, daily if possible.

Can help identify intermittent problems.

Application bugs can be found and fixed.

Connector Broken Pipe

- For HTTP Connector indicates that the remote client aborted the request.
- For a web server JK Connector indicates that the web server process or thread was terminated.
- These are normal and rarely due to a problem with Tomcat.

Troubleshooting - Common Problems

- Tomcat Pauses
- Memory Usage
- No Processors Available
- mod_jk errors
- Database connections

Notes

Troubleshooting - Tomcat Pause

- Tomcat freezes or pauses with no requests being processed.
- Usually due to a long pause from JVM garbage collection.

Notes

Use java –verbose:gc startup argument to collect GC data.

A long pause can cause a cascading effect and high load once Tomcat starts handling requests again.

Don't set the Connector acceptCount to high.

Troubleshooting - Memory Usage

- JVM system memory usage
- OutOfMemory exceptions and memory leaks

Notes

JVM system memory usage

- •JVM memory usage will increase over time to your -Xmx maximum memory config.
- •The JVM will never release that memory back to the OS.
- •The JVM manages its memory usage internally.
- •Use the java -verbose:gc option to collect data on JVM memory usage.

OutOfMemory exceptions and memory leaks

- •These are almost always caused by your application code.
- •Try increasing -Xmx if you have available memory on the system.
- •Stop and restart Tomcat
- •Profile your web application to determine where and why it is using excessive amounts of memory or leaking memory.

Troubleshooting -
No Processors Available

- Tomcat pause from long GC times
- Tomcat overloaded
- Database connection delays
- Connector maxProcessors set too low
- Thread Stacktrace Dump

Notes

Tomcat overloaded

Review request volume, Tomcat CPU usage, request latency, and GC data.

Database connection delays

If there are database connection problems each request which uses the database could pause for as long as your db connection timeout. This puts processing for requests which use the db into a long wait. More and more Processors get used because too many requests are waiting for the db.

Connector maxProcessors set to low

If there is no other problem causing the Connector to run out of Processors, your maxProcessor setting may be to low. I recommend setting it to twice the number of concurrent requests you expect Tomcat to process.

Apache mod_jk does a one-to-one mapping of httpd processes to Tomcat processors. If you have many web sites virtually hosted but not all of them use Tomcat, consider running a separate instance of Apache for just those sites that use Tomcat.

Thread Stacktrace Dump

If Tomcat is having problems, signal the JVM to do a Thread Stacktrace Dump. Reviewing the Stacktrace for each thread will often help you identify the source of the problem. Look for thread synchronization bottlenecks.

http://developer.java.sun.com/developer/technicalArticles/Programming/Stacktrace/

Troubleshooting - mod_jk errors

- Client connection aborted or network problems
- jk_tcp_socket_recvfull failed, Receiving from tomcat failed
- Error connecting to Tomcat, error sending request body, can't receive response

Notes

ERROR: Client connection aborted or network problems

The remote browser client aborted the HTTP request. It is normal to see this error once in a while. This error occurs more frequently when request latency has increased due to Tomcat being overloaded.

Error - jk_tcp_socket_recvfull failed, Receiving from tomcat failed

All of Tomcats AjpProcessor's are in use and Tomcat rejected the connection. Tomcat is overloaded or the Connector maxProcessors needs to be increased.

Error connecting to the Tomcat process, error sending request, can't receive body

Either Tomcat isn't running, Tomcat and mod_jk are configured to use different ports, or a firewall is preventing the connection.

Troubleshooting - Database Connections

- Database Connection Failures
- Random Connection Closed Exceptions

Notes

Database connection failures

•Long JVM GC times greater than db connection timeout can cause intermittent db connection failures.

•Database connection pool out of connections due to abandoned db pooled connections which are never recycled.

•Spike in traffic to site using up all available db connections. This can either be in your db connection pool configuration or in your database server configuration.

Random Connection Closed Exceptions

•Usually caused by a bug in the web application usage of the db.

•See Appendix C -- Random Connection Closed Exceptions

Tomcat Performance Tuning and Troubleshooting

Questions?

Comments?

Notes

Appendix A -- conf/server.xml Tomcat 4.1 configuration

```
<Server port="8005" shutdown="SHUTDOWN" debug="0">

  <Service name="Tomcat-Apache">

    <Connector className="org.apache.ajp.tomcat4.Ajp13Connector"
               port="8009" minProcessors="50" maxProcessors="375"
               acceptCount="10" connectionTimeout="0" debug="0"/>

    <Engine name="Tomcat-Apache" defaultHost="www.myhost.com" debug="0">

      <Logger className="org.apache.catalina.logger.FileLogger"
              prefix="catalina_log." suffix=".txt" timestamp="true"/>

      <Host name="www.myhost.com" debug="0" appBase="webapps" swallowOutput="true"
            unpackWARs="true" autoDeploy="true" liveDeploy="false">

        <Realm  className="org.apache.catalina.realm.JDBCRealm" debug="0"
                driverName="org.gjt.mm.mysql.Driver"
                connectionURL="jdbc:mysql://localhost/realm?autoReconnect=true"
                connectionName="admin" connectionPassword="password"
                userTable="users" userNameCol="user_name" userCredCol="user_pass"
                userRoleTable="user_roles" roleNameCol="role_name" digest="MD5" />

        <Logger className="org.apache.catalina.logger.FileLogger"
                prefix="myhost_log-" suffix=".txt" timestamp="true"/>

        <DefaultContext debug="0" reloadable="false" swallowOutput="true">
          <!-- JNDI JDBC DataSource Resource for using MySQL dB -->
          <Resource name="jdbc/data" auth="CONTAINER"
                    type="javax.sql.DataSource"/>
          <ResourceParams name="jdbc/data">
            <parameter>
              <name>factory</name>
              <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
            </parameter>
            <parameter><name>username</name><value>username</value></parameter>
            <parameter><name>password</name><value>password</value></parameter>
            <parameter>
              <name>driverClassName</name>
              <value>org.gjt.mm.mysql.Driver</value></parameter>
            <parameter>
              <name>url</name>
              <value>jdbc:mysql://localhost/myhost?autoReconnect=true</value>
            </parameter>
            <parameter>
              <name>maxIdle</name>
              <value>50</value>
            </parameter>
            <parameter>
              <name>maxActive</name>
              <value>200</value>
            </parameter>
            <parameter>
              <name>maxWait</name>
              <value>10000</value>
            </parameter>
            <parameter>
              <name>logAbandoned</name>
              <value>true</value>
            </parameter>
            <parameter>
              <name>removeAbandoned</name>
              <value>true</value>
            </parameter>
            <parameter>
              <name>removeAbandonedTimeout</name>
              <value>300</value>
            </parameter>
          </ResourceParams>
        </DefaultContext>
```

```
        <Context path="/manager"
                docBase="/usr/local/kinetic/tomcat4/server/webapps/manager"
                debug="0" privileged="true">
          <Valve className="org.apache.catalina.valves.RemoteAddrValve"
                allow="127.0.0.1"/>
        </Context>

      </Host>

    </Engine>

  </Service>

</Server>
```

Appendix B -- conf/web.xml Jasper 2 configuration

```
<!-- The JSP page compiler and execution servlet, which is the mechanism  -->
<!-- used by Tomcat to support JSP pages.  Traditionally, this servlet    -->
<!-- is mapped to URL patterh "*.jsp".  This servlet supports the         -->
<!-- following initialization parameters (default values are in square    -->
<!-- brackets):                                                            -->
<!--                                                                       -->
<!--    checkInterval       If development is false and relaoding is true, -->
<!--                        background compiles are enabled. checkInterval -->
<!--                        is the time in seconds between checks to see   -->
<!--                        if a JSP page needs to be recompiled. [300]    -->
<!--                                                                       -->
<!--    compiler            Which compiler Ant should use to compile JSP   -->
<!--                        pages.  See the Ant documenation for more      -->
<!--                        information. [javac]                           -->
<!--                                                                       -->
<!--    classdebuginfo      Should the class file be compiled with         -->
<!--                        debugging information?  [true]                 -->
<!--                                                                       -->
<!--    classpath           What class path should I use while compiling   -->
<!--                        generated servlets?  [Created dynamically      -->
<!--                        based on the current web application]          -->
<!--                                                                       -->
<!--    development         Is Jasper used in development mode (will check -->
<!--                        for JSP modification on every access)?  [true] -->
<!--                                                                       -->
<!--    enablePooling       Determines whether tag handler pooling is      -->
<!--                        enabled  [true]                                -->
<!--                                                                       -->
<!--    fork                Tell Ant to fork compiles of JSP pages so that -->
<!--                        a separate JVM is used for JSP page compiles   -->
<!--                        from the one Tomcat is running in. [true]      -->
<!--                                                                       -->
<!--    ieClassId           The class-id value to be sent to Internet      -->
<!--                        Explorer when using <jsp:plugin> tags.    -->
<!--                        [clsid:8AD9C840-044E-11D1-B3E9-00805F499D93]   -->
<!--                                                                       -->
<!--    javaEncoding        Java file encoding to use for generating java  -->
<!--                        source files. [UTF8]                           -->
<!--                                                                       -->
<!--    keepgenerated       Should we keep the generated Java source code  -->
<!--                        for each page instead of deleting it? [true]   -->
<!--                                                                       -->
<!--    largefile           Should we store the static content of JSP      -->
<!--                        pages in external data files, to reduce the    -->
<!--                        size of the generated servlets?  [false]       -->
<!--                                                                       -->
<!--    logVerbosityLevel   The level of detailed messages to be produced  -->
<!--                        by this servlet.  Increasing levels cause the  -->
<!--                        generation of more messages.  Valid values are -->
<!--                        FATAL, ERROR, WARNING, INFORMATION, and DEBUG. -->
<!--                        [WARNING]                                      -->
<!--                                                                       -->
<!--    mappedfile          Should we generate static content with one     -->
<!--                        print statement per input line, to ease        -->
<!--                        debugging?  [false]                            -->
<!--                                                                       -->
<!--    reloading           Should Jasper check for modified JSPs?  [true] -->
<!--                                                                       -->
<!--    scratchdir          What scratch directory should we use when      -->
<!--                        compiling JSP pages?  [default work directory  -->
<!--                        for the current web application]               -->
<!--                                                                       -->
<!-- If you wish to use Jikes to compile JSP pages:                        -->
<!--    Set the init parameter "compiler" to "jikes".  Define              -->
<!--    the property "-Dbuild.compiler.emacs=true" when starting Tomcat.   -->
<!--    If you get an error reporting that jikes can't use UTF8 encoding,  -->
<!--    try setting the init parameter "javaEncoding" to "ISO-8859-1".     -->
```

```xml
<servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
    <init-param>
        <param-name>logVerbosityLevel</param-name>
        <param-value>ERROR</param-value>
    </init-param>

    <!-- Ant is used to compile JSP pages, setting fork to true tells  -->
    <!-- to compile the JSP page in a separate process.  This prevents -->
    <!-- memory leaks due to javac and reduces GC due to JSP page       -->
    <!-- compiles.  Do not set fork to true on MS Windows               -->
    <init-param>
        <param-name>fork</param-name>
        <param-value>true</param-value>
    </init-param>

    <!-- Set the following two parameters to false for production -->
    <init-param>
        <param-name>development</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>mappedfile</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
</servlet>
```

Appendix C - Random Database Connection Closed Exceptions

These can occur when one request gets a db connection from the connection pool and closes it twice. When using a connection pool, closing the connection just returns it to the pool for reuse by another request, it doesn't close the connection. And Tomcat uses multiple threads to handle concurrent requests. Here is an example of the sequence of events which could cause this error in Tomcat:

1. Request 1 running in Thread 1 gets a db connection.
2. Request 1 closes the db connection.
3. The JVM switches the running thread to Thread 2
4. Request 2 running in Thread 2 gets a db connection(the same db connection just closed by Request 1).
5. The JVM switches the running thread back to Thread 1
6. Request 1 closes the db connection a second time in a finally block.
7. The JVM switches the running thread back to Thread 2
8. Request 2 Thread 2 tries to use the db connection but fails because Request 1 closed it.

Here is an example of properly written code to use a db connection obtained from a connection pool:

```
Connection conn = null;
Statement stmt = null;  // Or PreparedStatement if needed
ResultSet rs = null;
try {
  conn = ... get connection from connection pool ...
  stmt = conn.createStatement("select ...");
  rs = stmt.executeQuery();
  ... iterate through the result set ...
  rs.close();
  rs = null;
  stmt.close();
  stmt = null;
  conn.close(); // Return to connection pool
  conn = null;  // Make sure we don't close it twice
} catch (SQLException e) {
  ... deal with errors ...
} finally {
  // Always make sure result sets and statements are closed,
  // and the connection is returned to the pool
  if (rs != null) {
    try { rs.close(); } catch (SQLException e) { ; }
    rs = null;
  }
  if (stmt != null) {
    try { stmt.close(); } catch (SQLException e) { ; }
    stmt = null;
  }
  if (conn != null) {
    try { conn.close(); } catch (SQLException e) { ; }
    conn = null;
  }
}
```

Appendix D - Example UNIX shell script

Script is owned by the user tomcat runs as and uses setuid so that the script can be run by someone other than root. An alternative would be for the script to be run as root.

```
-r-sr-x---   1 tomcatd  staff           573 Nov 15 06:59 check_tomcat.sh
```

Here are the crontab entries for running this script every 10 minutes:

```
# Check Tomcat every 10 minutes for load problems
0,10,20,30,40,50 * * * * $HOME/check_tomcat.sh
```

Here is the script:

```sh
#!/bin/sh

# Script for detecting major tomcat problems

# Get the current date
date=`/usr/bin/date "+%Y-%m-%d"`
# Format the name of todays catalina connector log file
logfile="/usr/local/tomcat4/logs/catalina_log.$date.txt"

# If the log file doesn't exist we have nothing to check
if [ ! -r $logfile ]; then
  exit
fi

# Check the last 50 lines in the log for a line indicating that a new
# Tomcat Processor was started
num=`/usr/local/bin/tail -50 $logfile | \
  /usr/local/bin/egrep "Starting background thread" | \
  /usr/local/bin/wc -l`

if [ $num -ge 5 ]; then
  # Signal the JVM to do a Thread Stacktrace Dump
  /usr/bin/pkill -QUIT java
  # Send email to administrator that Tomcat is having problems
  /usr/bin/mailx -r admin\@your.host -s "Tomcat load problems" admin\@your.host
fi
```