APACHE TOMCAT COOKBOOK

Hot Recipes for Apache Tomcat

Java Code Geeks
Java 2 Java Developers Resource Center

Apache Tomcat Cookbook

Contents

Preface

Apache Tomcat, often referred to as Tomcat, is an open-source web server developed by the Apache Software Foundation (ASF). Tomcat implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment in which Java code can run.

Tomcat is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation, released under the Apache License 2.0 license, and is open-source software. Tomcat 7.x implements the Servlet 3.0 and JSP 2.2 specifications. It requires Java version 1.6, although previous versions have run on Java 1.1 through 1.5. Versions 5 through 6 saw improvements in garbage collection, JSP parsing, performance and scalability. Native wrappers, known as "Tomcat Native", are available for Microsoft Windows and Unix for platform integration. Tomcat 8.x implements the Servlet 3.1 and JSP 2.4 Specifications. (https://en.wikipedia.org/wiki/Apache_Tomcat)

In this ebook, we provide a compilation of Tomcat examples that will help you kick-start your own web projects. We cover a wide range of topics, from installation and configuration, to logging and clustering. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike.

JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

You can find them online at https://www.javacodegeeks.com/

Chapter 1

How to Install Tomcat on Ubuntu Linux

Apache Tomcat is a web server and servlet container that is used to serve Java applications. A servlet is a Java technology-based Web component, managed by a container, that generates dynamic content.

1.1 The tools

- Ubuntu Linux 16.04
- Java JDK
- Apache Tomcat

1.2 Introduction

In this example we are going to install on Ubuntu Linux: Java JDK 8. Tomcat Server.

We are going to create a script to make Tomcat start with the system and easily start and stop the Tomcat service. Edit the tomcat users to access the Tomcat management console.

1.3 Prerequisites

- Ubuntu Linux installed

1.4 Download the JDK

Go to the page JDK Download

Figure 1.1: JDK Download

Accept the end user Select the JDK to download:

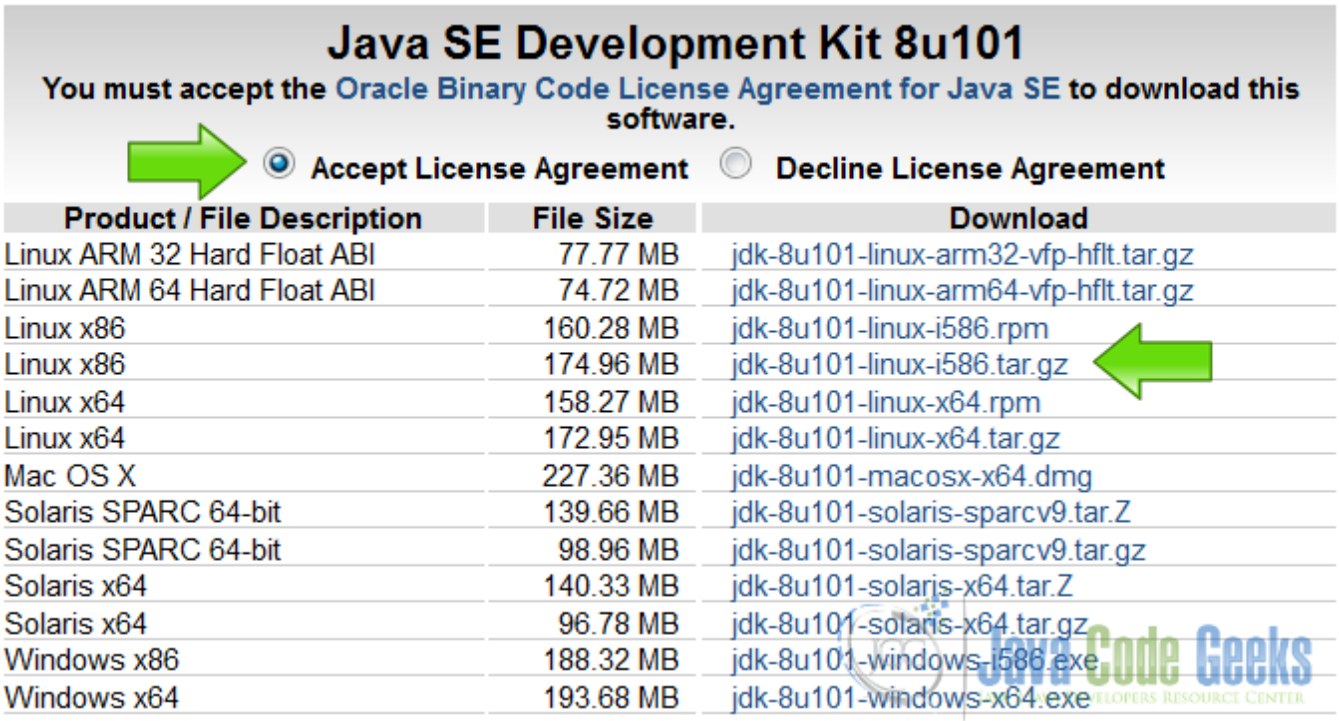

Figure 1.2: Accept agreement

1.5 Download Tomcat

Go to the page Tomcat download and download the tomcat server.

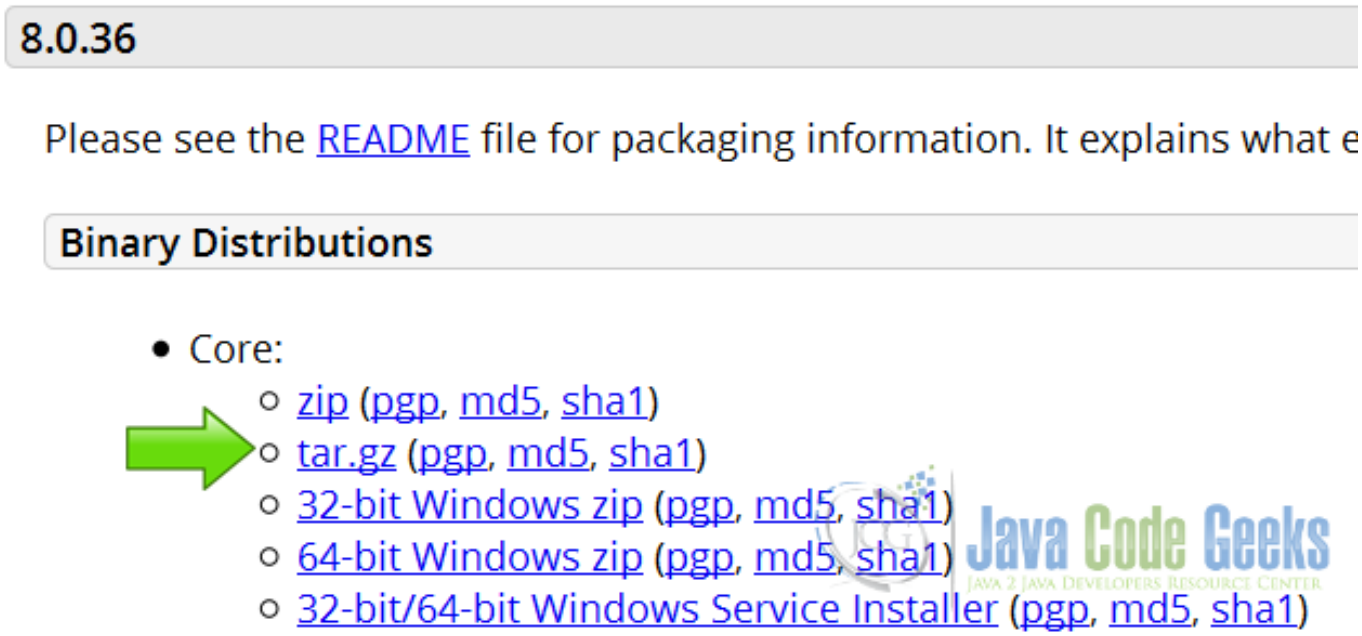

Figure 1.3: Download Tomcat

1.6 Install the JDK

Copy the JDK to the `/opt` directory.

Copy the JDK

```
sudo cp /home/Downloads/jboadas/jdk-8u77-linux-i586.tar.gz /opt/
```

Extract the JDK compressed file.

Extract the file

```
sudo tar xvf jdk-8u77-linux-i586.tar.gz
```

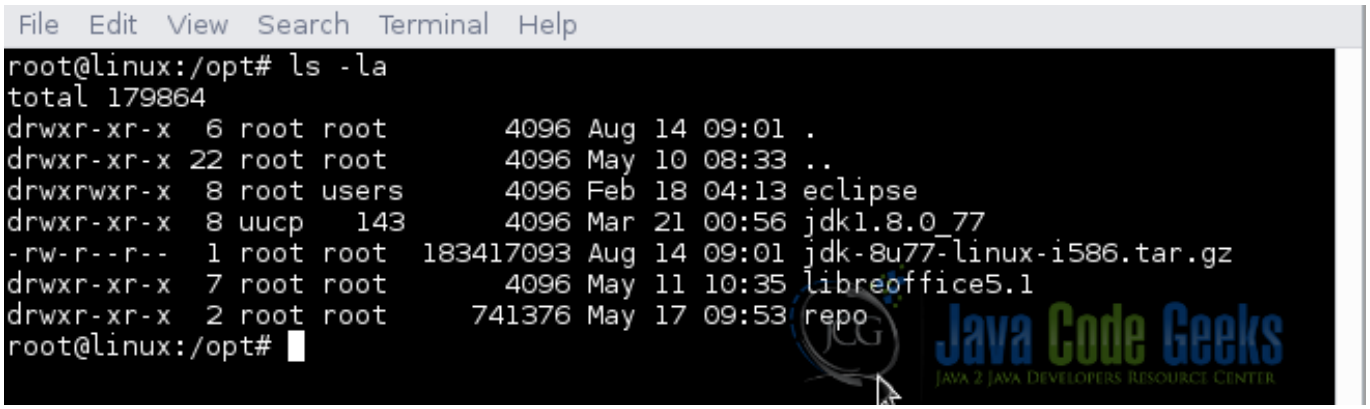

Figure 1.4: Extract the JDK

Update the alternatives to make Ubuntu aware of the JDK installation.

Update alternatives

```
sudo update-alternatives --install "/usr/bin/java" java "/opt/jdk1.8.0_77/bin/java" 1
sudo update-alternatives --install "/usr/bin/javac" javac "/opt/jdk1.8.0_77/bin/javac" 1
sudo update-alternatives --set java /opt/jdk1.8.0_77/bin/java
sudo update-alternatives --set javac /opt/jdk1.8.0_77/bin/javac
```

After we update the alternatives, we are going to check the installation. Execute the command:

Java Version

```
java -version
```

and you should get the output:

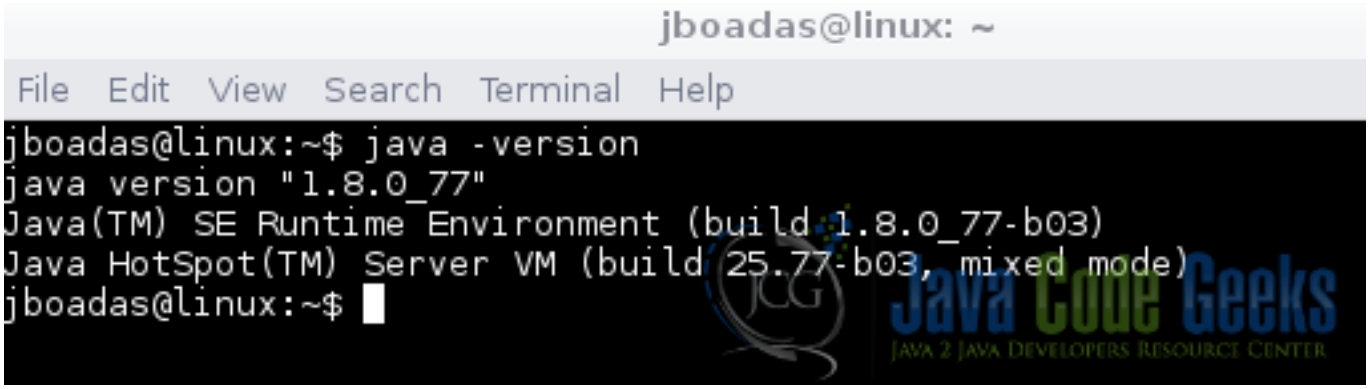

Figure 1.5: Java Version

1.7 Install Tomcat server

We are going to extract the Tomcat server in the `/opt` directory.

Extract Tomcat

```
sudo mkdir /opt/tomcat
sudo tar xvf apache-tomcat-8.0.33.tar.gz -C /opt/tomcat --strip-components=1
```

Create a Tomcat group to use with the server

Tomcat group

```
sudo groupadd tomcat
```

Create a Tomcat user to avoid use the root user with the Tomcat server

Tomcat user

```
sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat
```

Update the permissions of the Tomcat server to use with the new user and group.

Update permissions

```
cd /opt/tomcat
sudo chgrp -R tomcat conf
sudo chmod g+rwx conf
sudo chmod g+r conf/*
sudo chown -R tomcat work/ temp/ logs/
```

Create a Tomcat start script.

Init Tomcat script

```
vi /etc/init/tomcat.conf
```

The `tomcat.conf` script is used by the operative system to start the Tomcat service at boot time. This script is used to start and stop the service when is needed.

tomcat

```
description "Tomcat Server"
start on runlevel [2345]
stop on runlevel [!2345]
setuid tomcat
setgid tomcat
env JAVA_HOME=/opt/jdk1.8.0_77/jre/
env CATALINA_HOME=/opt/tomcat
exec $CATALINA_HOME/bin/catalina.sh run
```

`start on runlevel [2345]` Starts the service on these run levels

`stop on runlevel [!2345]` Stops the service on these run levels

`setuid tomcat` Sets the tomcat user.

`setgid tomcat` Sets the tomcat gtroup.

`env JAVA_HOME=/opt/jdk1.8.0_77/jre/` Exports the Java home.

`env CATALINA_HOME=/opt/tomcat` Exports the tomcat home.

`exec $CATALINA_HOME/bin/catalina.sh run` Runs the server.

1.8 Starts the Tomcat server

Go to the `opt/tomcat/bin` directory and execute the following command.

console

```
./catalina.sh start
```

You should see the following output

console

```
Using CATALINA_BASE:   /opt/tomcat
Using CATALINA_HOME:   /opt/tomcat
Using CATALINA_TMPDIR: /opt/tomcat/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /opt/tomcat/bin/bootstrap.jar:/opt/tomcat/bin/tomcat-juli.jar
Tomcat started.
root@linux:/opt/tomcat/bin#
```

Now its time of test our server. Open your browser in the URL `https://localhost:8080` and you should see the following page.

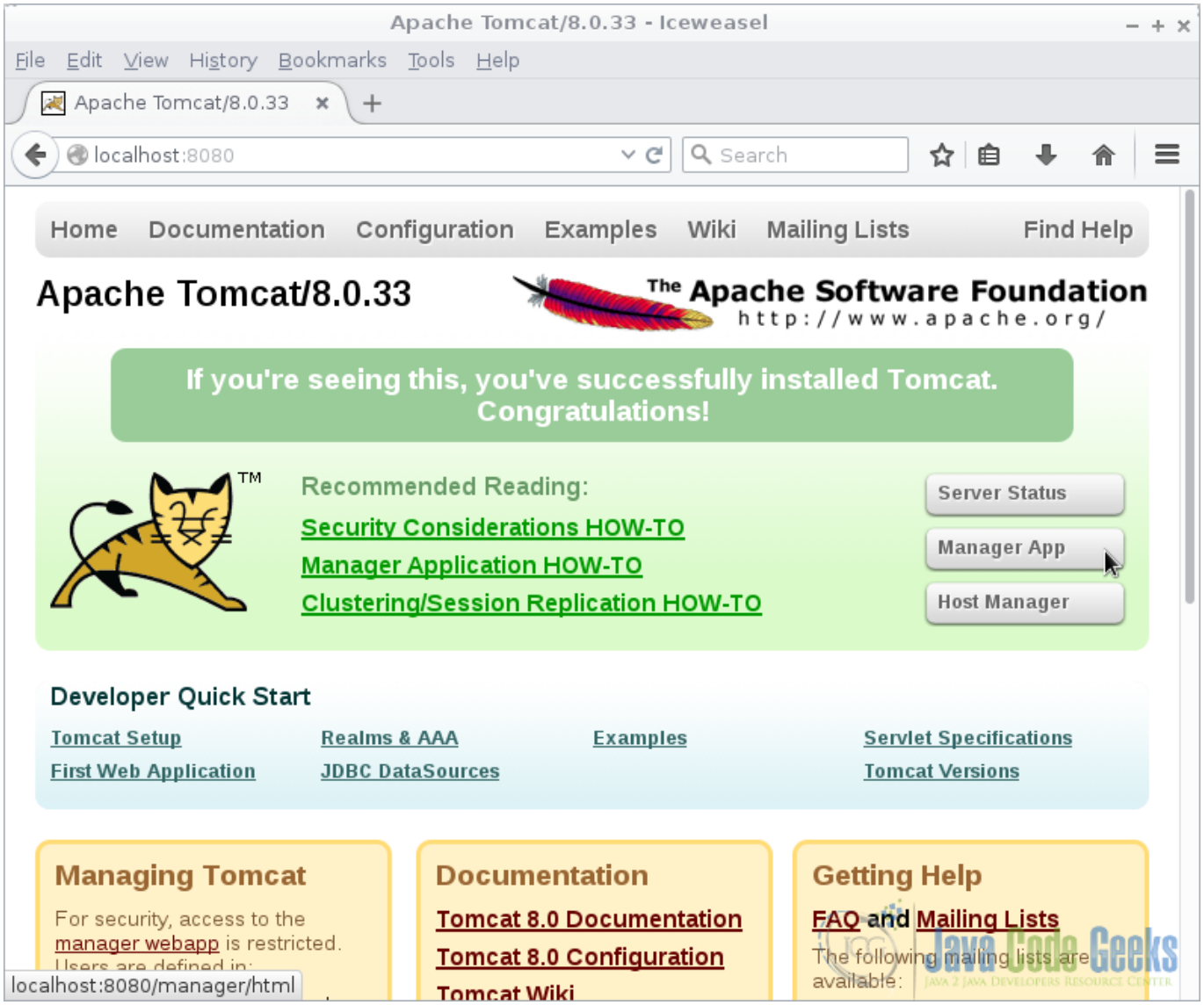

Figure 1.6: Tomcat Welcome

1.9 Activate the manager

To access the Tomcat manager we need to create a user with the privileges to do that. Edit the file `/opt/tomcat/conf/tomcat-users.xml`

In this file we are going to define the users to access the tomcat manager.

tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users xmlns="https://tomcat.apache.org/xml"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
        <user username="admin" password="admin" roles="manager-gui,admin-gui"/>
</tomcat-users>
```

`user username="admin" password="admin" roles="manager-gui,admin-gui"` Here we are defining a user admin with the password admin with the roles manager-gui and admin-gui Now restart the server and open again the URL `https://localhost:8080`. This time click on the `Manager App` button. No Tomcat will ask for credentials. You should see the following screen.

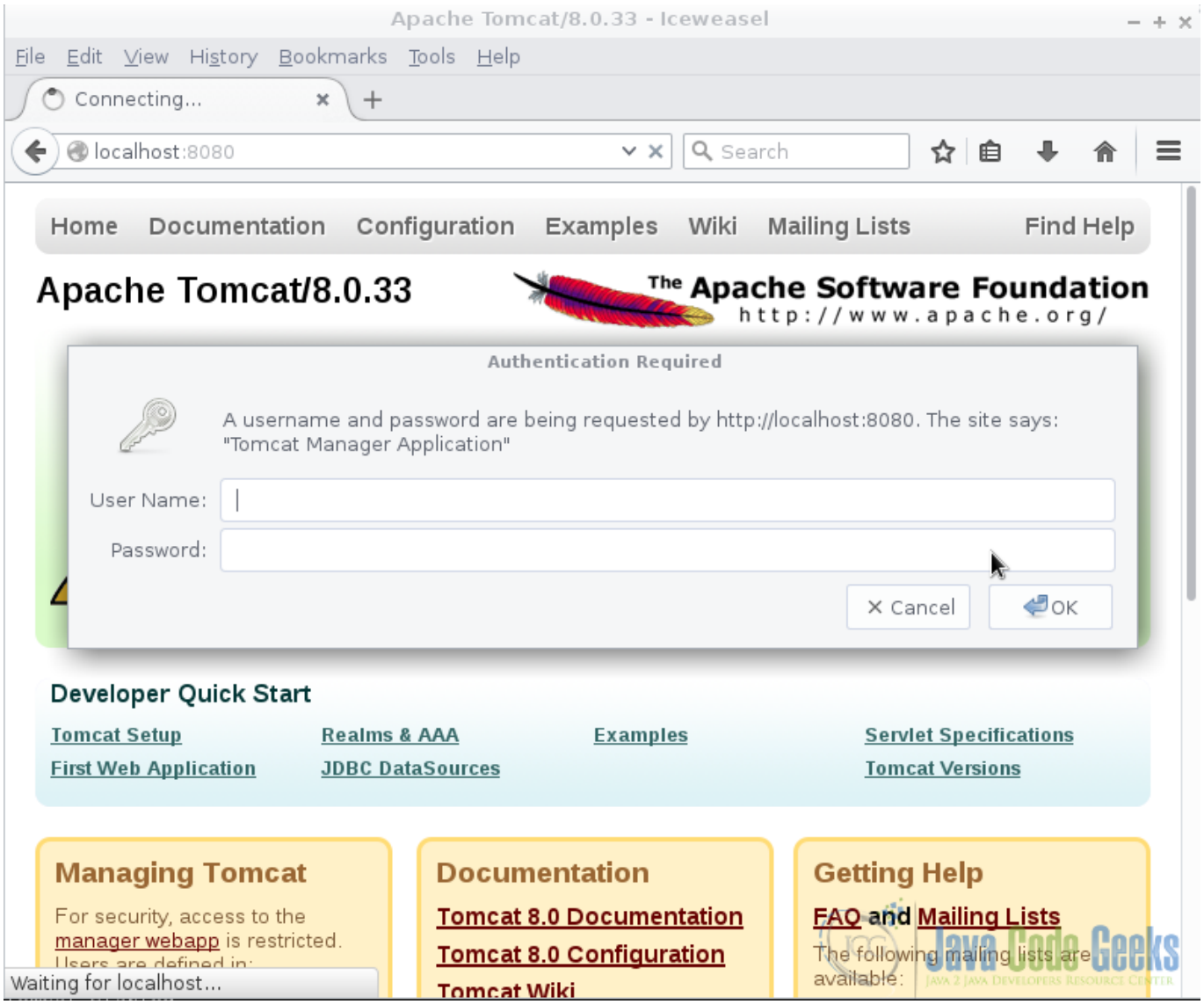

Figure 1.7: Tomcat Login

In the User Name write admin. In the Password write admin,then press enter. You should see the following screen.

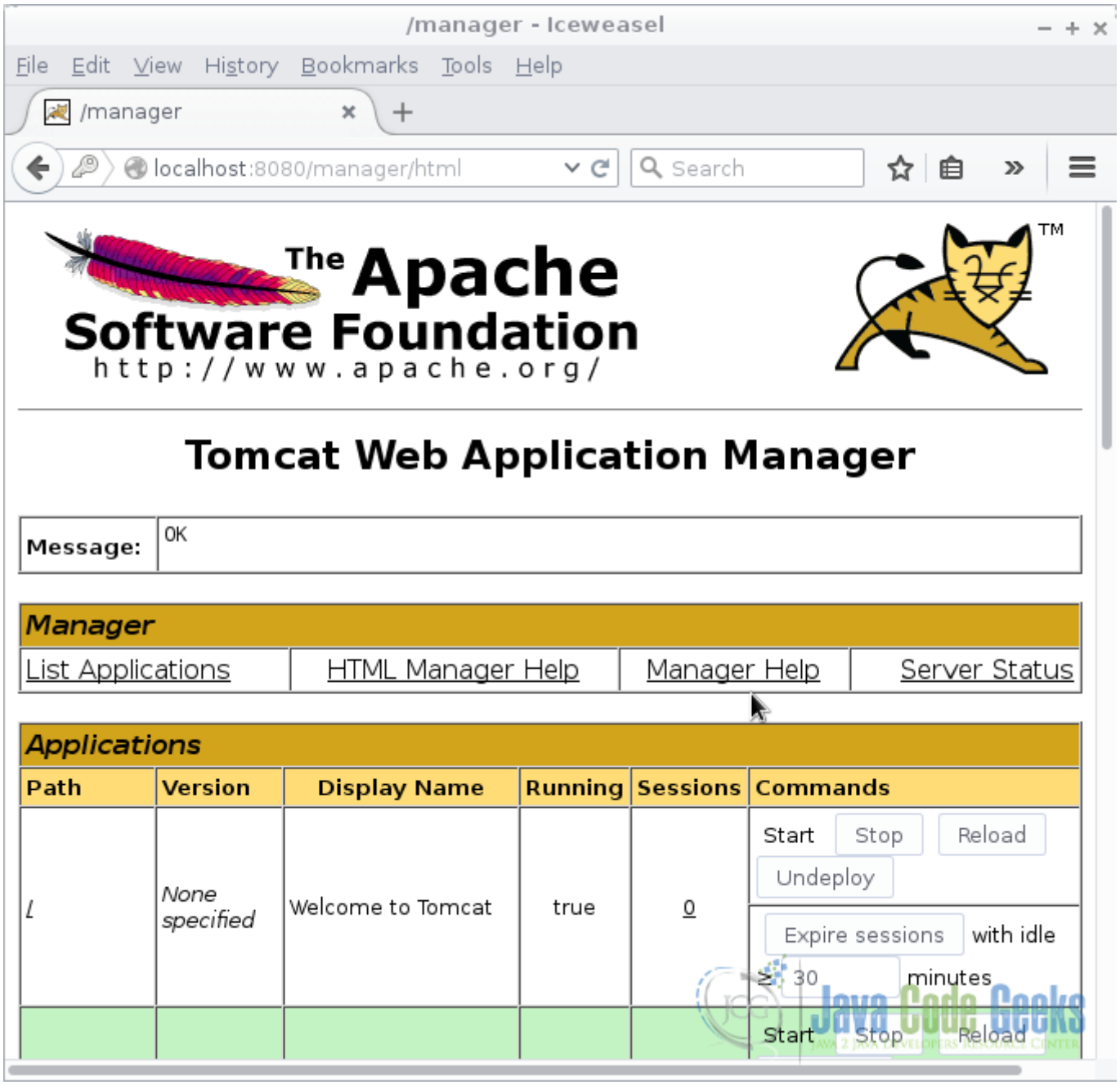

Figure 1.8: Tomcat Web Manager

1.10 Create a SSL Certificate

Run the following command to generate the certificate to make Tomcat support SSL. Generate Certificate

keytool -genkey -alias tomcat -keyalg RSA -keystore /opt/tomcat/keystore/tomcat

The tool is going to ask some questions to feed the certificate. The certificate is going to be in the folder /opt/tomcat/keystore/tomcat and the name of the certificate is tomcat. You can check the certificate with the command `keytool -list -keystore /opt/tomcat/keystore/tomcat`.

1.11 Use the certificate in Tomcat

Edit the file `/opt/tomcat/conf/server.xml` and add an SSL connector.

Connector

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"
                           keystoreFile="C:\Java\apache-tomcat-8.0.23\keystore\tomcat"
                   keystorePass="changeit" />
```

Restart tomcat and you are done. Now you can run your Applications under HTTPS in Tomcat.

1.12 Complete source code

tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users xmlns="https://tomcat.apache.org/xml"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
        <user username="admin" password="admin" roles="manager-gui,admin-gui"/>
</tomcat-users>
```

server.xml

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener"/>
  <Listener SSLEngine="on" className="org.apache.catalina.core.AprLifecycleListener"/>
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener"/>
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener"/>
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener"/>
  <GlobalNamingResources>
    <Resource auth="Container" description="User database that can be updated and saved" ↩
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory" name="UserDatabase" ↩
        pathname="conf/tomcat-users.xml" type="org.apache.catalina.UserDatabase"/>
  </GlobalNamingResources>
  <Service name="Catalina">
    <Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443 ↩
        "/>
    <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"
                           keystoreFile="C:\Java\apache-tomcat-8.0.23\keystore\tomcat"
                   keystorePass="changeit" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443"/>
    <Engine defaultHost="localhost" name="Catalina">
      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName=" ↩
            UserDatabase"/>
      </Realm>
      <Host appBase="webapps" autoDeploy="true" name="localhost" unpackWARs="true">
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs" ↩
            pattern="%h %l %u %t "%r" %s %b" prefix="localhost_access_log" suffix=".txt"/>

      </Host>
    </Engine>
```

```
  </Service>
</Server>
```

tomcat

```
description "Tomcat Server"
start on runlevel [2345]
stop on runlevel [!2345]
setuid tomcat
setgid tomcat
env JAVA_HOME=/opt/jdk1.8.0_77/jre/
env CATALINA_HOME=/opt/tomcat
exec $CATALINA_HOME/bin/catalina.sh run
```

1.13 Running the example

Run the command `/opt/tomcat/bin/catalina.sh start` to start the server. Open the browser in the URL `https://localhost:8080` to verify the server is running. Run the command `/opt/tomcat/bin/catalina.sh stop` to stop the server. Reboot the machine and verify that the script is starting the Tomcat server.

1.14 Results

You get a Tomcat server ready to deploy your Java war applications.

1.15 Download the Source Code

This was an example of: Tomcat on Ubuntu Linux.

Download

You can download the Eclipse project here: **TomcatOnUbuntu**

Chapter 2

How to Start and Restart Tomcat Server as a Service

Apache Tomcat is a web server and servlet container that is used to serve Java applications. A servlet is a Java technology-based Web component, managed by a container, that generates dynamic content.

2.1 The tools

- Debian based Linux distribution
- Microsoft Windows
- Java JDK
- Apache Tomcat

2.2 Introduction

When you download Apache Tomcat binary distribution, you need to unzip the folder. Every time you are going to use the server, you need to start and stop its service manually.

Most of the time, is needed to start the server with the system. To start/stop/restart the server, Tomcat provide some utilities. In this example we are going to install tomcat as a service and use the utilities provided by Tomcat to start/stop/restart the tomcat service.

2.3 Prerequisites

- JDK installed

2.4 Download Tomcat

Go to the page

- https://tomcat.apache.org/download-80.cgi and download the tomcat server as a zip compressed file for windows.

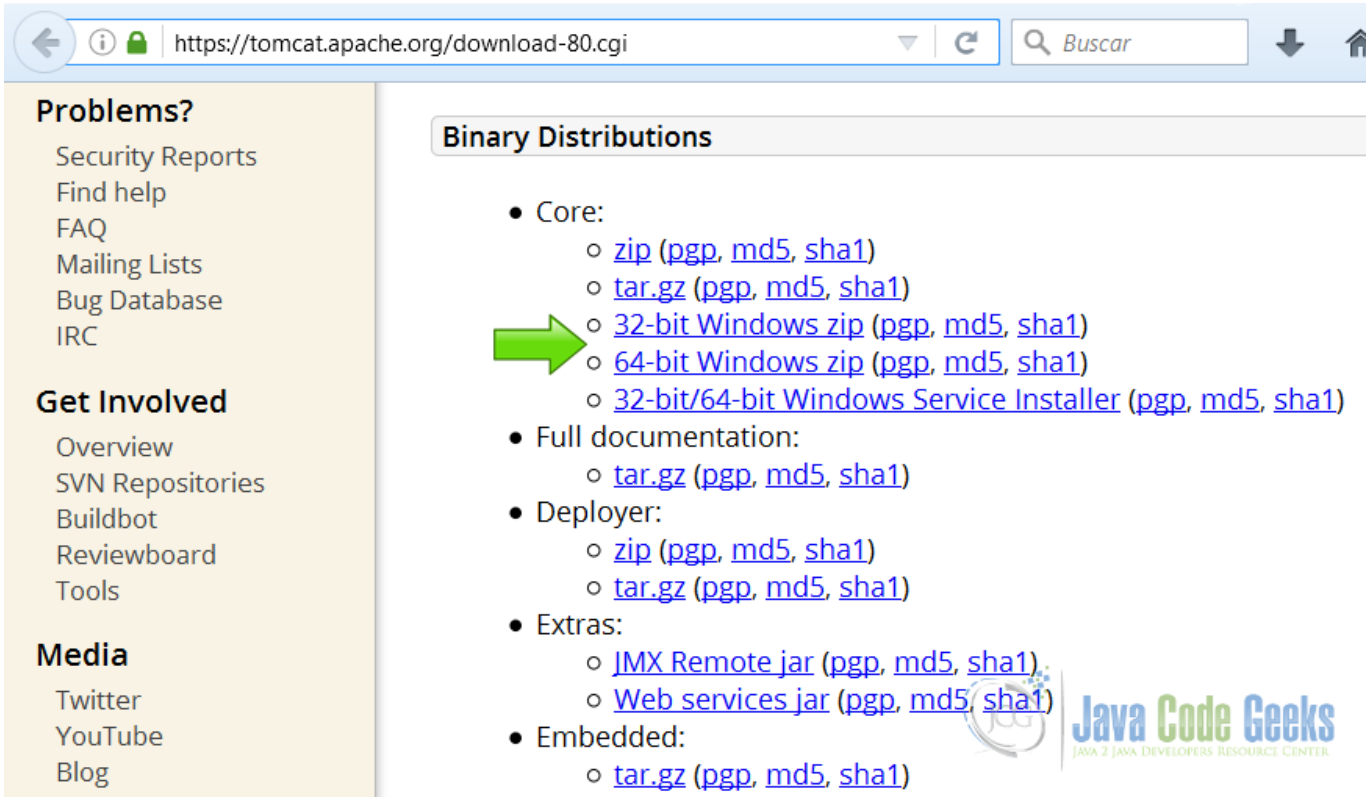

Figure 2.1: Download Tomcat for Windows

Download the tomcat server as a tar.gz compressed file for Linux.

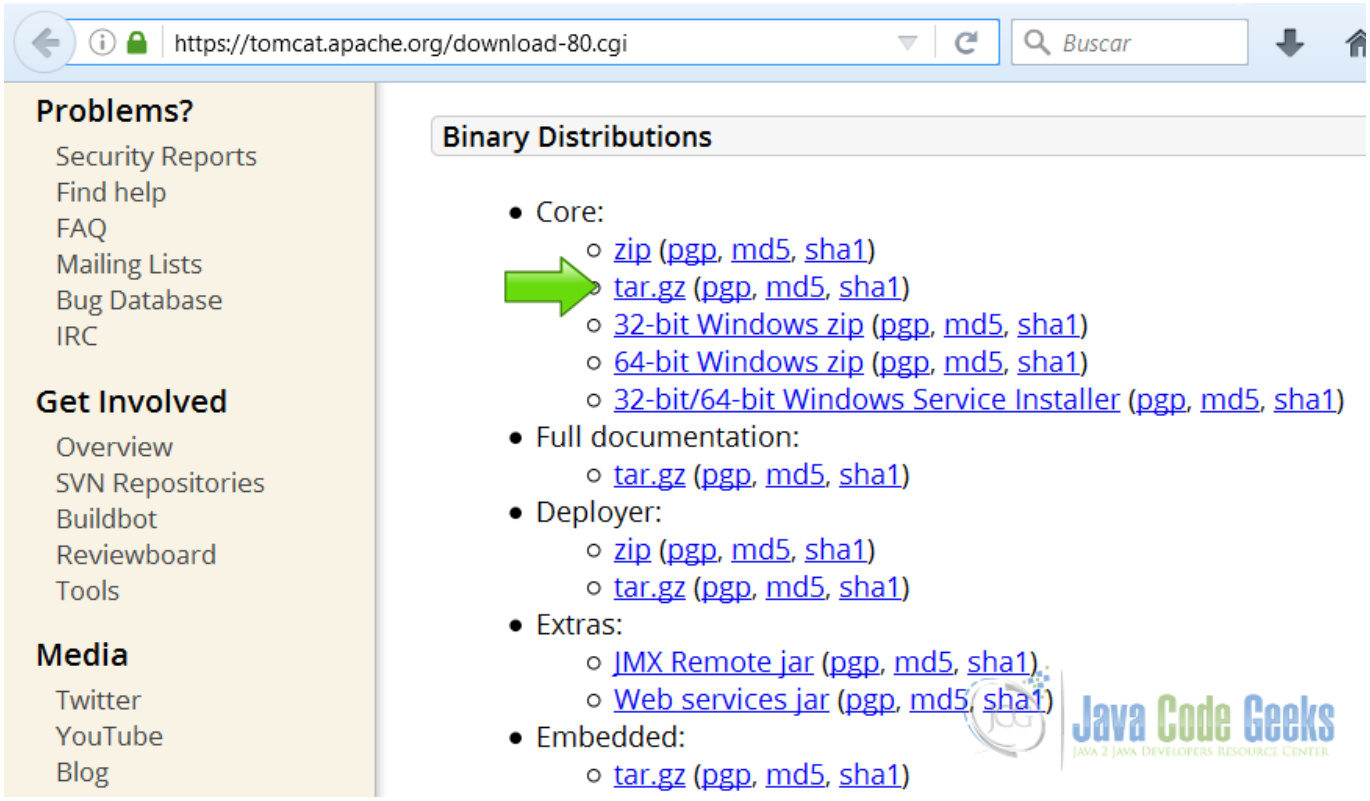

Figure 2.2: Download Tomcat for Linux

2.5 Tomcat in Windows

2.5.1 Unzip Tomcat

Unzip the tomcat downloaded zip in the folder you want to have it.

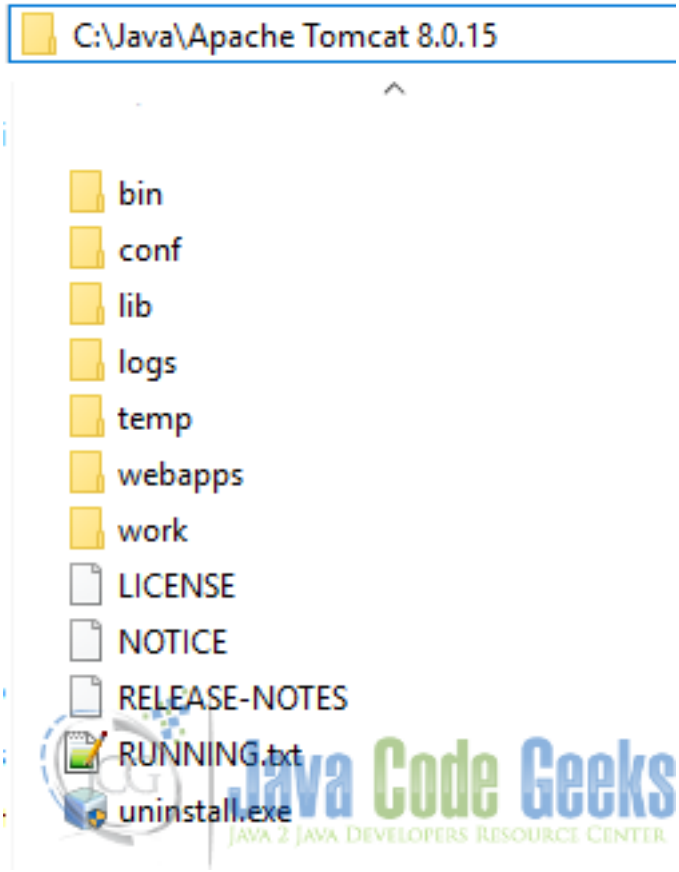

Figure 2.3: Unzip tomcat windows

2.5.2 Open the terminal console

Open the terminal and go to the folder where Tomcat is unzipped.

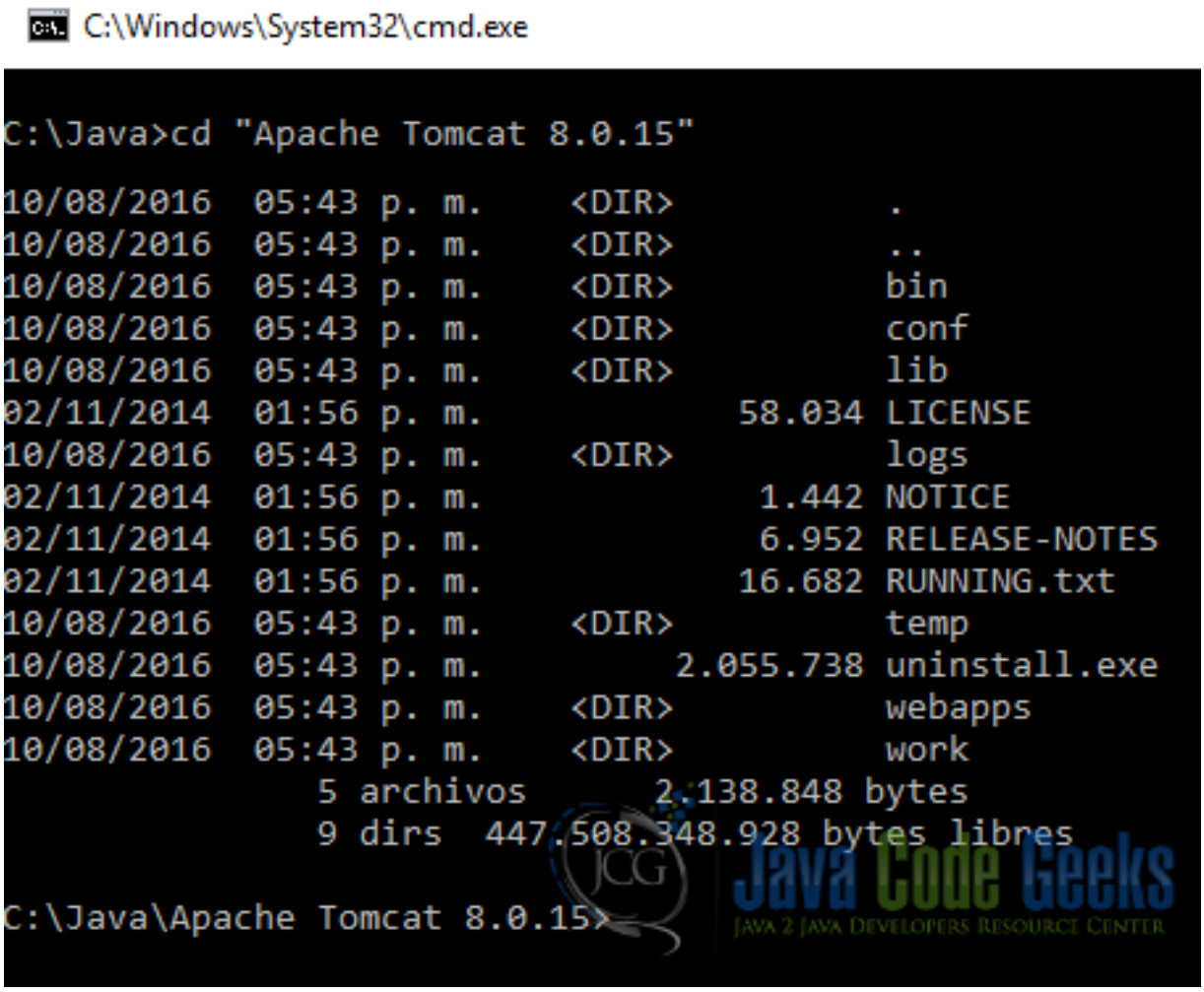

Figure 2.4: Tomcat folder

2.5.3 Install service

Go to the bin folder and type the command:

Install service

```
C:\Java\Apache Tomcat 8.0.27\bin>service install
```

This command install the Tomcat service in Windows. You should get the following output

Install service output

```
Installing the service 'Tomcat8' ...
Using CATALINA_HOME:    "C:\Java\Apache Tomcat 8.0.27"
Using CATALINA_BASE:    "C:\Java\Apache Tomcat 8.0.27"
Using JAVA_HOME:        "C:\Java\jdk1.8.0_40"
Using JRE_HOME:         "C:\Java\jdk1.8.0_40\jre"
Using JVM:              "C:\Java\jdk1.8.0_40\jre\bin\server\jvm.dll"
The service 'Tomcat8' has been installed.
```

2.5.4 Check service

You can check if the service is properly installed with the windows commands for services. Type the command

Check service

```
C:\Java\Apache Tomcat 8.0.27\bin>sc query Tomcat8
```

You should get an output similar to this:

sc query output

```
SERVICE_NAME: Tomcat8
TYPE                        : 10  WIN32_OWN_PROCESS
STATUS                      : 1  STOPPED
WIN32_OUTPUT_CODE               : 1077  (0x435)
SERVICE_OUTPUT_CODE             : 0  (0x0)
CONTROL_POINT               : 0x0
WAIT                        : 0x0
```

When you install the service, the service is stopped.

2.5.5 Start service

To start the service type this command:

Start tomcat service

```
C:\Java\Apache Tomcat 8.0.27\bin>sc start tomcat8
```

You should get an output similar to this:

Start service output

```
SERVICE_NAME: tomcat8
TYPE               : 10  WIN32_OWN_PROCESS
STATUS             : 2  START_PENDING
                                    (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
WIN32_OUTPUT_CODE  : 0  (0x0)
SERVICE-OUTPUT_CODE: 0  (0x0)
COMPROB_POINT      : 0x0
INITIAL-STATUS     : 0x7d0
PID                : 908
MARKS              :
```

It takes a while to start the service. Note that the status starts pending. You can check the service status with:

Check service

```
C:\Java\Apache Tomcat 8.0.27\bin>sc query Tomcat8
```

Wait until the service is started.

sc query output

```
SERVICE_NAME: Tomcat8
TYPE                        : 10  WIN32_OWN_PROCESS
STATUS                      : 3 STARTED
WIN32_OUTPUT_CODE               : 1077  (0x435)
SERVICE_OUTPUT_CODE             : 0  (0x0)
CONTROL_POINT               : 0x0
WAIT                        : 0x0
```

Open the URL `https://localhost:8080` in the browser and check that Tomcat is working. You should get the Tomcat Welcome screen:

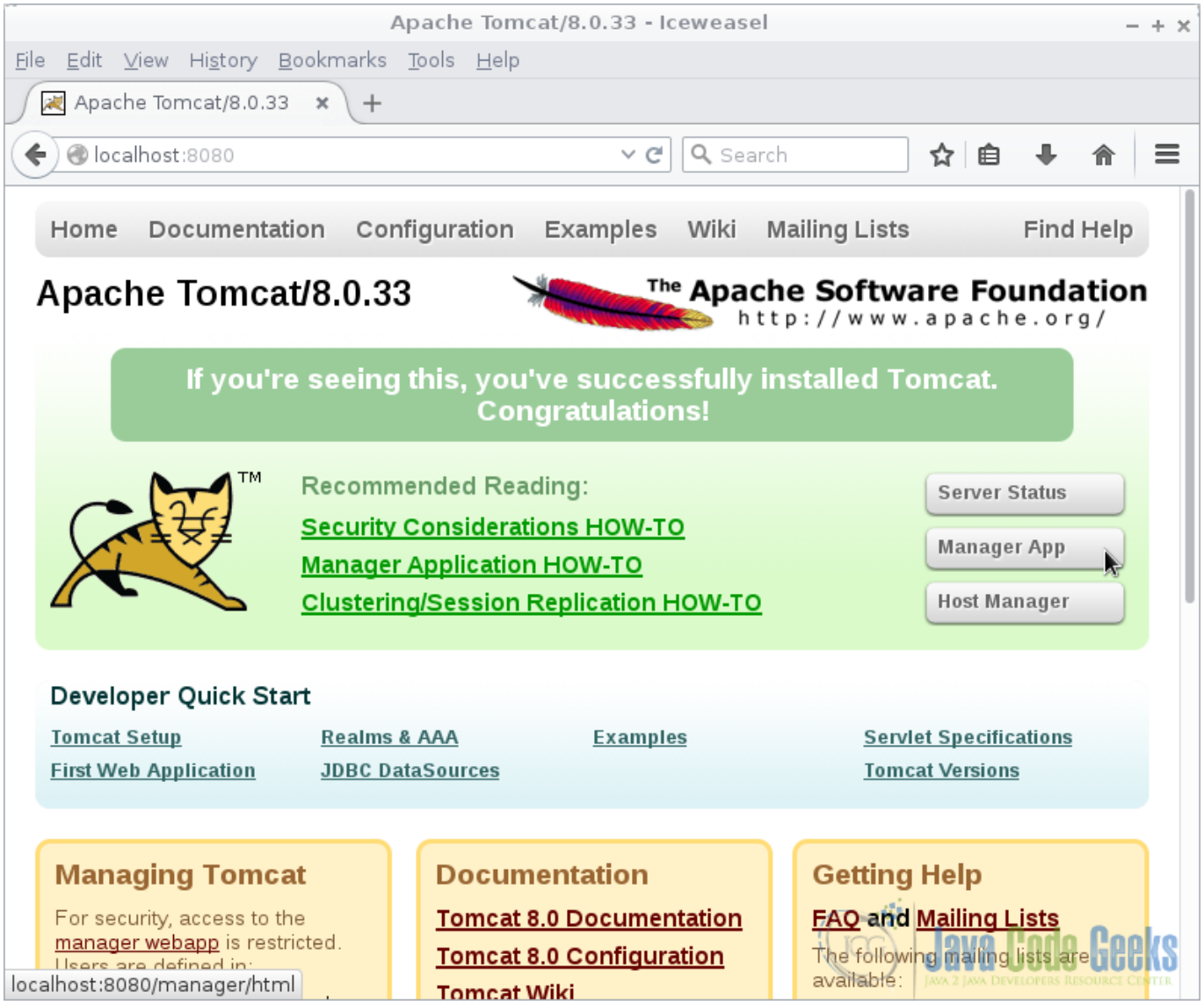

Figure 2.5: Tomcat Welcome

2.5.6 Stop service

To stop the Tomcat service type this command:

Stop tomcat service

```
C:\Java\Apache Tomcat 8.0.27\bin>sc stop tomcat8
```

You should get an output similar to this:

Stop service output

```
NOMBRE_SERVICIO: tomcat8
TYPE               : 10  WIN32_OWN_PROCESS
STATUS             : 3  STOP_PENDING
```

```
                                                    (NOT_STOPPABLE, NOT_PAUSABLE,  ↩
                                                        IGNORES_SHUTDOWN)
WIN32_OUTPUT_CODE  : 0  (0x0)
SERVICE-OUTPUT_CODE: 0  (0x0)
COMPROB_POINT      : 0x0
INITIAL-STATUS     : 0x7d0
PID                : 908
MARKS              :
```

It takes a while to stop the service. Note the status `STOP_PENDING`. You can check the service status with:

Check service

```
C:\Java\Apache Tomcat 8.0.27\bin>sc query Tomcat8
```

Wait until the service is stopped.

2.5.7 Autostart service

If you need to start the service when Windows start, type the following command:

Autostart service

```
C:\Java\Apache Tomcat 8.0.27\bin>sc config Tomcat8 start= auto
```

You should get an output similar to this:

Autostart service output

```
[SC] ChangeServiceConfig OK
```

Now restart the computer and check that Tomcat is starting when the system starts.

2.5.8 Remove service

If you need to remove the service, type the following command:

Remove service

```
C:\Java\Apache Tomcat 8.0.27\bin>service remove
```

And you should get the following output:

Remove service output

```
Removing the service 'Tomcat8' ...
Using CATALINA_BASE:    "C:\Java\Apache Tomcat 8.0.27"
The service 'Tomcat8' has been removed
```

2.6 Tomcat in Linux

Every Linux distribution has its own way to manage services. This procedure should work in deb based distributions. We are going to make this on Debian 8 Linux distribution.

2.6.1 Uncompress the tomcat folder

Open your Linux terminal.

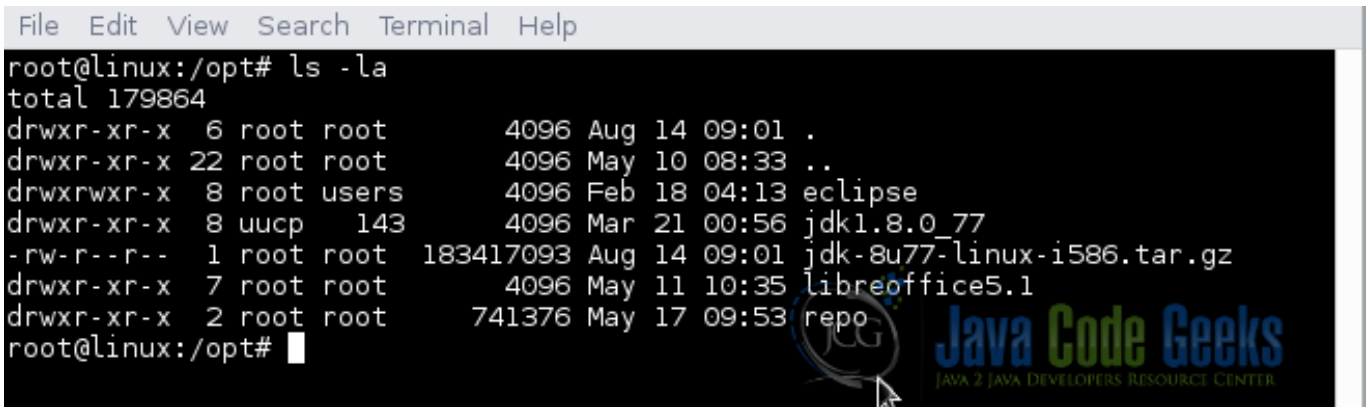

Figure 2.6: Linux terminal

Work as a superuser. Type the following command:

Superuser login `su` write your superuser password.

Copy the tomcat compressed file to the `/opt` directory.

Copy compressed file

```
cp /path_to_file/tomcat8.tar.gz /opt
```

go to the `/opt` directory.

Go to opt

```
cd /opt
```

Uncompress the tomcat file

Uncompress tomcat

```
tar -zxvf tomcat8.tar.gz
```

2.6.2 Create the service script

To work with tomcat as a service in Linux, we are going to create a script for this purpose

Init scripts folder

```
cd /etc/init.d
```

In this folder create the file `tomcat and give the appropriate permissions to the file.`

Script permissions

```
chown tomcat:tomcat tomcat8
chmod 755 tomcat8
```

Edit the file with your favorite editor.

Edit script

```
vi tomcat8
```

2.6.3 Script content

2.6.3.1 Script header

Script header

```
#!/bin/bash

### BEGIN INIT INFO
# Provides:          tomcat8
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Apache Tomcat
# Description:       Open source web server and Servlet container
### END INIT INFO
```

The header of the script is very important because it is used by the system to properly execute the script.

`#!/bin/bash` Indicates that this is a bash script. `###BEGIN INIT INFO` This mark the beginning of the header.

`#Provides:tomcat8` Indicates what is provided by the script. `#Required-Start:$remote_fs $syslog` Runlevels required to start.

`#Default-Start:2 3 4 5` Default start runlevels. `#Default-Stop:0 1 6` Default stop runlevels.

`#Short-Description:Apache Tomcat` A short description `#Description:Open source web server and Servlet container` A long description `###END INIT INFO` Mark the end of the header.

2.6.3.2 Initialize variables

Initialization variables

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export PATH=$JAVA_HOME/bin:$PATH
export CATALINA_HOME=/opt/apache-tomcat-8.0.33
export CATALINA_BASE=/opt/apache-tomcat-8.0.33
TOMCAT_USAGE="Usage: $0 {\e[00;32mstart\e[00m|\e[00;31mstop\e[00m|\e[00;32mstatus\e[00m|\e ↩
    [00;31mrestart\e[00m}"
SHUTDOWN_WAIT=20
```

`export JAVA_HOME=/usr/lib/jvm/java-8-oracle` JAVA_HOME is the directory where the JDK is located. `export PATH=$JAVA_HOME/bin:$PATH` The bin JAVA_HOME is exported to the PATH to use it everywhere.

`export CATALINA_HOME=/opt/apache-tomcat-8.0.33` Location of Tomcat directory. `export CATALINA_BASE=/opt/apache-tomcat-8.0.33` Location of Tomcat directory.

`TOMCAT_USAGE="Usage:$0 {e[00;32mstarte[00m|e[00;31mstope[00m|e[00;32mstatuse[00m|e[00;31mrestarte[00m}"` Feedback message showed when you run the script without parameters

`SHUTDOWN_WAIT=20` Waiting time to shutdown the server, to allow the server finish any pending tasks.

2.6.3.3 Print tomcat pid function

Tomcat PID

```
tomcat_pid() {
        echo `ps -fe | grep $CATALINA_BASE | grep -v grep | tr -s " "|cut -d" " -f2`
}
```

This function prints the tomcat process id using some terminal utilities.

`echo `ps -fe | grep $CATALINA_BASE | grep -v grep | tr -s " "|cut -d" " -f2`` prints the tomcat process id.

2.6.3.4 Print tomcat status function

Status function

```
status(){
  pid=$(tomcat_pid)
  if [ -n "$pid" ]; then echo -e "\e[00;32mTomcat is running with pid: $pid\e[00m"
  else echo -e "\e[00;31mTomcat is not running\e[00m"
  fi
}
```

`pid=$(tomcat_pid)` Gets the tomcat pid. `if [ -n "$pid" ];then echo -e "e[00;32mTomcat is runni ng with pid:$pide[00m"` If the pid exists then tomcat is running. `else echo -e "e[00;31mTomcat is not runninge[00m"` Prints the tomcat is not running.

2.6.3.5 Start tomcat function

Start function

```
start() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo -e "\e[00;31mTomcat is already running (pid: $pid)\e[00m"
  else
    echo -e "\e[00;32mStarting tomcat\e[00m"
    sh $CATALINA_HOME/bin/startup.sh
    status
  fi
  return 0
}
```

This function is running when you run the script with the start parameter.

`pid=$(tomcat_pid)` Gets the tomcat pid `if [ -n "$pid" ]` If the pid exists.

`echo -e "e[00;31mTomcat is already running (pid:$pid)e[00m"` The tomcat server is running. `echo -e "e[00;32mStarting tomcate[00m"` If the tomcat server is not running print the message that tomcat is starting.

`sh $CATALINA_HOME/bin/startup.sh` Run the startup script provided by tomcat. `status` Print the status.

`return 0` Return without errors.

2.6.3.6 Stop tomcat function

Stop function

```
stop() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo -e "\e[00;31mStoping Tomcat\e[00m"
        sh $CATALINA_HOME/bin/shutdown.sh

    let kwait=$SHUTDOWN_WAIT
    count=0;
    until [ `ps -p $pid | grep -c $pid` = '0' ] || [ $count -gt $kwait ]
    do
      echo -n -e "\n\e[00;31mwaiting for processes to exit\e[00m";
      sleep 1
      let count=$count+1;
```

```
    done

    if [ $count -gt $kwait ]; then
      echo -n -e "\n\e[00;31mkilling processes which didn't stop after $SHUTDOWN_WAIT ↩
          seconds\e[00m"
      kill -9 $pid
    fi
  else
    echo -e "\e[00;31mTomcat is not running\e[00m"
  fi

  return 0
}
```

This function stops the Tomcat server

`pid=$(tomcat_pid)` Gets the Tomcat PID

`if [ -n "$pid" ]` If the PID doesn't exist

`echo -e "e[00;31mStoping Tomcate[00m"` Prints the message that Tomcat is stopping

`sh $CATALINA_HOME/bin/shutdown.sh` Runs the script provided by tomcat to stop the server.

`let kwait=$SHUTDOWN_WAIT` Gets the timeout variable.

`count=0;` Initializes a counter.

`until [ `ps -p $pid | grep -c $pid = *0*] || [$count -gt $kwait]` Checks if the pid still exist.

`echo -n -e "ne[00;31mwaiting for processes to exite[00m";` Prints a waiting message.

`sleep 1` Sleeps.

`let count=$count+1;` Updates the counter.

`if [ $count -gt $kwait ];then` if the counter is equal to the waiting time.

`echo -n -e "ne[00;31mkilling processes which didn't stop after $SHUTDOWN_WAIT secondse [00m"` Prints a message that the server didn't stop before the waiting time.

`kill -9 $pid` Forces to quit the server process.

`echo -e "e[00;31mTomcat is not runninge[00m"` Prints a message that the server is not running.

`return 0` Returns without errors.

2.6.3.7 Script main body

When we run the script the parameters are selected using a case statement, to choose the right option.

Script main body

```
case $1 in
        start)
          start
        ;;
        stop)
          stop
        ;;
        restart)
          stop
          start
        ;;
        status)
          status
        ;;
```

```
        *)
          echo -e $TOMCAT_USAGE
        ;;
esac
exit 0
```

`case $1 in` Examines the script parameter.

`start)` If the parameter is start.

`start` Starts the server.

`stop)` If the parameter is stop.

`stop` Stops the server.

`restart)` If the parameter is restart.

`stop` Stops the server.

`start` Start the server.

`status)` If the parameter is status.

`status` Prints the status.

`*)` If there is no parameters other any other parameter.

`echo -e $TOMCAT_USAGE` Prints the help message.

`esac` End the case statement.

`exit 0` Exits without errors.

2.7 The complete Linux source code

tomcat8

```
#!/bin/bash

### BEGIN INIT INFO
# Provides:          tomcat8
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Apache Tomcat
# Description:       Open source web server and Servlet container
### END INIT INFO

export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export PATH=$JAVA_HOME/bin:$PATH
export CATALINA_HOME=/opt/apache-tomcat-8.0.33
export CATALINA_BASE=/opt/apache-tomcat-8.0.33
TOMCAT_USAGE="Usage: $0 {\e[00;32mstart\e[00m|\e[00;31mstop\e[00m|\e[00;32mstatus\e[00m|\e ↩
    [00;31mrestart\e[00m}"
SHUTDOWN_WAIT=20

tomcat_pid() {
        echo `ps -fe | grep $CATALINA_BASE | grep -v grep | tr -s " "|cut -d" " -f2`
}

start() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
```

```
  then
    echo -e "\e[00;31mTomcat is already running (pid: $pid)\e[00m"
  else
    echo -e "\e[00;32mStarting tomcat\e[00m"
    sh $CATALINA_HOME/bin/startup.sh
    status
  fi
  return 0
}

status(){
          pid=$(tomcat_pid)
          if [ -n "$pid" ]; then echo -e "\e[00;32mTomcat is running with pid: $pid\e[00m"
          else echo -e "\e[00;31mTomcat is not running\e[00m"
          fi
}

stop() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo -e "\e[00;31mStoping Tomcat\e[00m"
        sh $CATALINA_HOME/bin/shutdown.sh

    let kwait=$SHUTDOWN_WAIT
    count=0;
    until [ `ps -p $pid | grep -c $pid` = '0' ] || [ $count -gt $kwait ]
    do
      echo -n -e "\n\e[00;31mwaiting for processes to exit\e[00m";
      sleep 1
      let count=$count+1;
    done

    if [ $count -gt $kwait ]; then
      echo -n -e "\n\e[00;31mkilling processes which didn't stop after $SHUTDOWN_WAIT ↩
          seconds\e[00m"
      kill -9 $pid
    fi
  else
    echo -e "\e[00;31mTomcat is not running\e[00m"
  fi

  return 0
}

case $1 in

        start)
          start
        ;;

        stop)
          stop
        ;;

        restart)
          stop
          start
        ;;

        status)
                status
```

```
        ;;

        *)
                echo -e $TOMCAT_USAGE
        ;;

esac
exit 0
```

2.8 Make the script run at boot

This command is used to make Linux aware to run the script at startup.

Run at boot

```
update-rc.d /etc/init.d/tomcat8
```

2.9 Linux results

2.9.1 Run script without arguments

We run the command without arguments

tomcat

```
root@debian:~# /etc/init.d/tomcat8
```

The script shows the help.

tomcat script output

```
Usage: /etc/init.d/tomcat8 {start|stop|status|restart}
```

2.9.2 Run script with status parameter

We run the script with the status parameter

tomcat status

```
root@debian:~# /etc/init.d/tomcat8 status
```

If the service is running we get the following output:

tomcat status running output

```
Tomcat is running with pid: 619
```

If the service is not running we get the following output:

tomcat status not running output

```
Tomcat is not running
```

2.9.3 Run script with start parameter

We run the script with the start parameter

tomcat start

```
root@debian:~# /etc/init.d/tomcat8 start
```

If the server is running we get the output

tomcat start with server running output

```
Tomcat is already running (pid: 3445)
```

If the server is not running we get the output

tomcat start with server not running output

```
Starting tomcat
Using CATALINA_BASE:   /opt/apache-tomcat-8.0.33
Using CATALINA_HOME:   /opt/apache-tomcat-8.0.33
Using CATALINA_TMPDIR: /opt/apache-tomcat-8.0.33/temp
Using JRE_HOME:        /usr/lib/jvm/java-8-oracle
Using CLASSPATH:       /opt/apache-tomcat-8.0.33/bin/bootstrap.jar:/opt/apache-tomcat ↩
    -8.0.33/bin/tomcat-juli.jar
Tomcat started.
Tomcat is running with pid: 2715
```

2.9.4 Run script with stop parameter

We run the script with the stop parameter

tomcat stop

```
root@debian:~# /etc/init.d/tomcat8 stop
```

If the server is running we get the output

tomcat stop with server running output

```
Stoping Tomcat
Using CATALINA_BASE:   /opt/apache-tomcat-8.0.33
Using CATALINA_HOME:   /opt/apache-tomcat-8.0.33
Using CATALINA_TMPDIR: /opt/apache-tomcat-8.0.33/temp
Using JRE_HOME:        /usr/lib/jvm/java-8-oracle
Using CLASSPATH:       /opt/apache-tomcat-8.0.33/bin/bootstrap.jar:/opt/apache-tomcat ↩
    -8.0.33/bin/tomcat-juli.jar
```

If the server is not running we get the output

tomcat stop with server not running output

```
Tomcat is not running
```

2.9.5 Run script with restart parameter

We run the script with the restart parameter

tomcat restart

```
root@debian:~# /etc/init.d/tomcat8 restart
```

If the server is running we get the output

tomcat restart with server running output

```
Stoping Tomcat
Using CATALINA_BASE:   /opt/apache-tomcat-8.0.33
Using CATALINA_HOME:   /opt/apache-tomcat-8.0.33
Using CATALINA_TMPDIR: /opt/apache-tomcat-8.0.33/temp
Using JRE_HOME:        /usr/lib/jvm/java-8-oracle
Using CLASSPATH:       /opt/apache-tomcat-8.0.33/bin/bootstrap.jar:/opt/apache-tomcat ↩
    -8.0.33/bin/tomcat-juli.jar

waiting for processes to exit
Starting tomcat
Using CATALINA_BASE:   /opt/apache-tomcat-8.0.33
Using CATALINA_HOME:   /opt/apache-tomcat-8.0.33
Using CATALINA_TMPDIR: /opt/apache-tomcat-8.0.33/temp
Using JRE_HOME:        /usr/lib/jvm/java-8-oracle
Using CLASSPATH:       /opt/apache-tomcat-8.0.33/bin/bootstrap.jar:/opt/apache-tomcat ↩
    -8.0.33/bin/tomcat-juli.jar
Tomcat started.
Tomcat is running with pid: 2874
```

If the server is not running we get the output

tomcat restart with server not running output

```
Tomcat is not running
Starting tomcat
Using CATALINA_BASE:   /opt/apache-tomcat-8.0.33
Using CATALINA_HOME:   /opt/apache-tomcat-8.0.33
Using CATALINA_TMPDIR: /opt/apache-tomcat-8.0.33/temp
Using JRE_HOME:        /usr/lib/jvm/java-8-oracle
Using CLASSPATH:       /opt/apache-tomcat-8.0.33/bin/bootstrap.jar:/opt/apache-tomcat ↩
    -8.0.33/bin/tomcat-juli.jar
Tomcat started.
Tomcat is running with pid: 3571
```

2.10 Download the Source Code

This was an example of how to start and restart Tomcat server as a service.

Download

You can download the Linux script here: **tomcat8**

Chapter 3

Tomcat server.xml Configuration Example

Almost every application container will have some form of a server.xml file. It's basically where every meta-data or configurations that the container needs for it to complete it's initialization. This can be configured so that software designers and architects can inject services needed on runtime or upon destruction (stop). It is equally important to know this as to how every code or software works.

For this post, we will tackle on understanding and configuring tomcat apache server by analysing the server.xml file.

Pre-requisites:

- Installed Apache Tomcat 7. (get the source from apache tomcat site)

For the installation instructions, go here.

3.1 The Tomcat Installed Directory.

Once you installed tomcat, it will be placed in your local storage. For windows, it's usually in "Program Files" folder, for Mac or Linux, it can be on the `/user/var/opt` or `/User/<>/Application` folder. Once you're in the directory, you can see the different folders and files available:

- **bin**: for Tomcat's binaries and startup scripts.
- **conf**: global configuration applicable to all the webapps. The default installation provides:
  - `catalina.policy` for specifying security policy.
  - Two Properties Files: `catalina.properties` and `logging.properties`,
  - Four Configuration XML Files: `server.xml` (Tomcat main configuration file), `web.xml` (global web application deployment descriptors), `context.xml` (global Tomcat-specific configuration options) and `tomcat-users.xml` (a database of user, password and role for authentication and access control).

The `conf` also contain a sub-directory for each engine, e.g., `Catalina`, which in turn contains a sub-sub-directory for each of its hosts, e.g., `localhost`. You can place the host-specific context information (similar to`context.xml`, but named as `webapp.xml` for each webapp under the host).

lib: Keeps the JAR-file that are available to all webapps. The default installation include `servlet-api.jar` (Servlet), `jasper.jar` (JSP) and `jasper-el.jar` (EL). External JARs can be put here such as MySQL JDBC driver (`mysql-connector-java-5.1.{xx}-bin.jar`) and JSTL (`jstl.jar` and `standard.jar`).

logs: contains the engine logfile `Catalina.{yyyy-mm-dd}.log`, host logfile `localhost.{yyyy-mm-dd}.log`, and other application logfiles such as `manger` and `host-manager`. The access log (created by the`AccessLogValve`) is also kept here.

webapps: the default `appBase` - web applications base directory of the host `localhost`.

work: contains the translated servlet source files and classes of JSP/JSF. Organized in hierarchy of engine name (`Catalina`), host name (`localhost`), webapp name, followed by the Java classes package structure.

temp: temporary files.

3.2 Tomcat Architecture

Tomcat is an **HTTP server**. Tomcat is also a servlet container that can execute Java Servlet, and converting JavaServer Pages (JSP) and JavaServerFaces (JSF) to Java Servlet. Tomcat employs a hierarchical and modular architecture as shown below:

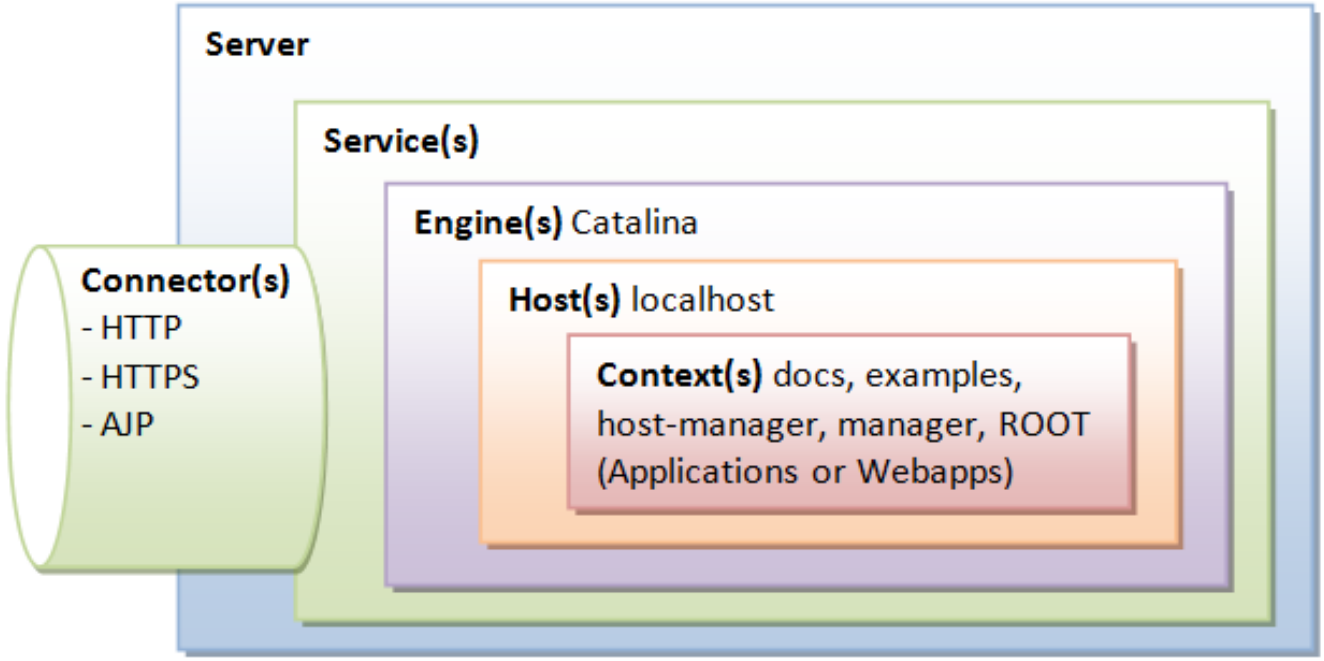

Figure 3.1: Tomcat Architecture

3.3 The Main Configuration File (server.xml)

Tomcat's main configuration file is the `server.xml`, kept under the `<CATALINA_HOME>/conf` directory. The default `server.xml` is reproduced as follows (after removing the comments and minor touch-ups):

server.xml

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
```

```
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>

  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

    <Engine name="Catalina" defaultHost="localhost">

      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
               resourceName="UserDatabase"/>
      </Realm>

      <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t "%r" %s %b" />
      </Host>
    </Engine>
  </Service>
</Server>
```

3.3.1 Server

Server (Line 2) is top component, representing an instance of Tomcat. It can contains one or more Services, each with its own Engines and Connectors.

```
<Server port="8005" shutdown="SHUTDOWN"> ..... </Server>
```

3.3.1.1 Common Attributes

- **className** - Java class name of the implementation to use. This class must implement the org.apache.catalina.Server interface. If no class name is specified, the standard implementation will be used.
- **address** - The TCP/IP address on which this server waits for a shutdown command. If no address is specified, localhost is used.
- **port** - The TCP/IP port number on which this server waits for a shutdown command. Set to -1 to disable the shutdown port.
- **shutdown** - The command string that must be received via a TCP/IP connection to the specified port number, in order to shut down Tomcat.

3.3.2 Listeners

The Server contains several Listeners (Lines 3-7). A Listener listens and responses to specific events. The JasperListener enables the Jasper JSP engine, and is responsible for re-compiling the JSP pages that have been updated.

```
<Listener className="org.apache.catalina.core.JasperListener" />
```

The GlobalResourcesLifecycleListener enables the global resources, and makes possible the use of JNDI for accessing resources such as databases.

```
<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
```

3.3.2.1 Common Attributes

- **SSLEngine** - Name of the SSLEngine to use. off: do not use SSL, on: use SSL but no specific ENGINE. The default value is on. This initializes the native SSL engine, which must be enabled in the APR/native connector by the use of the SSLEnabled attribute.
- **SSLRandomSeed** - Entropy source used to seed the SSLEngine's PRNG. The default value is builtin. On development systems, you may want to set this to /dev/urandom to allow quicker start times.
- **FIPSMode** - Set to on to request that OpenSSL be in FIPS mode (if OpenSSL is already in FIPS mode, it will remain in FIPS mode). Set to enter to force OpenSSL to enter FIPS mode (an error will occur if OpenSSL is already in FIPS mode). Set to require to require that OpenSSL already be in FIPS mode (an error will occur if OpenSSL is not already in FIPS mode).

3.3.3 Global Naming Resources

The element (Line 9-15) defines the JNDI (Java Naming and Directory Interface) resources, that allows Java software clients to discover and look up data and objects via a name. The default configuration defines a JNDI name called UserDatabase via the element (Line 10-14), which is a memory-based database for user authentication loaded from `conf/tomcat-users.xml`.

```
<GlobalNamingResources>
  <Resource name="UserDatabase" auth="Container"
            type="org.apache.catalina.UserDatabase"
            description="User database that can be updated and saved"
            factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
            pathname="conf/tomcat-users.xml" />
</GlobalNamingResources>
```

You can define other global resource JNDI such as MySQL database to implement connection pooling.

3.3.4 Services

A Service associates one or more Connectors to a Engine. The default configuration defines a Service called "Catalina", and associates two Connectors: HTTP and AJP to the Engine.

```
<Service name="Catalina"> ..... </Service>
```

3.3.4.1 Common Attributes

- **className** - Java class name of the implementation to use. This class must implement the org.apache.catalina.Service interface. If no class name is specified, the standard implementation will be used.
- **name** - The display name of this Service, which will be included in log messages if you utilize standard Catalina components. The name of each Service that is associated with a particular Server must be unique.

3.3.5 Connectors

A Connector is associated with a TCP port to handle communications between the Service and the clients. The default configuration defines two Connectors: HTTP/1.1: Handle HTTP communication and enable Tomcat to be an HTTP server. Clients can issue HTTP requests to the server via this Connector, and receive the HTTP response messages.

```
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />
```

The default chooses TCP port 8080 to run the Tomcat HTTP server, which is different from the default port number of 80 for HTTP production server. You can choose any number between 1024 to 65535, which is not used by any application, to run your Tomcat server. The connectionTimeout attribute define the number of milliseconds this connector will wait, after accepting a connection, for the request URI line (request message) to be presented. The default is 20 seconds. The redirect attribute re-directs the SSL requests to TCP port 8443. AJP/1.3: Apache JServ Protocol connector to handle communication between Tomcat server and Apache HTTP server.

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

You could run Tomcat and Apache HTTP servers together, and let the Apache HTTP server handles static requests and PHP; while Tomcat server handles the Java Servlet/JSP. Read "How To Configure Tomcat to work with Apache".

3.3.6 Containers

Tomcat refers to Engine, Host, Context, and Cluster, as container. The highest-level is Engine; while the lowest-level is Context. Certain components, such as Realm and Valve, can be placed in a container.

3.3.7 Engine

A Engine is the highest-level of a container. It can contains one or more Hosts. You could configure a Tomcat server to run on several hostnames, known as virtual host.

```
<Engine name="Catalina" defaultHost="localhost"/>
```

The Catalina Engine receives HTTP requests from the HTTP connector, and direct them to the correct host based on the hostname/IP address in the request header.

3.3.7.1 Common Attribute

- **backgroundProcessorDelay** - This value represents the delay in seconds between the invocation of the backgroundProcess method on this engine and its child containers, including all hosts and contexts. Child containers will not be invoked if their delay value is not negative (which would mean they are using their own processing thread). Setting this to a positive value will cause a thread to be spawn. After waiting the specified amount of time, the thread will invoke the backgroundProcess method on this engine and all its child containers. If not specified, the default value for this attribute is 10, which represent a 10 seconds delay.
- **className** - Java class name of the implementation to use. This class must implement the org.apache.catalina.Engine interface. If not specified, the standard value (defined below) will be used.
- **defaultHost** - The default host name, which identifies the Host that will process requests directed to host names on this server, but which are not configured in this configuration file. This name MUST match the name attributes of one of the Host elements nested immediately inside.
- **jvmRoute** - Identifier which must be used in load balancing scenarios to enable session affinity. The identifier, which must be unique across all Tomcat servers which participate in the cluster, will be appended to the generated session identifier, therefore allowing the front end proxy to always forward a particular session to the same Tomcat instance.
- **name** - Logical name of this Engine, used in log and error messages. When using multiple Service elements in the same Server, each Engine MUST be assigned a unique name.
- **startStopThreads** - The number of threads this Engine will use to start child Host elements in parallel. The special value of 0 will result in the value of Runtime.getRuntime().availableProcessors() being used. Negative values will result in Runtime.getRuntime().availableProcessors() + value being used unless this is less than 1 in which case 1 thread will be used. If not specified, the default value of 1 will be used.

3.3.8 Realm

A Realm is a database of user, password, and role for authentication (i.e., access control). You can define Realm for any container, such as Engine, Host, and Context, and Cluster.

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase ↩
      "/>
</Realm>
```

The default configuration defines a Realm (UserDatabaseRealm) for the Catalina Engine, to perform user authentication for accessing this engine. It uses the JNDI name UserDatabase defined in the GlobalNamingResources. Besides the UserDatabaseRealm, there are: JDBCRealm (for authenticating users to connect to a relational database via the JDBC driver); DataSourceRealm (to connect to a DataSource via JNDI; JNDIRealm (to connect to an LDAP directory); and MemoryRealm (to load an XML file in memory).

3.3.8.1 Common Attributes

- **className** - Java class name of the implementation to use. This class must implement the org.apache.catalina.Realm interface.

3.3.9 Hosts

A Host defines a virtual host under the Engine, which can in turn support many Contexts (webapps).

```
<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true"/>
```

The default configuration define one host called localhost. The appBase attribute defines the base directory of all the webapps, in this case, webapps. By default, each webapp's URL is the same as its directory name. For example, the default Tomcat installation provides four webapps: docs, examples, host-manager and manager under the webapps directory. The only exception is ROOT, which is identified by an empty string. That is, its URL is https://localhost:8080/. The unpackWARs specifies whether WAR-file dropped into the webapps directory shall be unzipped. For `unpackWARs="false"`, Tomcat will run the application from the WAR-file directly, without unpacking, which could mean slower execution. The autoDeploy attribute specifies whether to deploy application dropped into the webapps directory automatically.

3.3.9.1 Common Attributes

- **appBase** - The Application Base directory for this virtual host. This is the pathname of a directory that may contain web applications to be deployed on this virtual host. You may specify an absolute pathname, or a pathname that is relative to the $CATALINA_BASE directory. If not specified, the default of webapps will be used.
- **xmlBase** - The XML Base directory for this virtual host. This is the pathname of a directory that may contain context XML descriptors to be deployed on this virtual host. You may specify an absolute pathname for this directory, or a pathname that is relative to the $CATALINA_BASE directory. If not specified the default of conf will be used.
- **createDirs** - If set to true, Tomcat will attempt to create the directories defined by the attributes appBase and xmlBase during the startup phase. The default value is true. If set to true, and directory creation fails, an error message will be printed out but will not halt the startup sequence.
- **autoDeploy** - This flag value indicates if Tomcat should check periodically for new or updated web applications while Tomcat is running. If true, Tomcat periodically checks the appBase and xmlBase directories and deploys any new web applications or context XML descriptors found. Updated web applications or context XML descriptors will trigger a reload of the web application. The flag's value defaults to true.
- **backgroundProcessorDelay** - This value represents the delay in seconds between the invocation of the backgroundProcess method on this host and its child containers, including all contexts. Child containers will not be invoked if their delay value is not negative (which would mean they are using their own processing thread). Setting this to a positive value will cause a thread to be spawn. After waiting the specified amount of time, the thread will invoke the backgroundProcess method on this host and all its child containers. A host will use background processing to perform live web application deployment related tasks. If not specified, the default value for this attribute is -1, which means the host will rely on the background processing thread of its parent engine.

- **className** - Java class name of the implementation to use. This class must implement the org.apache.catalina.Host interface.
- **deployIgnore** - A regular expression defining paths to ignore when autoDeploy and deployOnStartup are set. This allows you to keep your configuration in a version control system, for example, and not deploy a .svn or CVS folder that happens to be in the appBase. This regular expression is relative to appBase. It is also anchored, meaning the match is performed against the entire file/directory name. So, foo matches only a file or directory named foo but not foo.war, foobar, or myfooapp. To match anything with "foo", you could use .**foo.**.
- **deployOnStartup** - This flag value indicates if web applications from this host should be automatically deployed when Tomcat starts. The flag's value defaults to true.
- **failCtxIfServletStartFails** - Set to true to have each child contexts fail its startup if any of its servlet that has load-on-startup >=0 fails its own startup. Each child context may override this attribute.If not specified, the default value of false is used.
- **name** - Usually the network name of this virtual host, as registered in your Domain Name Service server. Regardless of the case used to specify the host name, Tomcat will convert it to lower case internally. One of the Hosts nested within an Engine MUST have a name that matches the defaultHost setting for that Engine.
- **startStopThreads** - The number of threads this Host will use to start child Context elements in parallel. The same thread pool will be used to deploy new Contexts if automatic deployment is being used. The special value of 0 will result in the value of `Runtime.getRuntime().availableProcessors()` being used. Negative values will result in `Runtime.getRuntime().availableProcessors() + value` being used unless this is less than 1 in which case 1 thread will be used. If not specified, the default value of 1 will be used.
- **undeployOldVersion** - This flag determines if Tomcat, as part of the auto deployment process, will check for old, unused versions of web applications deployed using parallel deployment and, if any are found, remove them. This flag only applies if autoDeploy is true. If not specified the default value of false will be used.

3.3.10 Cluster

Tomcat supports server clustering. It can replicate sessions and context attributes across the clustered server. It can also deploy a WAR-file on all the cluster.

3.3.10.1 Common Attributes

- **className** - The main cluster class, currently only one is available, org.apache.catalina.ha.tcp.SimpleTcpCluster
- **channelSendOptions** - The Tribes channel send options, default is 8. This option is used to set the flag that all messages sent through the SimpleTcpCluster uses. The flag decides how the messages are sent, and is a simple logical OR.
- **channelStartOptions** - Sets the start and stop flags for the object used by the cluster. The default is Channel.DEFAULT which starts all the channel services, such as sender, receiver, multicast sender and multicast receiver.
- **heartbeatBackgroundEnabled** - Flag whether invoke channel heartbeat at container background thread. Default value is false. Enable this flag don't forget to disable the channel heartbeat thread.
- **notifyLifecycleListenerOnFailure** - Flag whether notify LifecycleListeners if all ClusterListener couldn't accept channel message. Default value is false.

3.3.11 Valve

A Valve can intercept HTTP requests before forwarding them to the applications, for pre-processing the requests. A Valve can be defined for any container, such as Engine, Host, and Context, and Cluster. In the default configuration, the AccessLogValve intercepts an HTTP request and creates a log entry in the log file, as follows:

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
       prefix="localhost_access_log." suffix=".txt"
       pattern="%h %l %u %t "%r" %s %b" />
```

3.3.11.1 Common Attributes

- **className** -Set value to org.apache.catalina.ha.tcp.ReplicationValve
- **filter** - For known file extensions or urls, you can use this Valve to notify the cluster that the session has not been modified during this request and the cluster doesn't have to probe the session managers for changes. If the request matches this filter pattern, the cluster assumes there has been no session change. An example filter would look like `filter=".*.gif|.*.js|.*.jpeg|.*.jpg|.*.png|.*.htm|.*.html|.*.css|.*.txt"`. The filter is a regular expression using `java.util.regex`.
- **primaryIndicator** - Boolean value, so to true, and the replication valve will insert a request attribute with the name defined by the primaryIndicatorName attribute. The value inserted into the request attribute is either Boolean.TRUE or Boolean.FALSE
- **primaryIndicatorName** - Default value is org.apache.catalina.ha.tcp.isPrimarySession The value defined here is the name of the request attribute that contains the boolean value if the session is primary on this server or not.
- **statistics** - Boolean value. Set to true if you want the valve to collect request statistics. Default value is false

Other valves include:

- **RemoteAddrValve**: which blocks requests from certain IP addresses
- **RemoteHostValve**: which blocks request based on hostnames
- **RequestDumperValve**: which logs details of the requests
- **SingleSignOn Valve**: when placed under a , allows single sign-on to access all the webapp under the host.

For more information about the server.xml top/nested level elements and attributes, you can check them out here.

3.4 Alternative configuration (server-<name>.xml)

3.4.1 Including the server-.xml file

Now what if we need to add modifications to the server.xml file for our application? We can't just change the server.xml file for one application as it might affect the entire initialization of all applications deployed. How can we isolate a specific change for a given application?

The Answer: **Create a server-<name>.xml**

The server-.xml is a custom file that can be included to isolate the changes needed by a specific app. All files with this format will be called after the server.xml file.

3.4.2 Replacing the server.xml with our own server-<name>.xml

This is not recommended but for the curious mind, you can always edit the catalina.bat to use your own server.xml file instead of the server.xml

```
catalina.bat start -config \conf\server-<name>.xml
```

Overall, the server.xml or your own server-<name>.xml file is the core configuration of your container. It's a way for application developers and release managers to put in resources that complements the strategy of deploying J2EE applications on the container. Other J2EE compliant containers (vendors), in one way or the other, has the same configuration file that allows custom injections and bindings, allowing them control over what services will be available when an application is deployed, redeployed or undeployed.

Chapter 4

Tomcat tomcat-users.xml configuration example

This article describes the configuration of the `tomcat-users.xml` file for Apache Tomcat 7 web server. `tomcat-users.xml` is the default user database for container-managed authentication in Tomcat.

4.1 Web Application Security Concepts

4.1.1 Authentication

To access a restricted resource on the server, Tomcat challenges a user to produce user details to confirm that they are who they say they are.

4.1.2 Authorization

Once a user is authenticated, the server determines whether this user is authorized to access the restricted resource requested.

4.1.3 Realm

A realm is a repository of user information; it is an abstraction of the data store - text file, JDBC database or a JNDI resource. This has the following information: username, password and the roles which are assigned to the users.

Both of the authentication and authorization make up the security policy of a server. Tomcat uses realms to implement container-managed security and enforce specific security policies.

4.1.4 Container Managed Security

Container managed security provides enforcing and implementing security policies on the web server.

- Also known as declarative security (for authentication and authorization)
- Defined in Java Servlet specification
- Relieves the programmer to write security related code (though they can if they want to)
- Provides consistency over multiple applications

4.2 tomcat-users.xml

Tomcat configuration files are found in the directory: CATALINA_HOME/conf (where CATALINA_HOME environment variable is the Tomcat installation directory). The main configuration file is `server.xml`. `tomcat-users.xml` is one of the configuration files.

An example of the `tomcat-users.xml` file is shown below:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="tomcat"/>
    <role rolename="role1"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
    <user username="both" password="tomcat" roles="tomcat,role1"/>
    <user username="role1" password="tomcat" roles="role1"/>
</tomcat-users>
```

`<tomcat-users>`: This is the root element. This has two nested elements: role and user.

`<role>`: Each role that a user can play is defined with a `<role>` element. The attribute `rolename` specifies the name.

`<user>`: Each user has a `<user>` entry. This has three required attributes: `username`, `password` and `roles`. Note that a user can have more than one role.

- username - Username this user must log on with.
- password - Password this user must log on with (in clear text).
- roles - Comma-delimited list of the role names associated with this user.

NOTE: For a newly installed Tomcat 7 web server, the role and user entries were commented in the `tomcat-users.xml`.

4.3 Realms

Configure Tomcat to support container managed security by connecting to an existing "database" of usernames, passwords, and user roles. This is required in case of using a web application that includes one or more `<security-constraint>` elements, and a `<login-config>` element defining how users are required to authenticate themselves.

Servlet Specification describes a portable mechanism for applications to declare their security requirements (in the `web.xml` deployment descriptor). There is no portable API defining the interface between a servlet container and the associated user and role information.

To "connect" a servlet container to some existing authentication database or mechanism that already exists in the production environment - Tomcat defines a Java interface (`org.apache.catalina.Realm`) that can be implemented by "plug in" components to establish this connection.

Six standard plug-ins are provided, supporting connections to various sources of authentication information: `JDBCRealm`, `DataSourceRealm`, `JNDIRealm`, `UserDatabaseRealm`, `MemoryRealm` and `JAASRealm`.

`UserDatabaseRealm` and `MemoryRealm` access or refer to the `tomcat-users.xml` file.

4.3.1 MemoryRealm

Accesses authentication information stored in an in-memory object collection, which is initialized from an XML document (`tomcat-users.xml`).

`MemoryRealm` is a simple demonstration implementation of the Tomcat `Realm` interface; it is not designed for production use. At start-up time, `MemoryRealm` loads information about all users, and their corresponding roles, from an XML document (by default, this document is loaded from `$CATALINA_BASE/conf/tomcat-users.xml`). Changes to the data in this file are not recognized until Tomcat is restarted.

To configure MemoryRealm, create a <Realm> element and nest it in $CATALINA_BASE/conf/server.xml file. The <Realm> element can be nested inside any one of the following Container elements: Engine (this realm will be shared across all web applications on all virtual hosts), Host (this realm will be shared across all web applications for this virtual host), or Context (this realm will be used only for this web application).

```
<Realm className="org.apache.catalina.realm.MemoryRealm" />
```

- className attribute: This is a required attribute. This is the Java class name of the implementation to use. This class must implement the org.apache.catalina.Realm interface.
- pathname attribute: If no path-name is specified, the default value is CATALINA_HOME/conf/tomcat-users.xml. Absolute or relative (to $CATALINA_BASE) path-name to the XML file containing our user information.

NOTE: The CATALINA_BASE environment variable specifies location of the root directory of the "active configuration" of Tomcat. It is optional to define this variable. It defaults to be equal to CATALINA_HOME.

4.3.2 UserDatabaseRealm

Accesses authentication information stored in an UserDatabase JNDI resource, which is typically backed by an XML document (tomcat-users.xml).

UserDatabaseRealm is an implementation of the Tomcat Realm interface that uses a JNDI resource to store user information. By default, the JNDI resource is backed by an XML file. It is not designed for large-scale production use. At start-up time, the UserDatabaseRealm loads information about all users, and their corresponding roles, from an XML document (by default, this document is loaded from $CATALINA_BASE/conf/tomcat-users.xml). The users, their passwords and their roles may all be editing dynamically; Tomcat provides MBeans that may be accessed via JMX for this purpose. Changes may be saved and will be reflected in the XML file.

To configure UserDatabaseRealm, create a <Realm> element and nest it in your $CATALINA_BASE/conf/server.xml file.

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase"/ ↩
    >
```

resourceName attribute: The name of the global UserDatabase resource that this realm will use for user, password and role information. This attribute value is also defined as follows in server.xml:

```
<Resource name="UserDatabase" auth="Container"
        type="org.apache.catalina.UserDatabase"
        description="User database that can be updated and saved"
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/tomcat-users.xml">
</Resource>
```

NOTE: The name and location of the tomcat-users.xml file can be changed.

4.4 The Manager Application

Tomcat manager application is provided as part of the Tomcat distribution and is stored in the CATALINA_HOME/webapps/manager directory by default. It's a special web application that allows manage other web applications while the Tomcat server is running. One can, for example, deploy, undeploy, start, and stop web applications on the server using this tool.

By default, access to the manager application is disabled; this can be accessed only by an authenticated user. The default realm for the manager application is tomcat-users.xml.

To set up the manager application, add a user with the manager role to this file. The role manager names can be found in the web.xml file of the Manager web application. One of the available roles is manager-gui - provides access to the HTML interface. For example, add the manager role and then alter an existing user (such as tomcat), as follows:

```
<role rolename="manager-gui"/>
<user username="tomcat" password="tomcat" roles="tomcat, manager-gui"/>
```

Access the manager application by one of the ways:

- https://localhost:8080/ and click the "Manager App" button
- https://localhost:8080/manager/html

This will prompt for the user name and password. Enter the values from the `tomcat-users.xml`.

Chapter 5

Tomcat web.xml Configuration Example

5.1 Introduction

In this example we will talk about Apache Tomcat Servlet/JSP container also referred to as Tomcat server and the `web.xml` file that provides different options to configure Tomcat. Apache Tomcat is developed as an open source software and is available under Apache License version 2.

Apache Tomcat is a light server that is aimed at performance rather than extensive functionality coverage, and therefore it is used by many large-scale web applications that require speed and ease of configuration and deployment. Tomcat server is not only popular because of its performance, but also because of its ease of set-up and configuration. Tomcat supports server and web application level configuration. Such flexibility allows fine tuning of the server performance, and security. The main purpose of using Tomcat is as a platform for developing and deploying web applications and web services.

5.2 Background

There are many versions of the Tomcat server. In this example we will talk about Apache Tomcat Servlet/JSP container version 8.0.20 that implements the Servlet 3.1 and JavaServer Pages 2.3 specifications. In this example 64-bit Windows 7 Professional SP1 was used. JDK 7 for Windows 7 64-bit was installed and configured.

Note that in this article we use Windows specific directory path separator `\`, which is different from the one used on Linux like systems. Below are critical directories for the Tomcat:

- `TOMCAT_ROOT_DIR` (known as `$CATALINA_HOME`, where catalina is the project name of the Tomcat server) is the directory where you have placed Apache Tomcat folder, e.g. `C:\apache-tomcat-8.0.20-windows-x64\apache-tomcat-8.0.20` in our example. In addition, Tomcat server can be configured for multiple instances by defining `$CATALINA_BASE` for each instance of the Tomcat server. By default both folders refer to the single Tomcat server installation.
- `TOMCAT_ROOT_DIR\bin` is the directory, where server startup, shutdown, and other scripts are located.
- `TOMCAT_ROOT_DIR\conf` is the directory that contains configuration files and related to those files DTDs (Document Type Definition). The files in this folder are used for the server level performance tuning, security, load balancing, etc. We will talk about `web.xml` configuration file located in this folder.
- `TOMCAT_ROOT_DIR\logs` is the directory, where the log files are stored by default.
- `TOMCAT_ROOT_DIR\webapps` is the directory, where the webapps you place in the server are stored.

All Tomcat server related configurations from above folders are read at the server's start-up. Thus, if you change any of the Tomcat server's configuration files, you need to restart the server. Also note that the configurations discussed in this article are mainly applicable to Tomcat 8, but may not be applicable to earlier or later versions. Please refer to the official documentation here for the version you want to install.

Now let's look at `web.xml` deployment descriptor configuration file, which is one of the main configuration files for the Tomcat server. It is located in the `TOMCAT_ROOT_DIR\conf` folder. This configuration file is used for basic web application's configuration shared by all web applications that will be deployed on the Tomcat server instance. Each of the web application may optionally override shared configurations by defining their own `web.xml` file located in `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\WEB-INF` folder. Note that web application specific configurations should NEVER be placed in the shared `web.xml`. Describing options for the per web application configuration file is outside of the scope of this article, but you can read more about it here or here.

From now on, whenever we say `web.xml`, we refer to the Tomcat server's `web.xml` located in the `TOMCAT_ROOT_DIR\conf` and shared by all web applications deployed on that server. Note that the ordering of the configuration elements in the server's `web.xml` must be followed as specified by the Java Servlet 3.1. Read more about Java Servlet 3.1 here.

5.3 Tomcat Server Installation

In this example we will install Tomcat 8 on the Windows 7 Professional 64-bit system, which has JDK 7 installed. If you need a refresher on how to set up JDK 7 on Windows 7 please refer here.

Once you checked that you have JDK 7 setup, get Tomcat server distribution package from the Apache Tomcat's official website. As of writing of this article the latest version of Tomcat server is Tomcat 8 and it is available from here. The Tomcat distribution used in this example is a zip of the binary core for 64-bit Windows.

We will not discuss in details installation steps for Tomcat, but you can refer to the Apache Tomcat server's official installation howto here on how to setup the Tomcat 8.

Assuming you have installed Tomcat, let's look at how to verify if it is working.

To run Tomcat server on Windows you need to configure it as a Windows service. This needs to be done each time you restart your computer. To manually start Tomcat you need to open Windows's command line prompt by searching for cmd in the start-up menu. Then, in the command line prompt navigate to `TOMCAT_ROOT_DIR\bin` folder and run **"service.bat"** script with **"install"** option. Below is the screen shot of running service.bat install.

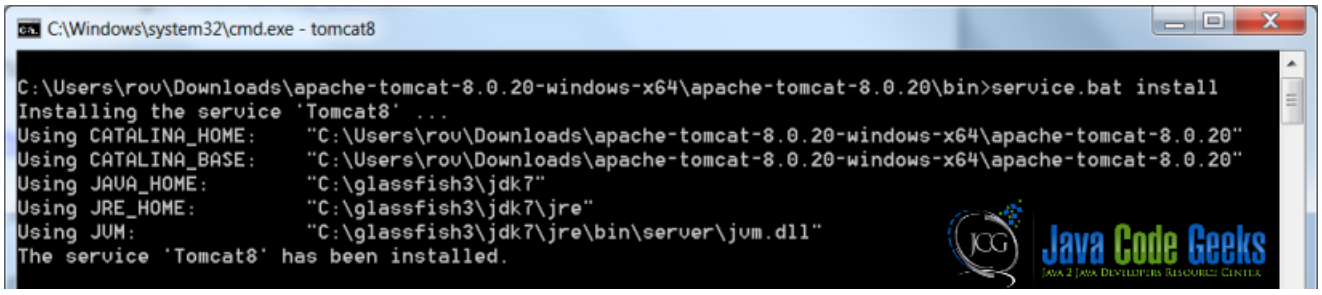

Figure 5.1: setting Tomcat

Note: you must have administrative privileges to run Tomcat server from the command line prompt. To verify that the Tomcat 8 is running on the Windows type tomcat8 command into the open command line prompt. It will open a new command line window with Tomcat startup information as shown in the picture below.

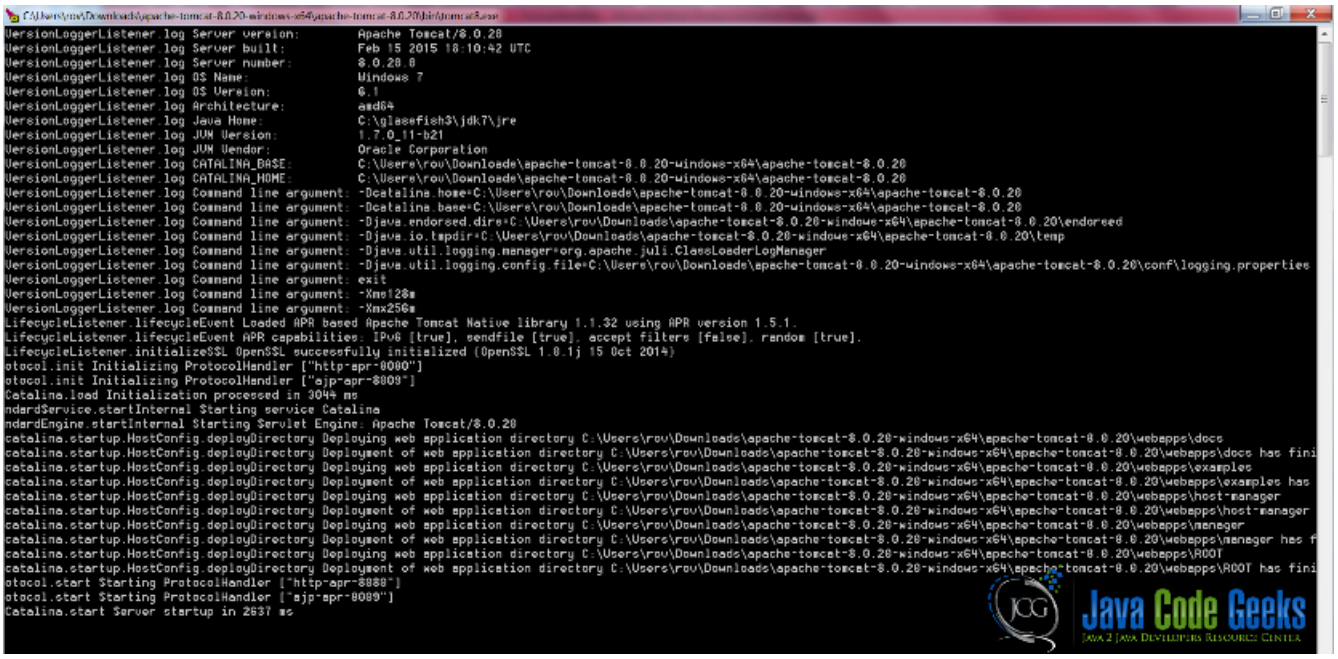

Figure 5.2: Tomcat start up text

Freshly installed Tomcat has some default web application in the `TOMCAT_ROOT_DIR\webapps` directory. We will not discuss those web applications in depth here, but you can read about them here and here.

5.4 Using Tomcat Server Management App

We can verify that the Tomcat server was installed and configured correctly in several ways. One of the ways is by using manager web application provided by default with Tomcat. To access the manager web app several steps need to be performed. First, verify that the Tomcat was registered with Windows as a service as explained in previous section. Then, add a user and a role elements in the `TOMCAT_ROOT_DIR\config\tomcat-users.xml` configuration file as shown below:

```
<tomcat-users xmlns=https://tomcat.apache.org/xml xmlns:xsi=https://www.w3.org/2001/ ↩
    XMLSchema-instance xsi:schemaLocation=https://tomcat.apache.org/xml tomcat-users.xsd  ↩
    version="1.0">
  <role rolename="manager-gui"/>
  <user username="tomcat" password="tomcat" roles="manager-gui"/>
</tomcat-users>
```

The above entry in the `tomcat-users.xml` allows access to the manager web app provided by default with each Tomcat instance. The user name and the password in the example could be any legal value. Once you have added above entry in the `tomcat-users.xml` configuration file, restart the Tomcat server by closing the status command line prompt, and starting it again with tomcat8 command as explained earlier.

Read more on how to configure and run manager web application here.

Once you setup the user name and password and started Tomcat service you can open the following URL in the web-browser of your choice `https://localhost:8080/manager/status`. The screenshot of the web page is below:

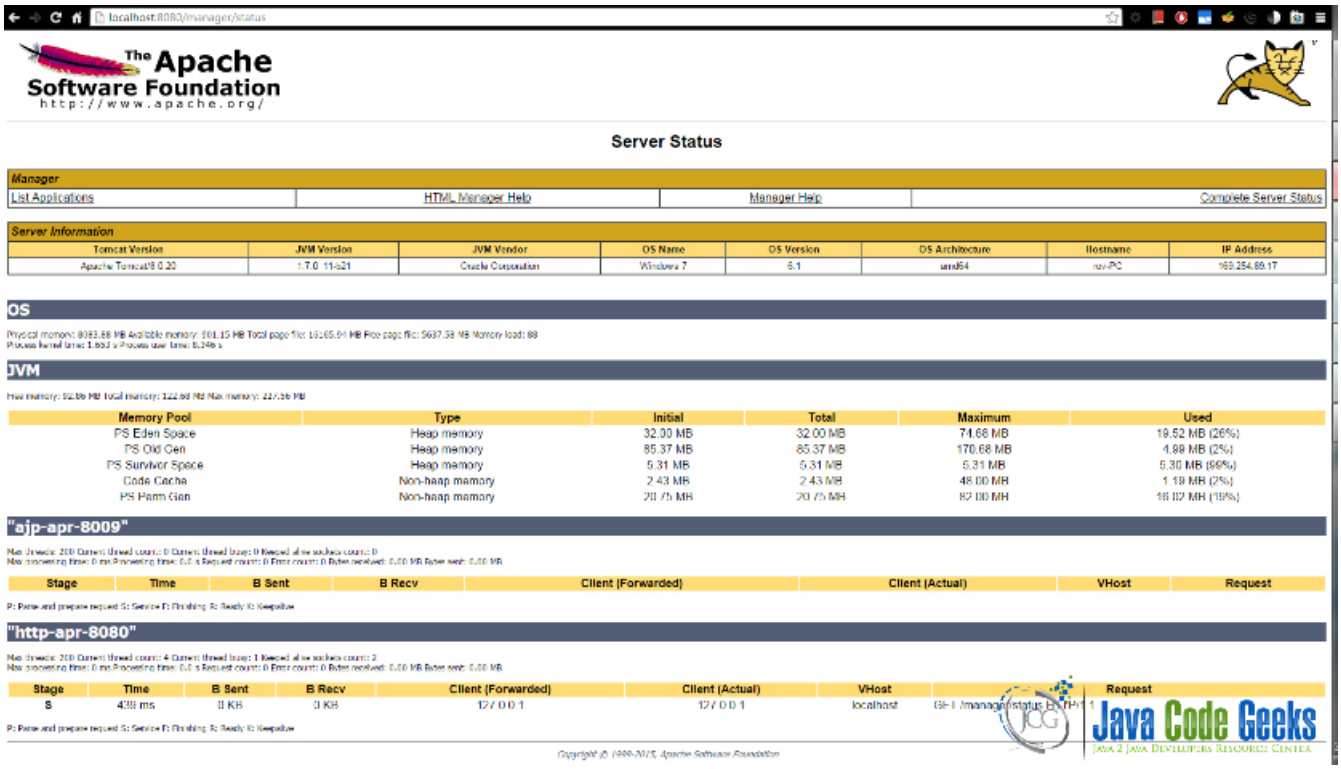

Figure 5.3: Tomcat status web

The screenshot above was taken from the web page opened in Google Chrome browser version 41.0.X. The above URL contains basic information on Tomcat's version, OS it runs on, and JVM properties. Note in the screenshot that the initial maximum heap-memory size is relatively small. It is generally recommended to increase that limit to a higher value depending on your system capabilities and expected requirements of the web applications that will be hosted. Read more on Tomcat's heap-memory settings here.

5.5 Tomcat server's web.xml configuration file

Up to this point, we have registered Tomcat as a Windows service, and now we are ready to look at `web.xml` configuration file located in `TOMCAT_ROOT_DIR\conf` folder. Below is the `web.xml` with no options:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="https://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://xmlns.jcp.org/xml/ns/javaee
  https://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
version="3.1">
  ...
</web-app>
```

The above simplified `web.xml` contains `<web-app>` element which will contain options for operating all web applications deployed in this Tomcat instance. The `<web-app>` element represents the configuration options for a web application. It is required that all deployment descriptors must confirm to XML schema for the Servlet 3.1. `Web-app` is the root element for the deployment descriptor, `web.xml` file.

5.5.1 Configuring Shared DataSource configuration using JNDI DataSource

In this section, configuration of the shared data source across all web applications is described. Note that per web application data source configuration is also possible and is a common choice for small projects. In this section, we will look at adding MySQL data source. Note that you are expected to be familiar with Context, context configured per web application running within a Tomcat server, and Host, host configured per Tomcat instance in the `TOMCAT_ROOT_DIR \conf\server.xml` file, configurations.

Get the `MySQL JDBC` and add it to the library of the shared resource located in the `TOMCAT_ROOT_DIR\lib` folder. In our case we copy `Connector/J 3.0.11-stable` (the official JDBC Driver) jar into the `TOMCAT_ROOT_DIR\lib` folder.

Perform MySQL server configuration as described in here section "1. MySQL configuration".

Next perform Context configurations as described in here section "2. Context configuration".

Next configure `web.xml` as shown below:

```
<resource-ref>
 <description>MySQL Datasource example</description>
 <res-ref-name>jdbc/TestDB</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
</resource-ref>
```

Note that it is expected that resource with name `jdbc/TestDB` was configured in the Context. `<resource-ref>` element represents the resource manager connection factory reference, such as JDBC DataSource. The sub elements of the `<resource-ref>` represent the following:

- `<description>` is a description of this resource reference
- `<res-ref-name>` is a JNDI name and must be unique within deployment environment, e.g. web application.
- `<res-type>` is the type of the data source. This type is expected to be implemented by the data source. For the example data source `javax.sql.DataSource` read more here
- `<res-auth>` specifies if the deployment component code signs on programmatically to the resource manager, or if the container will sign on to the resource manager for the deployment component. In the latter, the container needs information to sign on from the web application.

For more extended list for different database configurations refer here.

5.5.2 Configuring Jasper 2 Java Server Pages (JSP) Engine

Tomcat 8.0 implements JavaServer Pages 2.3 specification with Jasper 2 JSP engine. This version of the JSP engine was re-designed for better performance compared to the previous version. An extended list of parameters and their purpose could be found here.

```
<servlet>
        <servlet-name>jsp</servlet-name>
        <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
        <init-param>
                <param-name>fork</param-name>
                <param-value>false</param-value>
        </init-param>
        <init-param>
                <param-name>xpoweredBy</param-name>
                <param-value>false</param-value>
        </init-param>
        <load-on-startup>3</load-on-startup>
</servlet>
...
```

```
<servlet-mapping>
        <servlet-name>jsp</servlet-name>
        <url-pattern>*.jsp</url-pattern>
        <url-pattern>*.jspx</url-pattern>
</servlet-mapping>
```

The above `<servlet>` element is a small Java program that runs with a Tomcat server. A servlet receives and responds requests from the Web clients. The JSP compiler servlet is the mechanism used by Tomcat server to support JSP pages. Below is explanation for sub elements:

- `<servlet-name>` must be unique within web application
- `<servlet-class>` read more on `org.apache.jasper.servlet.JspServlet` here
- `Fork=false` option tells Ant not to fork compiles of JSP pages, but to compile them on the JVM that Tomcat is running on (default is true)
- `XpoweredBy=false` option disables adding X-Powered-By response header by generated servlet (default is false)
- `<load-on-startup>` with value 3 means this servlet will be loaded after servlets with a lower value
- `<url-pattern>` is the pattern for which any matched URL request will be serviced by this servlet

JSP servlet's options may be used for optimization of Tomcat server request processing. One of such optimizations could be settings development option to false, which will disable on access checks for JSP page's compilation. To see the full list of options go here.

5.5.3 Configuring SSI (Server Side Includes)

SSI are directives that are placed within HTML tags and evaluated on the server during page processing. SSI allows you adding dynamically generated content to the HTML pages without the need to serve the whole HTML page. SSI support on Tomcat is available as both a servlet and a filter. Only one of two SSI types should be used.

By default SSI support is disabled on Tomcat. You can read more about this `web.xml` option here. Note that SSI directives could be used to execute external to Tomcat JVM programs, which may result in the security vulnerability to your Tomcat instance.

To use SSI look for SSI filter or SSI servlet in the default `web.xml` in `TOMCAT_ROOT_DIR\conf\web.xml`. Read comments and uncomment appropriate sections of the configuration file to enable SSI.

5.5.4 Configuring CGI (Common Gateway Interface)

CGI defines a way for a web server to interact with external content-generating programs that conform to the CGI spec requirements.

By default SSI support is disabled on Tomcat. You can read more about this `web.xml` option here. Note that CGI scripts could be used to execute external to Tomcat JVM programs, which may result in the security vulnerability to your Tomcat instance.

To use CGI look for CGI servlet and servlet-mapping elements in the default `web.xml` in `TOMCAT_ROOT_DIR\conf\web.xml`. Read comments and uncomment appropriate sections of the configuration file to enable CGI servlet.

5.5.5 Configuring Default Servlet

Default servlet serves static resources and directory listing (if directory listing is enabled). It is declared in `TOMCAT_ROOT_DIR\conf\web.xml` and it is defined on all Tomcat instances. Below is an `web.xml` entry for the Default Servlet:

```
<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>
```

A description of each of the sub elements in the build in default `<servlet>` element is below:

- `<servlet-name>` is the canonical name of this servlet. The name must be unique within the web application.
- `<servlet-class>` is the fully qualified class name of this servlet
- `<init-param>` is the optional parameter for this servlet that contains a name value pair that is used when the servlet is initialized. This element contains two sub elements `<param-name>` and `<param-value>`, which represent parameter's name and value
- `<load-on-startup>` is the indicator of the load order for this servlet. If the value is a lowest positive integer, the servlet loaded first. To load the servlet at any time the value should be set to a negative integer or should be leaved undefined

In the second part of the `web.xml`, we define `<servlet-mapping>` element which is a servlet mapping for the build in default servlet. Note that each servlet defined in the `web.xml` must be mapped as in the default servlet example either in the server's `web.xml` or web application's `web.xml`. A description of each of the sub elements in the `<servlet-mapping>` element is below:

- `<servlet-name>` a canonical name of the servlet. Note the `<servlet-name>` sub element's value in the `<servlet-mapping>` must match the value specified in the `<servlet>` element's similar sub element.
- `<url-pattern>` the url pattern which will be used to map all requests with matching patterns to the servlet

As you can see from configuration of the default servlet the default servlet is loaded at Tomcat server startup, directory listing is disabled, and debugging is disabled. To see extended list of options that you can modify go here. There you can find the available parameters descriptions and settings.

Most of the Tomcat server related security configurations are set in the Default Servlet element in the `TOMCAT_ROOT_DIR\conf\web.xml`. There are three main attributes of this element that need to be mentioned:

- `readonly` configuration should be set to true to prevent clients from deleting or modifying static resources on the server, and prevent clients from uploading new resources to the server.
- `listing` configuration should be set to false to prevent directory listing which if misused could use significant CPU and lead to DOS (Denial of Service) attacks.
- `showServerInfo` should be set to false (true by default). If directory listing is enabled Tomcat version number will be included in the info which is undesired.

For additional security it is recommended to have `web.xml` for each individual web application with appropriate directory listing settings. On security settings read more here.

5.5.6 Default Session Configuration

You can set session timeout for all web application in the `web.xml`. By default the session timeout is 30 minutes as shown below:

```
<session-config>
        <session-timeout>30</session-timeout>
</session-config>
```

To change the value for when the session should be deleted change the `<session-timeout>` value.

5.5.7 Default MIME Type Mappings

You can configure mapping for the data types that could be processed by the web applications in this Tomcat instance. There are many MIME types set by default in the `web.xml` for all web applications. Tomcat will generate a header entry for the content type automatically based on the static resource's file name extension using the MIMTE types specified here. Below some of the MIME types from the long list:

```
<mime-mapping>
        <extension>gif</extension>
        <mime-type>image/gif</mime-type>
</mime-mapping>
<mime-mapping>
        <extension>java</extension>
        <mime-type>text/x-java-source</mime-type>
</mime-mapping>
<mime-mapping>
        <extension>wav</extension>
        <mime-type>audio/x-wav</mime-type>
</mime-mapping>
<mime-mapping>
        <extension>xhtml</extension>
        <mime-type>application/xhtml+xml</mime-type>
</mime-mapping>
```

Each <mime-mapping> element represents mapping between resource type and its `Content-Type` header value. `<extension>` sub element stands for the resource's extension, and `<mime-type>` stands for the mime type that the file extension will map to.

5.5.8 Default Welcome File List

When a request is made to a directory the Default Servlet tries to find the `welcome file` in that directory. This listing is recommended to be overwritten in the per web application deployment configuration file (`web.xml`). If no `welcome file` found Tomcat returns 404.

```
<welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

`<welcome-file-list>` element contains default welcome files to be retrieved when any web application is accessed. Each `<welcome-file>` in the list is attempted to be accessed from top to bottom. If none are present in the folder being requested a 404 status is returned.

5.5.9 Configuring Clustering/Session Replication

For enabling clustering several steps need to be performed, and one of them is to add `<distributable/>` element in the `TOMCAT_ROOT_DIR/conf/web.xml`.

In addition other than `web.xml` server configuration files need to be modified as explained in here. Note that the request session gets distributed in the cluster to each web application that has `<distributable/>` element in it. It is recommended that all Tomcat instances configured the same.

5.6 Conclusion

The above options are some of the main Tomcat server's configuration options that are shared by all web applications deployed on the server. There are more `web.xml` options that could be set up on the Tomcat server that are not discussed here. For the complete list of the possible Tomcat server's `web.xml` configuration file options and their description refer to the Java Servlet 3.1 specifications here. For more extended explanation of the Apache Tomcat server and related to infrastructure please refer here and here.

Chapter 6

Tomcat context.xml Configuration Example

In Tomcat, the Context Container represents a single web application running within a given instance of Tomcat. A web site is made up of one or more Contexts. For each explicitly configured web application, there should be one context element either in server.xml or in a separate context XML fragment file.

Once a Context has been defined, Catalina will attempt to match incoming HTTP requests to its context path. There is no limit to the number of Contexts that can be defined, as long as each Context is given its own unique context path. The context path, which is contained within the context, specifies where the application's resources can be located. Applications can either be stored in a Web Application Archive (WAR) file, in which case they will be dynamically uncompressed as needed, or as organized unpacked resources in a directory.

Don't do the same Context configuration work twice. Tomcat lets you save complete server configuration profiles, and apply them to new instances with a single click.

Once Catalina has matched a context with a request, the selected Context passes the request to the correct servlet to process the request, based on definitions contained in the web application deployment descriptor file.

Here is an example of context.xml

context.xml

```
<?xml version='1.0' encoding='utf-8'?>

<!--  ++++++++++++++++++++++++++++++++++++++++++++++

This file is one of a suite of configuration files ...
  context.xml
  web.xml
  tomcat_user.xml
  server.xml
that we set up on our "deploying Apache httpd and Tomcat" course to
configure Tomcat and web applications that are running within it.

++++++++++++++++++++++++++++++++++++++++++++++++++  -->

<!-- The contents of this file will be loaded for each web application -->

<Context reloadable="true" privileged="true">

<!-- **** Note - we have added in the reloadable and privileged attributes
to enable the invoker servlet and cgi support (other changes needed in
web.xml too for that, though **** -->

    <!-- Default set of monitored resources -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>

    <!-- Disable session persistence across Tomcat restarts -->
```

```
    <Manager pathname="" />

    <!-- Enable Comet connection tracking (provides events
         on session expiration as well as webapp lifecycle) -->

    <Valve className="org.apache.catalina.valves.CometConnectionManagerValve" />
</Context>
```

- **Context reloadable = "true"** and **privileged = "true"** - to enable reloading and invoker servlet and cgi support.
- **WatchedResource = "web.xml"** - we are setting this context.xml to monitor the changes made to the web.xml of the application being deployed.
- **Manager** - specifying this will disable session persistence across tomcat restarts
- **Valve** - specifies the connection tracker for the web application lifecycle.

The Default Context elements may be defined for multiple web applications and these can be configured individually. The application will override all that's defined in this file.

- In the $CATALINA_BASE/conf/context.xml file: the Context element information will be loaded by all web applications.
- In the $CATALINA_BASE/conf/[enginename]/[hostname]/context.xml.default file: the Context element information will be loaded by all web applications of that host.

6.1 Common Attributes of context.xml

allowCasualMultipartParsing Set to true if Tomcat should automatically parse multipart/form-data request bodies when `HttpServletRequest.getPart*` or `HttpServletRequest.getParameter*` is called, even when the target servlet isn't marked with the @MultipartConfig annotation (See Servlet Specification 3.0, Section 3.2 for details). Note that any setting other than false causes Tomcat to behave in a way that is not technically spec-compliant. The default is false

backgroundProcessorDelay This value represents the delay in seconds between the invocation of the backgroundProcess method on this context and its child containers, including all wrappers. Child containers will not be invoked if their delay value is not negative (which would mean they are using their own processing thread). Setting this to a positive value will cause a thread to be spawn. After waiting the specified amount of time, the thread will invoke the backgroundProcess method on this host and all its child containers. A context will use background processing to perform session expiration and class monitoring for reloading. If not specified, the default value for this attribute is -1, which means the context will rely on the background processing thread of its parent host.

className Java class name of the implementation to use. This class must implement the org.apache.catalina.Context interface. If not specified, the standard value (defined below) will be used.

containerSciFilter The regular expression that specifies which container provided SCIs should be filtered out and not used for this context. Matching uses java.util.regex.Matcher.find() so the regular expression only has to match a sub-string of the fully qualified class name of the container provided SCI for it to be filtered out. If not specified, no filtering will be applied.

cookies Set to true if you want cookies to be used for session identifier communication if supported by the client (this is the default). Set to false if you want to disable the use of cookies for session identifier communication, and rely only on URL rewriting by the application.

crossContext Set to true if you want calls within this application to ServletContext.getContext() to successfully return a request dispatcher for other web applications running on this virtual host. Set to false (the default) in security conscious environments, to make getContext() always return null.

docBase The Document Base (also known as the Context Root) directory for this web application, or the pathname to the web application archive file (if this web application is being executed directly from the WAR file). You may specify an absolute pathname for this directory or WAR file, or a pathname that is relative to the appBase directory of the owning Host.

failCtxIfServletStartFails Set to true to have the context fail its startup if any servlet that has load-on-startup >=0 fails its own startup.

fireRequestListenersOnForwards Set to true to fire any configured ServletRequestListeners when Tomcat forwards a request. This is primarily of use to users of CDI frameworks that use ServletRequestListeners to configure the necessary environment for a request. If not specified, the default value of false is used.

logEffectiveWebXml Set to true if you want the effective web.xml used for a web application to be logged (at INFO level) when the application starts. The effective web.xml is the result of combining the application's web.xml with any defaults configured by Tomcat and any web-fragment.xml files and annotations discovered. If not specified, the default value of false is used.

override Set to true to ignore any settings in both the global or Host default contexts. By default, settings from a default context will be used but may be overridden by a setting the same attribute explicitly for the Context.

path The context path of this web application, which is matched against the beginning of each request URI to select the appropriate web application for processing. All of the context paths within a particular Host must be unique. If you specify a context path of an empty string (""), you are defining the default web application for this Host, which will process all requests not assigned to other Contexts.

preemptiveAuthentication When set to true and the user presents credentials for a resource that is not protected by a security constraint, if the authenticator supports preemptive authentication (the standard authenticators provided with Tomcat do) then the user' credentials will be processed. If not specified, the default of falseis used.

privileged Set to true to allow this context to use container servlets, like the manager servlet. Use of the privileged attribute will change the context's parent class loader to be the Server class loader rather than the Shared class loader. Note that in a default installation, the Common class loader is used for both the Server and the Shared class loaders.

reloadable Set to true if you want Catalina to monitor classes in /WEB-INF/classes/ and /WEB-INF/lib for changes, and automatically reload the web application if a change is detected. This feature is very useful during application development, but it requires significant runtime overhead and is not recommended for use on deployed production applications. That's why the default setting for this attribute is false. You can use the Manager web application, however, to trigger reloads of deployed applications on demand.

resourceOnlyServlets Comma separated list of Servlet names (as used in /WEB-INF/web.xml) that expect a resource to be present. Ensures that welcome files associated with Servlets that expect a resource to be present (such as the JSP Servlet) are not used when there is no resource present. This prevents issues caused by the clarification of welcome file mapping in section 10.10 of the Servlet 3.0 specification. If the org.apache.catalina.STRICT_SERVLET_COMPLIANCE system property is set to true, the default value of this attribute will be the empty string, else the default value will be jsp.

sendRedirectBody If true, redirect responses will include a short response body that includes details of the redirect as recommended by RFC 2616. This is disabled by default since including a response body may cause problems for some application component such as compression filters.

sessionCookieDomain The domain to be used for all session cookies created for this context. If set, this overrides any domain set by the web application. If not set, the value specified by the web application, if any, will be used.

sessionCookieName The name to be used for all session cookies created for this context. If set, this overrides any name set by the web application. If not set, the value specified by the web application, if any, will be used, or the name JSESSIONID if the web application does not explicitly set one.

sessionCookiePath The path to be used for all session cookies created for this context. If set, this overrides any path set by the web application. If not set, the value specified by the web application will be used, or the context path used if the web application does not explicitly set one. To configure all web application to use an empty path (this can be useful for portlet specification implementations) set this attribute to / in the global CATALINA_BASE/conf/context.xml file.

sessionCookiePathUsesTrailingSlash Some browsers, such as IE, will send a session cookie for a context with a path of /foo with a request to /foobar. To prevent this, Tomcat will add a trailing slash to the path associated with the session cookie so, in the above example, the cookie path becomes /foo/. However, with a cookie path of /foo/, IE will no longer send the cookie with a request to /foo. This should not be a problem unless there is a servlet mapped to /*. In this case this feature will need to be disabled. The default value for this attribute is true. To disable this feature, set the attribute to false.

swallowAbortedUploads Set to false if Tomcat should not read any additional request body data for aborted uploads and instead abort the client connection. This setting is used in the following situations: - the size of the request body is larger than the maxPostSize configured in the connector - the size limit of a MultiPart upload is reached - the servlet sets the response status to 413

(Request Entity Too Large) - Not reading the additional data will free the request processing thread more quickly. Unfortunately most HTTP clients will not read the response if they can not write the full request. The default is true, so additional data will be read.

swallowOutput If the value of this flag is true, the bytes output to System.out and System.err by the web application will be redirected to the web application logger. If not specified, the default value of the flag is false. **tldValidation** If the value of this flag is true, the TLD files will be XML validated on context startup. If the org.apache.catalina.STRICT_SERVLET_COMPLIANCE system property is set to true, the default value of this attribute will be true, else the default value will be false. Setting this attribute to true will incur a performance penalty.

useHttpOnly Should the HttpOnly flag be set on session cookies to prevent client side script from accessing the session ID? Defaults to true.

wrapperClass Java class name of the org.apache.catalina.Wrapper implementation class that will be used for servlets managed by this Context. If not specified, a standard default value will be used.

xmlBlockExternal If the value of this flag is true, the parsing of web.xml, web-fragment.xml, *.tld, *.jspx, *.tagx and tagPlugins.xml files for this web application will not permit external entities to be loaded. If not specified, the default value of true will be used.

xmlNamespaceAware If the value of this flag is true, the parsing of web.xml and web-fragment.xml files for this web application will be namespace-aware. Note that *.tld, *.jspx and *.tagx files are always parsed using a namespace-aware parser and that the tagPlugins.xml file (if any) is never parsed using a namespace-aware parser. Note also that if you turn this flag on, you should probably also turn xmlValidation on. If the org.apache.catalina.STRICT_SERVLET_COMPLIANCE system property is set to true, the default value of this attribute will be true, else the default value will be false. Setting this attribute to true will incur a performance penalty.

xmlValidation If the value of this flag is true, the parsing of web.xml and web-fragment.xml files for this web application will use a validating parser. If the org.apache.catalina.STRICT_SERVLET_COMPLIANCE system property is set to true, the default value of this attribute will be true, else the default value will be false. Setting this attribute to true will incur a performance penalty.

In summary, when an HTTP request is made, Catalina receives it, and passes it to the appropriate Context, which in turn passes the request to the appropriate servlet to serve it.

In the event that the request does not match any specific context paths, Catalina passes the request to a Context whose context path is a zero-length string. This Context is required for Tomcat to run properly, because it is considered the "default" web application, responsible for processing requests that do not match any other specific context paths.

For further reference: context.xml documentation in the Apache Tomcat website

Chapter 7

Tomcat clustering and session replication tutorial

7.1 Introduction

In this example we will discuss Apache Tomcat Servlet/JSP container's clustering and session replication related configurations. Apache Tomcat server is lightweight server that is relatively easy to set-up. It has many features that enable high flexibility. Tomcat can be fine-tuned to do well for some traffic, but if your traffic is expected to be large you may need to distribute the traffic over several Tomcat instances that we will refer to as workers sometimes. Thus you may want to set-up a cluster of Tomcat instances.

Having cluster enables you not only to distribute traffic to your web application, it also allows you to provide high availability (explained later) of your web application. To achieve these functionalities you will also need a load balancing server, such as Apache Httpd Web Server. Once you have all these in place you can add session replication to prevent a Tomcat server failure from affecting user's experience.

There are many possible ways to do clustering and session replication. Some of the most popular schemas for clustering and session replication are when all servers on the same machine or all servers on different machines. There is also options for orchestrator server that distributes load between workers within a cluster. Some of the possible load balancing capable servers are `Apache Httpd Web Server` and `Nginx`.

To have session replication across several Tomcat servers you need a front end server that accepts all connections, and then partitions connections to workers in the cluster.

In addition to load balancing server, you need to have session replication manager that is responsible for copying session and maintaining copies "up to date". Tomcat allows two types of the cluster-aware session manager. First one is DeltaManager which might be slower because it creates more session copies, but it is more reliable in case of several worker failures. The second one is BackUpManager that creates only one copy of a session on some other server, thus if one fails another takes over. Both have advantages and disadvantages. In our example we will use DeltaManager.

7.2 Environment

In this example 64-bit Windows 7 Professional SP1 was used. JDK 7 for Windows 7 64-bit was installed and configured.

A sample web application was developed on Eclipse Java EE IDE" Version Kepler Service Release 1 based on this example and was extended to enable and demonstrate clustering and session replication.

For a single Tomcat 8 server instance configuration example please refer here. We will extend simple Tomcat configuration and enable it for clustering and session replication, and we will discuss how to do that later. Note you do not have to go through that example, we will do most configuration related discussion we need here, but you can use it as a reference for some steps.

For load balancing Apache HTTP Server 2.4.12 (httpd) with mod_jk module was used. We will have three Tomcat 8 instances in our server, and they will be deployed on the same physical machine (Windows 7). Since all cluster servers are on the same machine Tomcat's build in in-memory session replication will be used.

The above environment and technologies were chosen to demonstrate common Tomcat clustering scheme using latest versions of required technologies. For example `mod_jk` is one of the recommended modes by Apache for load balancing, and is commonly used in production. In addition `mod_jk` is more mature and rich in feature, than other load balancing modules available for Apache Web Server. Also note that all technologies discussed here are free and most of them are open source projects.

Note that in this example we use Windows specific directory path separator `\`, which is different from the one used on Linux like systems. Below are critical directories for the Tomcat that will be referenced to in this example:

`TOMCAT_ROOT_DIR` (known as $CATALINA_HOME, where Catalina is the project name of the Tomcat server) is the directory where you have placed Apache Tomcat folder, e.g. C:\apache-tomcat-8.0.20-windows-x64\apache-tomcat-8.0.20 in our example. In addition, Tomcat server can be configured for multiple instances by defining $CATALINA_BASE for each instance of the Tomcat server. By default both folders refer to the single Tomcat server installation.

`TOMCAT_ROOT_DIR\conf` is the directory that contains configuration files and related to those files DTDs (Document Type Definition). The files in this folder are used for the server level performance tuning, security, load balancing, etc. We will talk about web.xml configuration file located in this folder.

`TOMCAT_ROOT_DIR\webapps` is the directory, where the webapps you place in the server are stored.

`TOMCAT_ROOT_DIR\webapps\PROJECT_DIR` is the directory, where a specific webapp is placed, when deployed on the server.

`TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\WEB-INF` is the directory, where the webapp's specific configuration files are placed. Those configuration files override container's configuration files.

`TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\META-INF` is the directory that holds web application specific context.xml configuration file.

7.3 Motivation and Notations

Let's first look at some terms that will be mentioned in this example, and that will help you understand better the purpose achieved by clustering with session replication.

- `Scalability` is related to server's ability to handle efficiently many concurrent requests simultaneously. In clustering this achieved by spreading work among servers to have as fast as possible request processing.
- `Load Balancing` is a technology aimed at distributing request load among a collection of servers.
- `Load Balancer` is the server that performs load balancing duties by distributing requests among servers on the cluster. Normally the goal of the load balancer is to distribute the work as evenly as possible among available servers. Load balancer (in our case Apache Httpd Web Server) must keep track of the servers on the cluster, and perform a failover protocol if needed. Load Balancer with Session Replication provide high availability of a web application, which means if one server fails, another server will take over. Thus the client's session will not be interrupted. The Load Balancer may also perform role of the first layer of security, and drop malicious requests before they even reach servers. Load Balancing server also provides single point of access to the cluster. Thus the client needs to connect to a single IP address making DNS look-up for the client's browser easier.
- `Session` object and all its fields must implement java.io.Serializable interface as it will be converted to bytecode by JVM to send it to another JVM.
- `Sticky Session` is when the load balancer remembers, which Tomcat server processes requests from a given client session. Thus all requests from the same client are sent to the same server.
- `JK-based Software Load Balancer mode options` is mod_jk, mod_proxy, mod_cluster is a mode of a load balancer being used for the web application. Read more here. Those options are known to be implemented by Apache Httpd Web Server, which is used as a load balancer. Read more about `mod_proxy` for Apache server here. `mod_proxy` is easier to configure Apache server module for implementing clustering and session replication for Tomcat servers (mostly small clusters). Read comparison on Apache FAQ. We will use `mod_jk` as it provides more load balancing features and is a common choice for the production clusters.

- `High Availability` is the set of technologies aimed at providing some guarantees that the application's service will be available for the clients for the longest possible time. It is also known as web application up-time, and it is usually expected to be 100%. Load Balancer, clustering, and session replication technologies aim at providing high availability to the web applications deployed on the cluster. There are also two levels of failover that relate to high availability `request-level` and `session-level`.
  - `Request-Level` failover is when the server that was serving requests goes down, all subsequent requests to that server are redirected by the `Load Balancer` to another server.
  - `Session-Level` failover is related to session replication discussed in this example and requires either session backup or session copying across all servers in the cluster to be configured. If session replication configured, when load balancer redirects requests from the failed server to another server that server will use its copy of the session to continue client's session from where it was before the failure. Thus the client will not notice any service interruption, which is the goal of high availability strategy.

Why we need clustering and session replication . One of the main advantages of using clustering and session replication is to provide scalability to a web application when demand for provided services increases. Another reason is reliability. For example if one server goes down another server can pick up and continue serving client request based on replicated session. Thus clustering and session replication provide reliability and scalability of your web application. Read more on why and what for clustering and session replication here.

Clustering and session replication is normally used with load balancing server that distributes load to cluster of servers that will process incoming requests. When a load balancing server decides which worker to send the request two it can make such decision in several ways. The way we implement in our example is the load balancer sends requests to the server that accepted first request from the client with the same session id. This mechanism is called sticky sessions.

In general there are two types of sessions. `Sticky sessions` are those that get attached to a specific Tomcat server and any future requests related to that session are serviced by that server. Requests for a not sticky session can be serviced by any server every time request is made. To use sticky session or not to use? If you want reliability via redundancy use sticky session. The choice is between reliability and performance, where non sticky session replication provides higher performance, while sticky session approach provides higher reliability.

Using `sticky sessions` with `session replication`, what does it do? It gives a nice fail over feature to your web application. If the server assigned to the session goes down, another Tomcat server will be assigned by the load balancer to pick up requests for that session and will continue serving any requests for that session . This is possible because the second Tomcat server had a copy of the session, thus it was able to continue serving requests as if nothing have happened.

In addition to deciding on `session replication` type, we need to decide on how we want our cluster to scale. There are several ways to construct your cluster. One of those design constructs is vertical clusters. Vertical cluster of Tomcat instances is when all instances are on the same physical machine, such that you can add Tomcat instances, but you cannot add another physical machine and deploy Tomcat on it. In this example due to limiting hardware, we will discuss vertical clustering design. Please read more on clustering design options and their pros and cons in this article by open logic.

7.4 Example Outline

In this example we will discuss how to configure three Tomcat instances. Once we complete Tomcat cluster set-up, we will need to configure load balancing server for the cluster. We will use Apache Httpd Server version 2.4 for the load balancing as it is free and frequently used with a Tomcat based cluster load balancer. You can download Apache Httpd Server here.

The load balancing technique that we will use in this article is known as software based load balancing. Another technique for load balancing is hardware based and discussed in details here. Load balancer will act as gateway for our cluster, e.g. we will make the load balancing server aware of the Tomcat instances available in the cluster. Later we will discuss how to configure Apache Httpd Server as a load balancer with `mod_jk` module.

7.5 Preparing for Cluster set-up

As mentioned earlier a cluster could be on a single physical or virtual machine. In our case we will create cluster on Windows 7, single physical machine. We will just need to adjust port numbers for each Tomcat instance in the cluster to accommodate for such setting.

First before we set-up the cluster, we need to create a single Tomcat instance and deploy a sample web application to it. Please refer to my previous article on how to set-up a single Tomcat 8 instance on Windows 7.

Regarding sample web application, we will build upon a web application demonstrated in this example. We will extend the application to be cluster aware and to provide response to the client with current time stamp and session id. For the cluster we will use three Tomcat 8 instances. We will kill one of them, and then start it back and we will see the session replication benefit in action as perceived from client's perspective.

7.6 Extending Web Application form "Create Web Application Project with Maven" Example

Before continuing, please go to this article, and follow all steps related to setting up the web application with maven.

The final `web.xml` will look like below.

web.xml

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

<web-app
  version="3.0"
  xmlns="https://java.sun.com/xml/ns/javaee"
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://java.sun.com/xml/ns/javaee https://java.sun.com/xml/ns/javaee ↩
      /web-app_3_0.xsd">

 <display-name>Archetype Created Web Application</display-name>

 <servlet>
  <servlet-name><span style="text-decoration: underline;">mvc</span>-dispatcher</servlet- ↩
      name>
  <servlet-class>
   org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/view-resolver-servlet.xml</param-value>
    </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
  <servlet-name><span style="text-decoration: underline;">mvc</span>-dispatcher</servlet- ↩
      name>
  <url-pattern>/</url-pattern>
 </servlet-mapping>

 <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/view-resolver-servlet.xml</param-value>
 </context-param>

 <listener>
  <listener-class>
   org.springframework.web.context.ContextLoaderListener
  </listener-class>
 </listener>
         <distributable/>
</web-app>
```

The above web.xml file has only one additional element `<distributable/>`. This is all we need in web.xml to let Tomcat know that our web application can be distributable in the cluster.

Next we need to add a method `SessRepl` to com.javacodegeeks.controller.ApplicationController.java. We add this method mainly for verification of session replication capabilities. All requests for `localhost:8079/SampleWebApp/SessRepl` will be mapped to this method. This method will return an html page with the session id of the client who made request, and a timestamp of the current request.

ApplicationController.java

```
@RequestMapping(value = "/SessRepl", method = RequestMethod.GET)
public String SessRepl(ModelMap model) {

        // get session id create automatically by load balancer
        String ssId = RequestContextHolder.currentRequestAttributes()
                        .getSessionId();
        // get date, month, year, hour, minute, second, and millisecond
        String currDate = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss:SSS")
                        .format(new Date());
        model.addAttribute("msgArgument",
                        "Maven Java Web Application Project: Success! Session id is "
                                        + ssId + " current date is " + currDate);

        return "index";
}
```

Finally, make sure to move `index.jsp` to `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\WEB-INF\pages\` folder.

That is it, our web application is ready to be built as explained here. Once you build it, get `SampleWebApp.war` file from the `target` folder and copy that war to each Tomcat instance on the cluster.

7.7 Clustering and Session Replication Technology Review

Just to recap, session replication implies that a org.apache.catalina.Session for the specific client will be copied in its entirety to all Tomcat instances in the cluster. By default the Tomcat instance is discovered in the cluster using UDP multicast. Read more on multicast here and here.

When deciding on how big your cluster should be, you need to consider expected load on your web application, especially number of simulations users. You also need to take into account your Tomcat settings discussed here, and make sure that each instance has enough computing resources (memory is the most important one) for processing requests.

Note that if your application is very resource intensive and has huge (millions of users) load, then your cluster and session replication configuration will need to be different than the one explained in this example. Yet there are many similarities in configuring large and small clusters.

Just to review what we are trying to accomplish in this example. We have several elements: Apache Httpd server acting as a `Load Balancer`, `Tomcat`; servers acting as workers serving requests; and session replication performed on the background cooperatively by all Tomcat servers. From developer point of view it is few settings, some performance and availability guarantee, and more servers to maintain. For user it is the same speed of response no matter how many other users are out there, and uninterrupted service even if some servers may fail.

No let's move to fun part building it all up: the cluster, the session replication, and the load balancer. In a sense load balancer make it all work together, and it could be seen as a conductor because it orchestrates servers and client requests.

7.8 Setting Up a Cluster

7.8.1 Setting Up an Apache Httpd Server for cluster management

As mentioned earlier we will use Apache Httpd Web Server as a load balancer. For load balancing capabilities we will add `mod_jk` connector module to the server. This module provides interoperability between Apache and Tomcat servers as well as

some load balancing functionality. This is a more mature, and load balancing feature reach module, which is often preferred module for production clusters. This module also offers fine grained monitoring and management of the cluster allowing taking offline for maintenance and bringing back online live Tomcat server instances. `mod_jk` connector module uses `AJP` protocol, which is an efficient protocol developed specifically for metadata communication between Apache and other servers.

There are is another newer module that also gains popularity. `mod_proxy_http` module is a simple alternative to `mod_jk` module, and is easier to configure. There are several variations of this module available for Apache Httpd Web Server. If you are interested in learning more on the difference between `mod_jk` and `mod_proxy` and their functionalities read here, here, and here.

Since we have chosen `mod_jk` module, we will need to do a little more setting up than for `mod_proxy`. We will need to modify `httpd.conf` Apache Web Server's main configuration file, and add a new file which will be used to manage Tomcat instances in the cluster.

7.8.2 Configure Tomcat server instances for cluster

This is a next step for Tomcat instance configuration. I assume that you have done the following steps earlier for each Tomcat instance:

- Download Apache Tomcat 8
- Unzip Tomcat and create two copies of the unzipped folder (thus you will have 3 folders of the same Tomcat binary).
- Add/change some configurations in server.xml as explained below

The directory with copies of Tomcat folders will look like below.

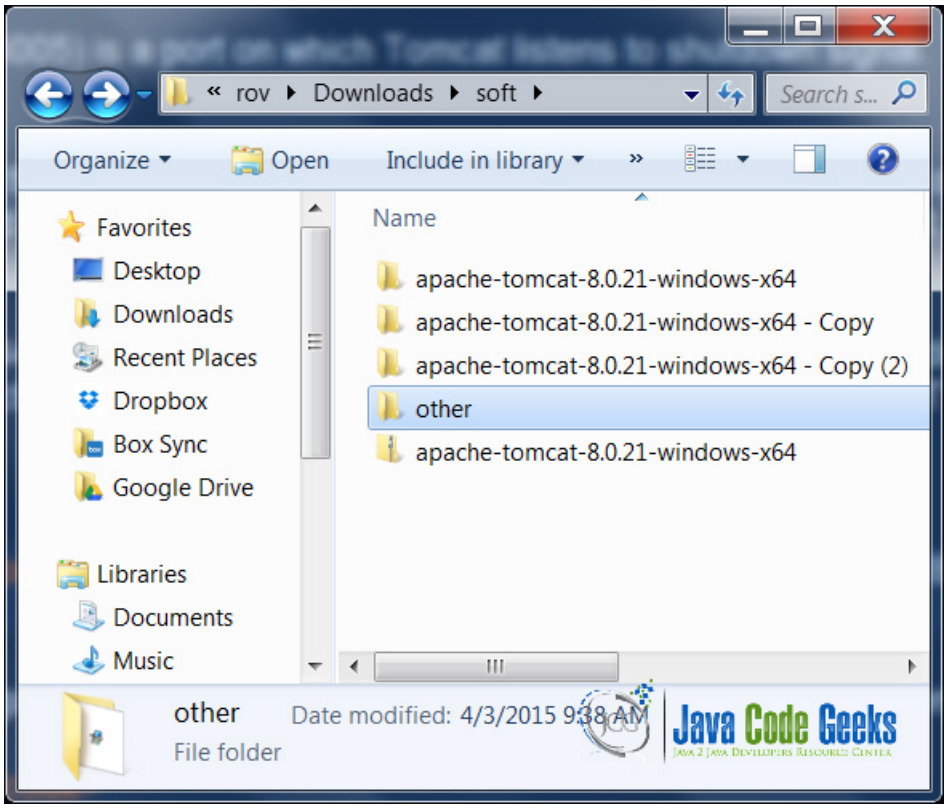

Figure 7.1: dirStructWin

Now, we will need to change the following ports for each Tomcat instance in `TOMCAT_ROOT_DIR\conf\server.xml`

- `Connector Port` (default 8080) is a port on which Tomcat listens for incoming HTTP requests.
- `Shutdown Port` (default 8005) is a port on which Tomcat listens to shutdown signal.
- `AJP Port` (default 8009) this port's name acronym stands for Apache JServ Protocol. This port is used to map requests based on certain configurations from Web Server, such as Apache Httpd Web Server to a worker server, such as Apache Tomcat.
- `Redirect Port` (default 8443) is a port used by Apache Tomcat internally for any redirection. There are two XML elements we need to update, when modifying this value, e.g. `Connector` for AJP, and `Connector` for HTTP/1.1.

Leave the first Tomcat instance with default values. For each next server, when modifying ports listed above in `TOMCAT_ROOT_DIR\conf\server.xml` just add `1` to the default port value, e.g. `Connector Port ...8080` will become `Connector Port ...8081` for the second Tomcat instance, etc.

Once you have modified all necessary ports, run automatic Windows service configuration on each server, but make sure you provide different `service name` for each Tomcat instance.

For each Tomcat instance go to `TOMCAT_ROOT_DIR\bin` and run `service.bat install [service name]`. One of possible commands would be `service.bat install Tomcat_1`.

Thus you will run the following commands, For Tomcat instance with AJP Port 8009 `service.bat install Tomcat_1` For Tomcat instance with AJP Port 8010 `service.bat install Tomcat_2` For Tomcat instance with AJP Port 8011 `service.bat install Tomcat_3`

Below is the result of running above commands for one of the Tomcat instance.

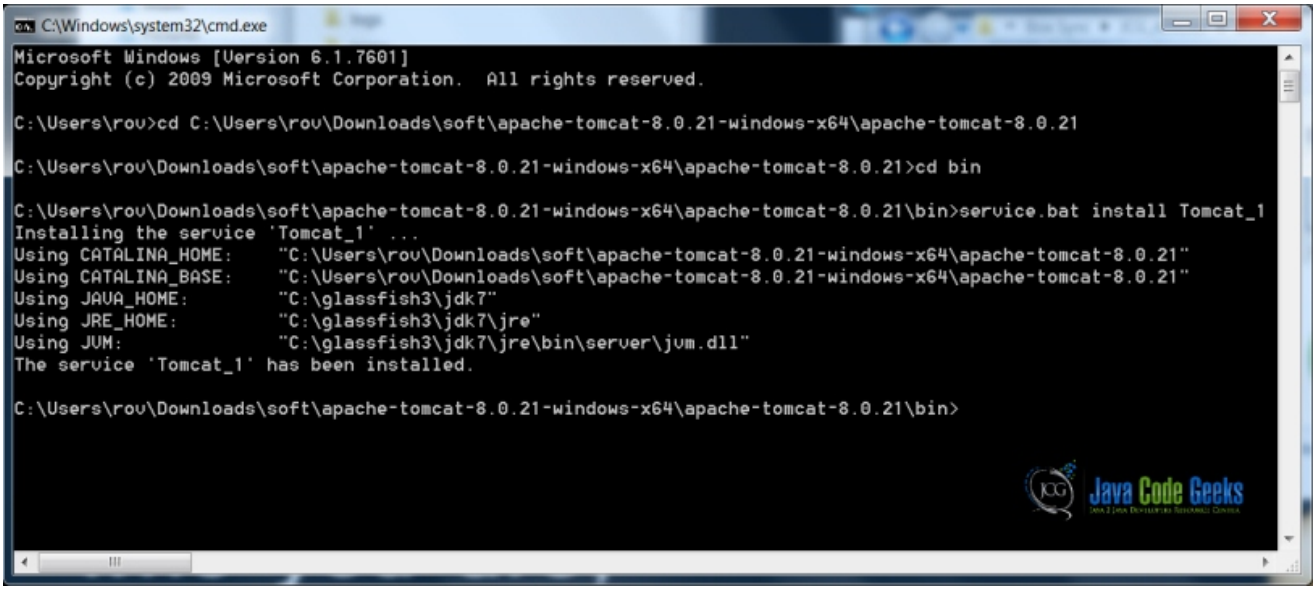

Figure 7.2: configTomcatServ

The reason we need to provide different names for each Tomcat service, so that they can be run in parallel as Windows services. Next, start each Tomcat instance as windows service with the same name as defined in the previous step.

For each Tomcat instance go to `TOMCAT_ROOT_DIR\bin` folder and run this command: `tomcat8 //TS//Tomcat_X`, where X should be replaced with corresponding number. `//TS//<service name>` is a command line directive to run a service with a given name. Read more on the available command line directives for Tomcat here.

7.8.3 Installing Apache Httpd Web Server

First go to official Apache web site click on `Binaries` and follow the download instructions. For my case, I had to go to the nearest mirror of the Apache binary download here. Then I clicked on `win32` link. Then I read windows related instructions

and warnings. I used `ApacheHaus` binary option for the Apache Httpd Web Server. In the download web site, I downloaded `Apache 2.4.12 x64`.

Once downloaded, unzip the Apache Web Server. Then read `readme_first` html file in the unzipped folder. To simplify configuration move `Apache24` folder to the root of your disk drive, for example `c:\Apache24`.

Now, run `cmd` as administrator, this is needed to install the server as a Windows service, which is a common practice.

Next, we need to change default port for Apache Httpd Web Server. The default is `port 80`. We will change it to `port 8079` to avoid conflicts with the ports taken by Tomcat instances.

Go to `c:\Apache24\conf`. Run `notepad httpd.conf`. We need to change two lines here:

- Find `Listen 80` and change it to `Listen 8079`. Listen allows you to bind the server to specific IP addresses and ports.
- Next find `ServerName localhost:80` and change it to `ServerName localhost:8079`. ServerName specifies the name and the port that this server will use to identify itself.

Thus you have modified two lines.

Next run `httpd -k install` in `c:\Apache24\bin` folder to install Apache Httpd Web Server as a service. To start it as a service in the same folder type `httpd -k start`. Test the server by going to url `https://localhost:8079/`. You should see a web page with some welcome text. Apache Httpd Web Server is ready.

Please see below running install for Apache Httpd Web Server

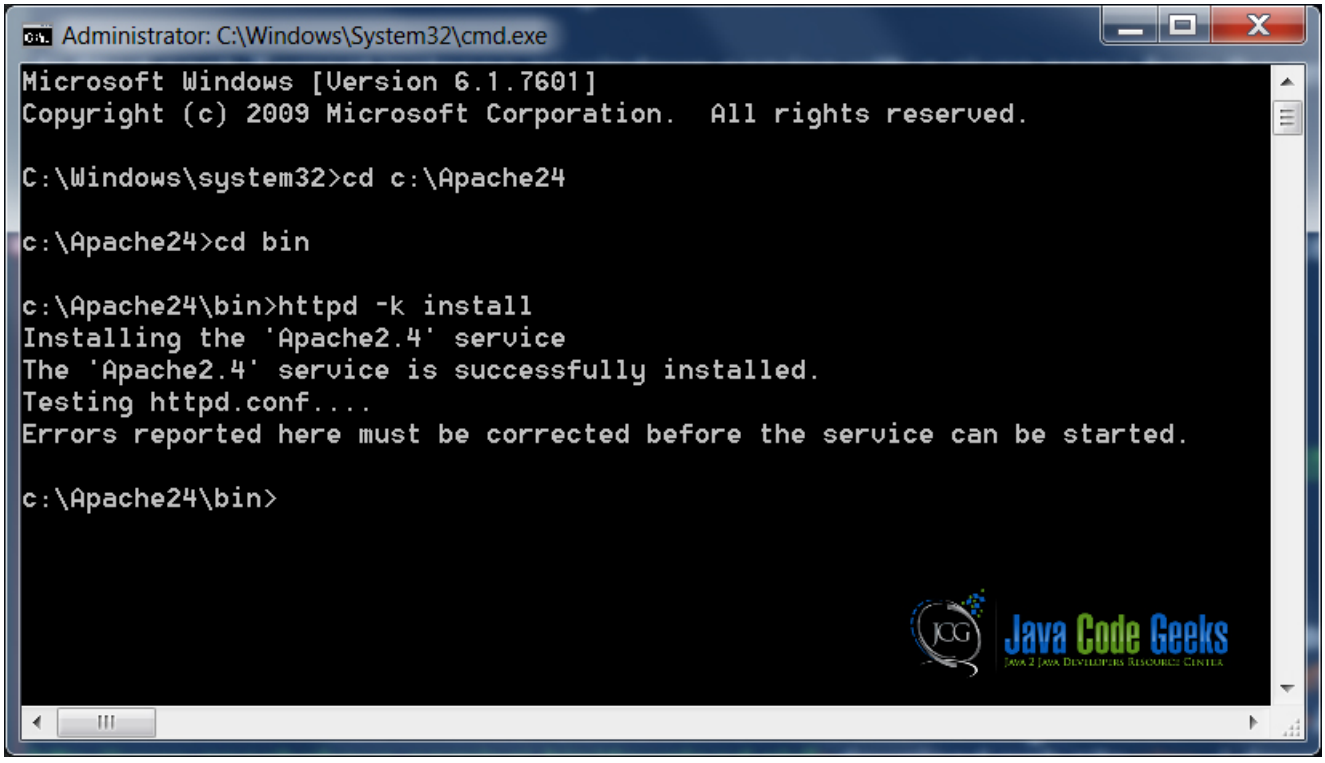

Figure 7.3: instServApacheHttpd

Now that you have three Tomcat instances ready to run, and Apache Httpd Web Server ready to run, we can move on to gluing them all together in a nice robust cluster.

7.8.4 Adding mod_jk load balancing module to the Apache Httpd Web Server

From now on all configuration is targeting at enabling servers to be cluster-aware. In this section we will enable Apache Httpd Web Server to be load balancer, and to know its workers in the cluster.

First we will need to add mod_jk module to the Apache Httpd Web Server for load balancing capabilities. To get it as binary go to this link. Click on windows link and download tomcat-connectors-1.2.40-windows-x86_64-httpd-2.4.x. Next, unzip the downloaded file and read the README file in it. Go to Apache's official website to read about connectors. After you unzip the module place its folder in C:\Apache24\modules folder of Apache Httpd Web Server. Next we will add necessary entries to Apache Httpd Web Server's configuration files to enable mod_jk module.

The configuration process is not simple as mod_jk configuration involves both load balancing and proxy related settings.

7.8.4.1 Configure mod_jk in C:\Apache24\conf\httpd.conf

First we need to configure the module that we just have added by adding the following entries in C:\Apache24\conf\httpd.conf.

```
# we need to instruct load of the new module
LoadModule jk_module C:\Apache24\modules\tomcat-connectors-1.2.40-windows-x86_64-httpd-2.4. ↩
    x\mod_jk.so

# Then we need to create, and point here the path to the worker configuration file
JkWorkersFile C:\Apache24\conf\workers.properties

# Then we need to configure and create if needed a folder, where to store information  ↩
    related to logging and memory usage
JkShmFile  C:\Apache24\logs\mod_jk.shm
JkLogFile C:\Apache24\logs\mod_jk.log

JkLogLevel info

# Now you will need to monitoring of the cluster
JkMount /jkmanager/* jkstatus
<Location /jkmanager>
        Order deny, allow
        Deny from all
        Allow from localhost
</Location>

# Next configure applications. Map all requests to our web application to the load balancer ↩
    .
JkMount  /* LoadBalancer
```

Let's take a closer look at each parameter.

LoadModule this configuration directive instructs loading of the mod_jk module.

JkWorkersFile defines the path from which workers configuration file will be loaded for load balancing purposes. Workers in our case are Tomcat server instances. We will populate this file in the next step.

JkShmFile the path for the shared memory files for the mod_jk module.

JkLogFile the path for the mod_jk module's log file.

JkLogLevel sets the logging level for the mod_jk module. Other levels are: debug, and error.

JkMount we use this directive to map all URL patterns related to any and our sample web application to the LoadBalancer virtual worker, which will distribute work among Tomcat workers. To enable load balancing for only specific Java applications modify to mapping /application/*.

Location this entry is used for security constraints. We will not configure security constraints for our simple example of load balancing.

7.8.4.2 Configure C:\Apache24\conf\workers.properties file

After configuring httpd.conf, we need to create a workers.properties file. This file will contain configuration for "workers", where workers are Tomcat instances that will handle client requests. Virtual servers that will handle load balancing and monitoring are also considered by mod_jk module as workers. The idea of virtual workers is not used in mod_proxy module.

Below is the `C:\Apache24\conf\workers.properties` file for our example. Create this file with the below content.

```
# First we define virtual worker's list
worker.list=jkstatus, LoadBalancer

# Enable virtual workers earlier
worker.jkstatus.type=status
worker.LoadBalancer.type=lb

# Add Tomcat instances as workers, three workers in our case
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009

worker.worker2.type=ajp13
worker.worker2.host=localhost
worker.worker2.port=8010

worker.worker3.type=ajp13
worker.worker3.host=localhost
worker.worker3.port=8011

# Provide workers list to the load balancer
worker.LoadBalancer.balance_workers=worker1,worker2,worker3
```

Let's look at each configuration entry more closely.

`worker.list` is used to load workers on the Apache Httpd Web Server start up. Requests mapped in httpd.conf can only be mapped to workers from this list.

`worker.<name>.<dirctive>=<value>` in our case the name is a worker string with count, e.g. worker1. Names are case sensitive and should be composed of alphanumeric characters with dashes or underscores. Workers and the directives related to them are one of the reasons `mod_jk` is still preferred in production. Read more on possible workers and directives for them on Apache's official website here.

`worker.<name>.type=<value>` with type directive we declare load balancer "lb" or "status" virtual worker type. For the Tomcat workers the type refers to the communication protocol.

`worker.<name>.port=<value>` with port directive we declare the port number to access the worker. This is useful in our example as it allows us to have several Tomcat instances running on the same machine. The port value must match the value in `<Connector port="8009" protocol="AJP/1.3">`

`worker.<name>.balance_workers=<comma separated list>` this is the required load balancer directive and is used to associate group of workers (cluster) with the load balancer. You could have multiple load balancers for different clusters. For more advanced load balancer settings please refer to this official how-to.

`worker.loadbalancer.balance_workers` this list contains list of workers that correspond to Tomcat instances configured for clustering. Note that the names in this list must match the name in jvmRoute attribute in `<Engine name="Catalina" defaultHost="localhost" jvmRoute="worker1">` element, which is in `TOMCAT_ROOT_DIR\conf\server.xml` file.

7.8.5 Configuring Tomcat instances for the cluster

Now since we have set up `load balancer` we need to configure each Tomcat instance for our cluster. We will need to perform two steps. The first step will be to enable session replication on each Tomcat instance, and the second step will be to configure the cluster.

7.8.5.1 Enabling Session Replication Concept

In this example we will set up in memory Session Replication because all Tomcat instances are on the same machine. We will use org.apache.catalina.ha.session.DeltaManager, so that each Tomcat instance gets a copy of sessions on all other Tomcat instances.

The load balancer can ensure that requests are send to the same "worker" each time. We will use `sticky sessions` to ensure that requests are served by the same Tomcat instance. If `sticky sessions` is enabled on the load balancer. If a given request is routed to a particular Tomcat instance, all subsequent requests with matching session id will be mapped to the same instance. If this Tomcat instance fails, load balancer will select another worker to continue processing the requests for the given session id. Note that if replication is not used, the session will be lost, but requests will still be served by the next worker. Tomcat's approach of in-memory session replication depends on the `sticky sessions` for failover and normal load balancing. This option is enabled by default in `mod_jk` module that we have installed and configured earlier.

The `Engine` element's `JvmRoute` attribute enables Load Balancer match requests to the JVM responsible for maintaining state of a given session. This achieved by appending the name of the JVM to the SESSSIONID of the request being mapped. Then the appended name is matched with the name of a worker form the `workers.properties` file. When configuring the `Engine` element you need to make sure that the name of the `JvmRoute` attribute matches the name in the `workers.properties` file located in `C:\Apache24\conf` folder.

7.8.5.2 Configuring Tomcat Instances for Session Replication

To set Tomcat instances to be part of a cluster, we will need to modify `TOMCAT_ROOT_DIR\conf\server.xml` configuration file for each Tomcat instance. in the `server.xml` find `<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>` and uncomment it. We will add elements in this configuration file as follows.

`TOMCAT_ROOT_DIR\conf\server.xml`

```
...
        <!-- We added attribute jvmRoute to the existing element -->
        <Engine name="Catalina" defaultHost="localhost" jvmRoute="[worker_name]">
        ...
        <!-- We uncommented this element and will have many config in this element's body  ↩
            -->
        <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"/>
                        <Sender className="org.apache.catalina.tribes.transport. ↩
                            ReplicationTransmitter">
                                <Transport className="org.apache.catalina.tribes.transport. ↩
                                    nio.PooledParallelSender"/>
                        </Sender>
                        <Receiver className="org.apache.catalina.tribes.transport.nio. ↩
                            NioReceiver"
                        address="auto" port="4000" autoBind="100"
                        selectorTimeout="5000" maxThreads="6"/>
                        <Interceptor className="org.apache.catalina.tribes.group. ↩
                            interceptors.TcpFailureDetector"/>
                        <Interceptor className="org.apache.catalina.tribes.group. ↩
                            interceptors.MessageDispatch15Interceptor"/>
                        <Interceptor className="org.apache.catalina.tribes.group. ↩
                            interceptors.ThroughputInterceptor"/>
                </Channel>

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

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

        </Cluster>
        ...
...
```

The above modification to `TOMCAT_ROOT_DIR\conf\server.xml` must be done for all Tomcat instances that are specified in the `C:\Apache24\conf\workers.properties` file. You need to replace `[worker_name]` with the corresponding name from the `workers.properties` file. Make sure that names and ports in `workers.properties` file match with the names and ports in corresponding `TOMCAT_ROOT_DIR\conf\server.xml` file.

Let's look at each configuration entry more closely.

`Engine`. This element needs to have defaultHost set to the same value as in workers.properties. We have added attribute jvmRoute to the standard configuration of this element. The value of jvmRoute attribute must match the name of a worker in workers.properties. The jvmRoute value must be unique for every worker added to the cluster.

`Cluster`. Within this element all other clustering elements are nested. It also supports different clustering related attributes. The class name `org.apache.catalina.ha.tcp.SimpleTcpCluster` in the `Cluster` element is the Java class that provides clustering capabilities that are included with the standard distribution of Tomcat 5.X and higher.

`channelSendOptions="8"` attribute is related to selecting clustering communication method. 8 is asynchronous communication method.

`Manager`. This element is used for session management by Tomcat. Inside the Cluster element this element defines, which session replication manager to use with clustering. In our example we use `org.apache.catalina.ha.session.DeltaManager`, which is a standard cluster-aware session replication manager. It copies sessions from any Tomcat instance to all other Tomcat instances. `expireSessionsOnShutdown="false"` attribute prevents a failing Tomcat instance from destroying sessions on other instances within the same cluster.

`notifyListenersOnReplication="true"` setting allows notification of the ClusterListeners when a session has been modified.

`Channel`. This element handles all communication between Tomcat instances within a cluster. In our example Channel uses multicast for communication between workers within the cluster. Single point broadcasting could also be configured. This element contains other elements that are responsible for a smaller part of the communication within the cluster.

`Membership`. This element defines the address that all Tomcat instances will use to keep track of each other. In this example we have used standard values.

`McastService`. This is the communication method between Tomcat instances within this cluster.

`Sender`. This element together with transport element nested inside it is used to configure communication type. In our case we have configured Non-Blocking Input Output NIO type of communication transport.

'Receiver`. This receiver component, which gets messages from other Tomcat instances' Sender components. This element could have configuration for addresses, buffers, thread limits, and more. In our example we have configured Tomcat instances to automatically discover each other via an automatically generated address. Interceptors. Interceptors elements are used to add information to the messages passed between Tomcat instances within the cluster.

`TcpFailureDetector`. This interceptor detects delays that may prevent another member from updating its table due to timeout. To prevent that it may provide alternative connection via reliable transport TCP.

`MessageDispatch15Interceptor` . This interceptor sends messages to a thread pool to send them asynchronously.

`ThroughputInterceptor`. This interceptor prints out traffic related statistics and provides it to each Tomcat instance. There are more interceptors that you can add for monitoring, reliability, and other purposes. The order in which you define interceptors here is the same order in which they are executed because they are linked list together in the order defined here. Read about interceptors here.

`Valve`. This element is nested in Cluster element to provide filtering. This element has many cluster specific implementations.

`ClusterListener`. This element listens to all messages send between Tomcat instances in the cluster and intercepts some of them as needed. This element behaves similar to interceptors, except that instead of modifying messages and passing them to Receiver component, they are the destination for the messages they intercept.

Once you have added above elements to the `server.xml` of the Tomcat instance, you need to restart it and your Tomcat instance will be cluster enabled. Note that you need to add above to all Tomcat instances, which you want to be part of the cluster, and that value for `jvmRoute` must be unique within the cluster.

7.9 Verify

The way we verify our cluster with session replication is by displaying back to the connecting client the session id that is associated with that client, and a current time stamp with seconds, and milliseconds.

For testing first stop all servers if any of them were running. Then start Apache Httpd Web Server as a Load Balancer with `mod_jk` module enabled. Then start three Tomcat instances. You need to use different web browsers to test sessions. If you connect to the same web page from the same browser, no matter from which tab it will use your old session instead of creating a new one.

Thus we can use IE, Google Chrome, and FireFox to test our session replication system.

First connected via Anonymous Google Chrome to Load Balancer. In my case I got assigned to worker 3, then shutdown worker 3

Then connected via Google Chrome to Load Balancer. In my case I got assigned to worker 1 then shutdown worker 1

Then connected via FireFox to Load Balancer. In my case I got assigned to worker 2, then turned back on worker 3

Next connected via IE to Load Balancer. In my case I got assigned to worker 3.

The below image demostrates log for one of the Tomcat instances and the messages related to other workers connecting and disconnecting from the cluster.

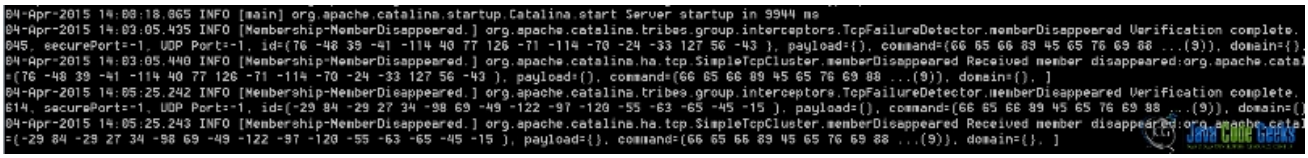

Figure 7.4: Cluster log

Finally, updated all connections. All previous sessions were preserved due to session replication, even though worker 1 is still down.

The image below demonstrates that the session id is related to different Tomcat instances (workers), even though some of them are down, the session they created is still active.

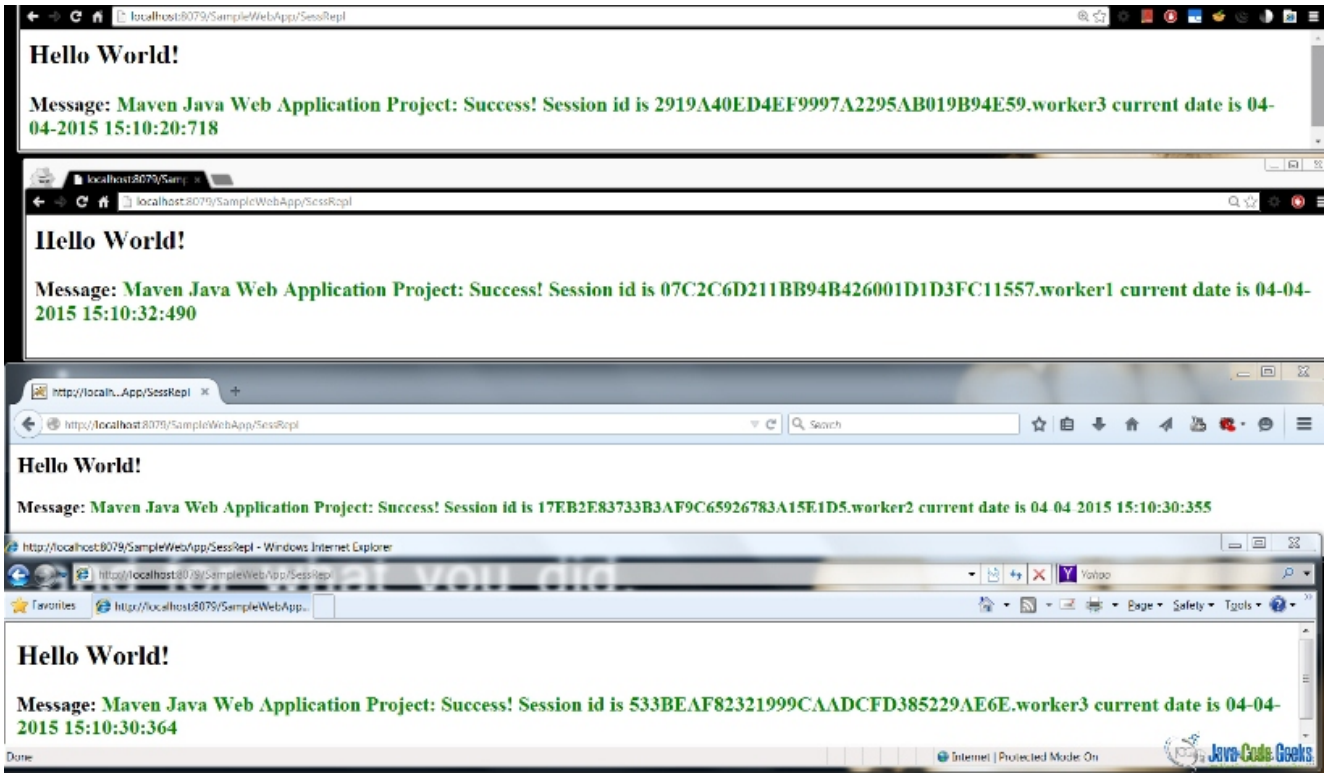

Figure 7.5: Session Replication

Thus as we can see from the screenshot and steps described session replication worked.

7.9.1 Conclusion

Clustering and session replication need to be configured only once in most cases, but they provide very valuable services. Once your web application becomes as popular as Facebook or Google, you will have to deal with a very large amount of load of requests and connections. There are many ways to deal with the increasing load, and the clustering is one of them. You basically keep adding more workers to the cluster. However, when you have many workers it becomes harder to keep all of them up all the time, thus some mechanism for failover becomes very important. Session replication is a well-known and widely used approach to provide failover guarantees to prevent service interruption. In this example we discussed one of the possible ways to set up cluster with session replication using open source technologies, such as Tomcat. There other possible ways to achieve this goal and below you can find some other good tutorials for doing similar configuration.

7.10 Download

Download

Download complete web app with tomcat server's, and Apache Httpd Web Server's configuration files here: **TomcatSessReplSampleWebApp**

7.11 Related articles

Apache Tomcat 8 Configuration Reference

Clustering/Session Replication HOW-TO

Load Balancer HOW-TO

Tomcat Clustering - A Step By Step Guide

An Introduction To Tomcat Cluster Configuration

Apache Tomcat mod_jk Connector Configuration Made Simple

Apache Tribes - Introduction

The Cluster Valve object

The Apache Tomcat Connector - Generic HowTo

The Apache Tomcat Connector

Working with mod_jk

The Apache Tomcat Connector - Webserver HowTo

The Apache Tomcat Connector - Reference Guide

LoadBalancer HowTo

FAQ/Connectors

Httpd FAQ

An In-Depth Look at Tomcat's Clustering Mechanisms

Chapter 8

Tomcat access log configuration example

This article describes the configuration of the access log for Apache Tomcat 7 web server.

A web server has the function to intercept user requests to resources, examine them and log them. This is an activity of web server administration.

Tomcat provides components called valves that can be placed in the request-processing stream (intercept user requests) for logging access to resources. This is a Tomcat specific interception mechanism.

Tomcat supplies the following standard valves for:

- Access logging
- Single sign-on for all web applications running on the server
- Request filtering/blocking by IP address and host name
- Detailed request dumps for debugging

Valves are specific to Tomcat and are not part of the Java Servlet specification. Web applications use filters as a similar interception mechanism; they are part of the Servlet specification and are defined for a web application. This article has details about access logging valves.

8.1 Access Log Valve

Access log valve logs requests. Access logs can be analyzed by standard log analysis tools to track page hit counts, user session activity, and so on.

An access log valve is not a logger. This does not use Apache Commons Logging (Tomcat's default logging mechanism). This Valve may be associated with any Catalina container - Context, Host, or Engine. The valve records all the requests processed by that container.

Valves are nested components in the Tomcat configuration hierarchy and can be placed inside <Engine>, <Host>, or <Context> containers of `CATALINA_HOME\conf\server.xml` configuration file. Each element has 0 or more Valve sub-elements.

NOTE: CATALINA_HOME environment variable is the Tomcat installation directory (conventionally named "apache-tomcat-[version]").

8.2 Configuring Access Log Valve

The Access Log Valve entry in `server.xml` looks like this for the engine, host or the context:

```
<Valve className="org.apache.catalina.valves.AccessLogValve"
    directory="logs"
    prefix="localhost_access_log."
    suffix=".txt"
    pattern="common"
    resolveHosts="false"/>
```

8.2.1 The Attributes for the Standard Access Log Valve

The `className` attribute represents the Java class of the valve. This must be `org.apache.catalina.valves.AccessLogValve`. This is a mandatory attribute. All other attributes for the valve are optional.

A complete list of all the attributes and their descriptions are found at: Apache Tomcat 7 Configuration Reference - The Valve Component Attributes

8.2.2 The Access Log File

The access log output file will be placed in the directory given by the "directory" attribute (default value is `CATALINA_BASE/logs`). The name of the file is composed by concatenation of the configured "prefix" (default value is "access_log.") and "suffix" (default value is zero length string - "") attributes. The format of the timestamp in the file name can be set using the "fileDateFormat" attribute (default value is "yyyy-MM-dd"). This timestamp will be omitted if the file rotation is switched off by setting "rotatable" to "false" (default value is "true").

The typical default log file name is: `access_log.2015-02-24`

NOTE: CATALINA_BASE environment variable specifies location of the root directory of the "active configuration" of Tomcat. It is optional to define this variable. By default it is equal to CATALINA_HOME.

8.2.3 Valve Chaining

Valves can be chained together to work in conjunction with each other so that requests go through more than one filter before reaching their destinations. For example, a valve configured at the engine level will intercept all requests to contexts on this engine, no matter what host or context they are bound for.

8.3 An Example Access Log Valve

This shows an example access log valve, log file and examines some of its attributes. This is the default configuration example as found in Tomcat 7 new installation.

Navigate to the `CATALINA_HOME\conf\server.xml`, and the `localhost` entry. The following is found:

```
<Engine name="Catalina" defaultHost="localhost">
    ...
    <Host name="localhost" ...
        <!-- Access log processes all example.
               Documentation at: /docs/config/valve.html
               Note: The pattern used is equivalent to using pattern="common"
        -->
        <Valve className="org.apache.catalina.valves.AccessLogValve"
            directory="logs"
            prefix="localhost_access_log."
            suffix=".txt"
            pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b"
        />
    </Host>

</Engine>
```

In case the Valve entry is commented, un-comment this, start (or restart) Tomcat. Point a web browser to https:// localhost:8080; this shows the default Tomcat welcome page.

The above configuration will create log files, in the common format, in `CATALINA_HOME/logs` directory. Now examine the directory, and open the "localhost_access_log.DATE.txt" (for example, "localhost_access_log.2015-02-24.txt") file. This is the access log. Note that from the file name the log file Valve attributes "prefix" is "localhost_access_log.", "fileDateFormat" is "YYYY-MM-DD" and the "suffix" is ".txt".

The access log has entries for the accessed web page and also for the associated image files. This is in the common log file format as shown below:

```
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:41 +0530] "GET / HTTP/1.1" 200 11418
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:41 +0530] "GET /tomcat.css HTTP/1.1" 200 5926
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:41 +0530] "GET /favicon.ico HTTP/1.1" 200 21630
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:41 +0530] "GET /tomcat.png HTTP/1.1" 200 5103
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:41 +0530] "GET /bg-nav.png HTTP/1.1" 200 1401
...
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:45 +0530] "GET /docs/ HTTP/1.1" 200 19367
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:45 +0530] "GET /docs/images/asf-logo.gif HTTP/1.1"  ↩
    200 7279
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:45 +0530] "GET /docs/images/tomcat.gif HTTP/1.1" 200 ↩
     2066
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:06:52 +0530] "GET /docs/logging.html HTTP/1.1" 200  ↩
    38251
0:0:0:0:0:0:0:1 - - [24/Feb/2015:14:23:58 +0530] "GET /docs/config/valve.html HTTP/1.1" 200 ↩
     111016
0:0:0:0:0:0:0:1 - - [24/Feb/2015:15:56:41 +0530] "GET /docs/index.html HTTP/1.1" 200 19367
0:0:0:0:0:0:0:1 - - [24/Feb/2015:15:56:51 +0530] "GET / HTTP/1.1" 200 11418
0:0:0:0:0:0:0:1 - - [24/Feb/2015:15:57:02 +0530] "GET /manager/html HTTP/1.1" 401 2538
0:0:0:0:0:0:0:1 - admin [24/Feb/2015:15:57:10 +0530] "GET /manager/html HTTP/1.1" 200 15829
0:0:0:0:0:0:0:1 - admin [24/Feb/2015:15:57:10 +0530] "GET /manager/images/tomcat.gif HTTP ↩
    /1.1" 200 2066
0:0:0:0:0:0:0:1 - admin [24/Feb/2015:15:57:10 +0530] "GET /manager/images/asf-logo.gif HTTP ↩
    /1.1" 200 7279
```

In the valve configuration - the Valve attribute pattern="%h %l %u %t "%r" %s %b" defined above is the same as pattern="common" which corresponds to the Common Log Format. The details of the pattern codes are as follows:

- **%h** - Remote hostname (or IP address if the resolveHosts attribute is set to false; by default the value is false).
- **%l** - Remote logical user name; this is always a hyphen (-).
- **%u** - Remote user that has been authenticated. In the example, "admin" and a hyphen (-). If there is none, it's a hyphen (-).
- **%t** - Date and time in common log file format.
- **%r** - The first line of the request. In the example, "GET / HTTP/1.1" (note that this is configured to be shown within quotes ("")).
- **%s** - The HTTP status code of the response. In the example 200 is the OK status.
- **%b** - Bytes sent count, excluding HTTP headers, and shows a hyphen (-) if zero.

NOTE: See the above section 2.1. The Attributes for the Standard Access Log Valve for a link pointing to the complete list of the attributes.

Chapter 9

Tomcat connection pool configuration example

9.1 Introduction

In this example we will discuss Apache Tomcat Servlet/JSP container's connection pull configuration via JNDI (Java Naming and Directory Interface) resources. The connection pool we will look at is javax.sql.DataSource, which is a JDBC API for getting a connection instance to a database. In this example we will discuss setting a global DataSource for MySQL database.

javax.sql.DataSource is a factory for getting connections to different physical data sources. javax.sql.DataSource interface is registered with the naming service based on JNDI API. A data source driver allows accessed to the database via DataSource interface. A DataSource object is looked up in the context based on registered through JNDI Resource. The connection to the data source, such as database is attempted when javax.sql.DataSource's getConnection() method is called. Connection pooling has many benefits and is part of the Java EE standard implemented by Tomcat. It has many benefits one of which is increased performance and reduced connection creation and removal overhead due to connection reuse by the connection pool.

Use of JDBC API allows for flexibility of changing data source implementation driver form MySQL to Oracle for example, and allows using improved data source specific connection pool implementations. It also abstracts away database connection related plumbing, and allows developers to focus on business logic. Using configurations for setting connection pool also allows for server or web application specific connection pool tuning to meet demands of the application(s) on that particular server.

Read more about JNDI here. The Tomcat version used for this example is 8, the Windows version is 7 Professional SP1, and the data source discussed is MySQL server version 5.1.

Note that you can set up database connection in Java code as well. Read more from Apache's website, or this stackoverflow question.

Note that in this example we will focus on server wide connection pull configuration, yet similar steps may be performed to do a web application specific connection pool configuration. Configuring the connection pool as a global resource results in the connection pool that is shared by all web applications deployed in the container.

In this example we will create a test web application called "testwebapp", and a test database called "JCGExampleDB".

9.2 Environment

In this example 64-bit Windows 7 Professional SP1 was used. JDK 7 for Windows 7 64-bit was installed and configured.

Note that in this example we use Windows specific directory path separator \, which is different from the one used on Linux like systems. Below are critical directories for the Tomcat that will be referenced to in this example:

- `TOMCAT_ROOT_DIR` (known as $CATALINA_HOME, where catalina is the project name of the Tomcat server) is the directory where you have placed Apache Tomcat folder, e.g. C:\apache-tomcat-8.0.20-windows-x64\apache-tomcat-8.0.20 in our example. In addition, Tomcat server can be configured for multiple instances by defining $CATALINA_BASE for each instance of the Tomcat server. By default both folders refer to the single Tomcat server installation.

- `TOMCAT_ROOT_DIR\conf` is the directory that contains configuration files and related to those files DTDs (Document Type Definition). The files in this folder are used for the server level performance tuning, security, load balancing, etc. We will talk about `web.xml` configuration file located in this folder.
- `TOMCAT_ROOT_DIR\lib` is the directory that contains libraries that are shared by all web applications deployed in the container.
- `TOMCAT_ROOT_DIR\webapps` is the directory, where the webapps you place in the server are stored.
- `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR` is the directory, where a specific webapp is placed, when deployed on the server.
- `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\WEB-INF` is the directory, where the webapp's specific configuration files are placed. Those configuration files override container's configuration files.
- `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\META-INF` is the directory that holds web application specific context.xml configuration file.

9.2.1 Preparing Environment

First we need to have Tomcat setted up. There are many tutorials on how to do that. You could refer to the official Apache Tomcat website here, or you can refer to my previous example for Apache Tomcat set-up and configuration here.

Before we start talking about connection pull configuration for MySQL DataSource, you need to have running MySQL server. I assume that you have already installed and set up MySQL database. If you need a refresher or a reference on how to install MySQL on Windows refer to this official tutorial. You can download MySQL installer for Windows here. When installing MySQL accept defaults. Note, once you installed mysql you can use MySQL command line client for accessing MySQL as root.

9.2.2 Preparing MySQL server

Before we can proceed, you need to prepare MySQL server to have some data that will help you test the connection to the database.

Start "MySQL command line client" for accessing MySQL as root, and provide your root password.

First create a database with name "JCGExampleDB" by logging to mysql and executing this sql command `CREATE DATABASE JCGExampleDB;`

Now create a user with name "test" and with password "test" as follows `GRANT ALL PRIVILEGES ON JCGExampleDB.* TO test@localhost IDENTIFIED BY 'test' WITH GRANT OPTION;`. This command creates a user "test" with password "test" and grants to that user access to our database.

Now exit MySQL command line client, and open a cmd. Next, go to the MySQL installation folder. For me it was `C:\Program Files\MySQL\MySQL Server 5.6\bin\`, so I typed `cd C:\Program Files\MySQL\MySQL Server 5.6\bin\` in the cmd (command line prompt).

Next, we need to open mysql as a newly created user. Run this command line in the cmd prompt `mysql.exe -u test -ptest`

Now run : `use JCGExampleDB;` command to use the newly created database.

Next create a table as follows : `create table testtable ( id int not null);` It does not matter what this table holds, we will not populate it. That is it we have a database, a user, and a table in our database. Now we are ready to proceed.

9.3 Java DataBase Connectivity JDBC

JDBC API acts as an interface between Java applications and database systems allowing the same Java code base to be used with different database systems. Thus JDBC API provides decoupling of database management systems, such as MySQL, and web application. This decoupling is achieved by deploying a database system specific JDBC driver that must implement JDBC API primitives. In case of MySQL recommended JDBC drivers is Connector/J that can be downloaded from here. This JDBC driver translates JDBC API calls to the database specific primitives.

9.4 DataBase Connection Pooling DBCP

DataBase Connection Pooling is achieved with connection thread pooling. The goal of connection thread pooling is to allow serving many requests. The basic principal behind this approach is similar to the one used by java.util.concurrent.Executors when creating fixed thread pool.

In DBCP it works the following way. Depending on how you configured your DBCP either on Tomcat start up or on the web application deployment specific number of connection threads is generated. Whenever a new connection request comes, it is queued to the requests queue. If there is available free (idle) connection thread, the request from the queue is allocated that connection thread immediately, otherwise the request needs to wait in the queue until a connection thread becomes available.

When connection thread is allocated it becomes (active), until it is deallocated by the web application explicitly or by the DBCP implicitly (after abandoned wait time expires). This schema allows reusing connection threads, and avoiding creating a connection thread for each new connection request. In addition, it allows developers to assume that each connection request will have dedicated JDBC connection. We will discuss later in more details how to configure the connection pool.

9.5 JDBC Driver for MySQL

First you need to obtain the MySQL database JDBC driver called Connector/J, and place it in `TOMCAT_ROOT_DIR\lib`. You can get Connector/J JDBC for MySQL here. Select "Platform Independent option" from the "Select Platform" menu. Then you will be asked to sign up for an Oracle account. Once you have registered, you may download the zip. Unpack the zip in any folder, and copy the `mysql-connector-java-5.1.35-bin` to `TOMCAT_ROOT_DIR\lib` folder. As of writing of this article Connector/J with version 5.1.35 was used.

Note that copying Connector/J JDBC Resource factory driver to the `TOMCAT_ROOT_DIR\lib` folder makes it available to Tomcat's internal classes and to the web applications deployed in this Tomcat instance.

9.6 Configuration of JNDI Resource for Connection Pool

Now let's consider how to configure a database resource to be shared by multiple web applications. Now that you have JDBC driver for MySQL, you can configure Tomcat server to use it to access MySQL server. To make database accessible you need to configure it as a Java Naming and Directory Interface Resource. Based on the Java Servlet 3.1 specifications that Tomcat 8 implements, all JNDI Resources must be specified in two configuration files in the `TOMCAT_ROOT_DIR\conf` folder:

- server.xml a "Resource" entry should be added to allow JNDI to locate and JDBC to configure the DataSource. For per web application Resource configuration a file context.xml will need to be created in `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\META-INF` to add "Resource" entry.
- web.xml a "Resource Reference" entry should be added to provide a reference to the data source that does not have server specific information, which allows easier portability.

The Resource entry has database server specific configuration information and credentials. Since we will create Resource entry in the Tomcat's configuration folder, this resource will be globally available to all web applications deployed on this Tomcat instance.

Below is the fragment of the `TOMCAT_ROOT_DIR\conf\server.xml` file content with Resource entry:

server.xml:

```
<?xml version='1.0' encoding='utf-8'?>
...
  <GlobalNamingResources>
    ...
    <Resource name="jdbc/JCGExampleDB"
                          global="jdbc/JCGExampleDB"
                          factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
                          auth="Container"
```

```
            type="javax.sql.DataSource"
                        username="test"
                        password="test"
                        driverClassName="com.mysql.jdbc.Driver"
                        description="JCG Example MySQL database."
                        url="jdbc:mysql://localhost:3306/JCGExampleDB"
                        maxTotal="10"
                        maxIdle="10"
                        maxWaitMillis="10000"
                        removeAbandonedTimeout="300"
                        defaultAutoCommit="true" />
    ...
</GlobalNamingResources>
```

The element `Resource` in the `GlobalNamingResources` is used to configure name, data type, and other parameters of a resource that is made available to all web applications deployed on the container. Below is the description of each attribute:

`name` this attribute defines the global name for this resource that will be used to reference this resource configuration. Note that "jdbc" is required by convention, so that all Resource names resolve to "jdbc" subcontext, which is relative to the `java:comp/env` naming context.

The "Container" value for the `auth` attribute tells container (Tomcat) to login to the database for the web application. Another possible option is "Application", in this case web application must provide login credentials.

`global` this attribute defines the global name for this resource.

`factory` this attribute defines the factory to be used to create this resource.

`auth` this attribute defines who should perform authentication. In our case "Container" will perform authentication on behalf of the web application.

`type` this attribute defines type of the class that will be returned by this resource.

user name and password are used by the "Container" during authentication to the resource, database in our case.

`driverClassName` is a fully qualified Java class name of the JDBC driver. The name we used here "com.mysql.jdbc.Driver" is the official driver name for the recommended Connector/J MySQL's JDBC DataSource driver.

`url` is passed to the JDBC driver for connecting to the MySQL database.

`maxTotal` is the maximum number of connections that can be allocated from the connection pool at the same time. Make sure you configure MySQL's "max_connections" to handle maximum total simultaneous connections. It is desired to have pool size to be small, but not smaller. An optimal pool size is a web application specific and can be found by load testing and monitoring.

`maxIdle` is the maximum number of connections that can be idle (not used by web application) at the same time.

`maxWaitMillis` is the maximum time in milliseconds that the pool manager will wait for the connection to be released by the web application before throwing an exception. This wait is activated when there is no available connections. In our case it is 10 seconds.

`removeAbandonedTimeout` is the number of seconds to be waited before the active connection considered abandoned, which should be set to the time you expect you longest query to run. In our example we have the default value, you can change it based on your server needs.

`defaultAutoCommit` attribute defines if auto-commit of the state of the connection should be performed. In our example we have the default value, which is true.

For more attributes that can be specified for a Resource please refer to the Tomcat's official website here to the "JDBC Data Sources" section subsection 4. Note that the configuration parameters for the server's DataSource connection pool should be selected based on the expected needs of all web applications whereas the same parameters for per web application connection pool configuration should only be tuned for that particular web application.

Note:

- The user you specify in the Resource element must exist on MySQL server. Test accessing MySQL with that user name and the password that you specify in the Resource entry. You can read more on how to connect to MySQL server from command line prompt here.

- The Resource name must be unique within Tomcat instance as that name is used to resolve data source reference to the data source instance.
- If you plan to use Realms for managing database security, the Realm should refer to the Resource by the name defined in the `GlobalNamingResources`. Discussing Realm is outside of scope of this article.

Tip: configuring Resource per web application, as opposed to per server as in our example, allows that application to be more portable.

9.7 Creating "testwebapp" in TOMCAT_ROOT_DIR\webapps folder

Let's create a sample web application on our server called `testwebapp`. Note, it is very important that you create a web application project with `testwebapp` name, otherwise some of the configurations that I demonstrate later will not work.

Creating web application is outside of scope of this article. You can refer to the JCG example Create Web Application Project with Maven Example, which I followed when creating `testwebapp`.

You will need to follow all steps of that example, and once you are done you will need to change some files. The reason for changing that example is to make this example simpler, otherwise this example would need to be extended to explain the steps that you will perform in the "Creating a Dynamic Web Project in Eclipse" example.

In the article that I followed I had to change the final `web.xml` of the web appliction. The lines that are I had to add were few, but I will show here the whole updated `web.xml` and I will point out which lines I added.

web.xml:

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

<web-app
  version="3.0"
  xmlns="https://java.sun.com/xml/ns/javaee"
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://java.sun.com/xml/ns/javaee https://java.sun.com/xml/ns/javaee ↩
      /web-app_3_0.xsd">

 <display-name>Archetype Created Web Application</display-name>

 <servlet>
  <servlet-name><span style="text-decoration: underline;">mvc</span>-dispatcher</servlet- ↩
      name>
  <servlet-class>
   org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/view-resolver-servlet.xml</param-value>
    </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
  <servlet-name><span style="text-decoration: underline;">mvc</span>-dispatcher</servlet- ↩
      name>
  <url-pattern>/</url-pattern>
 </servlet-mapping>

 <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/view-resolver-servlet.xml</param-value>
 </context-param>
```

```
 <listener>
  <listener-class>
   org.springframework.web.context.ContextLoaderListener
  </listener-class>
 </listener>
</web-app>
</Context>
```

And the part that I had to add to make the example that I followed work on Tomcat 8 was the following:

```
 <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/view-resolver-servlet.xml</param-value>
 </init-param>
```

I added that part based on what was indicated on official spring's website here. Without that part I was getting an error that a dispatcher configuration file cannot be found.

Later we will change another file in that example.

9.8 Configuration of "Resource Link" element in context.xml

Tomcat has a separate namespace for the global resources for the container. To expose those resources to web applications you need to include `ResourceLink` element in the per web application context.xml configuration file. Global Resource must be specified as a resource link in the `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\META-INF\context.xml` file.

You will need to create `META-INF` folder and `context.xml` file in the web appliction project that you have created earlier. The path you create for folder and file will be as follows `\testwebapp\src\main\webapp\META-INF\context.xml`.

Add the following entry to the newly created `context.xml` file:

context.xml:

```
<Context>

        <ResourceLink name="jdbc/JCGExampleDB"
        global="jdbc/JCGExampleDB"
        type="javax.sql.DataSource"
        />

</Context>
```

Let's look at each attribute of the `ResourceLink` element.

- `name` attribute is the name of the link to be created. For consistency it is better to give the same name to the link as the name of the global resource.
- `global` attribute is the name of the global resource defined in the global JNDI context in `server.xml` configuration file.
- `type` attribute is the fully qualified Java class name expected to be returned on lookup of this resource performed in the web application.

Using **ResourceLink** ensures that the web application uses the same global resource, instead of creating a new one. Read more about resource link here. You can allow per web application authentication configuration by adding the `factory="org.apache.naming.factory.DataSourceLinkFactory"` attribute to the "ResourceLink" element.

9.9 Configuration of "Resource Reference" in web.xml

"Resource Reference" is needed to enable a web application to look up a Resource using "Context" element prepared for that web application on its deployment, and to keep track of "Resources" that application depends on. "Resource Reference" should be specified in the `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\WEB-INF\web.xml` file and may reference a global Resource or a web application specific resource. As stated earlier, "Resource Reference" allows easier web application portability, and hides away resource specific configurations. Below is an example of the `resource-ref` entry in the "testwebapp" web application's `web.xml` file.

web.xml:

```
<web-app>
...
        <resource-ref>
                <description>
                This is a reference to the global Resource for MySQL database connetion.
                </description>
                <res-ref-name>
                jdbc/JCGExampleDB
                </res-ref-name>
                <res-type>
                javax.sql.DataSource
                </res-type>
                <res-auth>
                Container
                </res-auth>
        </resource-ref>
...
</web-app>
```

The element `resource-ref` above is used as a reference to the object factory for resources, such as JDBC DataSource, a JavaMail Session, or any other custom object factories. In our case we use this element to reference JDBC DataSource resource factory. Let's look at each of the sub-elements:

- `description` element is used to provide description related to the resource reference.
- `res-ref-name` element is used to provide the name of the "Resource" referenced. Note that there must exist a "Resource" entry with the same name as in thin element.
- `res-type` element is used to define the type of the object factory generated by "Resource".
- `res-auth` element is used to specify who will authenticate into the "Resource". In our case the authentication will be performed for us by the "Container".

Tip: it is important to follow element order defined by Servlet Specification for the deployment descriptors outlined here.

Note: The value of "res-ref-name" must be a name of an existing Resource configured as a global resource in the `TOMCAT_ROOT_DIR\conf\server.xml` or a web application specific resource configured in the `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\META-INF\context.xml` file. In your web application Java code, you will use the value of "res-ref-name" to get javax.sql.DataSource object for getting database java.sql.Connection. That is it for configuration. Now to use JDBC for MySQL in your web application you need to restart Tomcat.

9.10 Accessing Database "Resource" in a Web Application

For demonstrating that database connection works we modified `ApplicationController.java` that we generated when following Create Web Application Project with Maven Example. The final result after modification looks like following:

ApplicationController.java:

```
package com.javacodegeeks.controller;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/")
public class ApplicationController {

	@RequestMapping(value = "/Test", method = RequestMethod.GET)
	public String welcome(ModelMap model) throws NamingException, SQLException {

		/**
		 * Get initial context that has references to all configurations and
		 * resources defined for this web application.
		 */
		Context initialContext = new InitialContext();

		/**
		 * Get Context object for all environment naming (JNDI), such as
		 * Resources configured for this web application.
		 */
		Context environmentContext = (Context) initialContext
				.lookup("java:comp/env");
		/**
		 * Name of the Resource we want to access.
		 */
		String dataResourceName = "jdbc/JCGExampleDB";
		/**
		 * Get the data source for the MySQL to request a connection.
		 */
		DataSource dataSource = (DataSource) environmentContext
				.lookup(dataResourceName);
		/**
		 * Request a Connection from the pool of connection threads.
		 */
		Connection conn = dataSource.getConnection();
		StringBuilder msg = new StringBuilder();
		/**
		 * Use Connection to query the database for a simple table listing.
		 * Statement will be closed automatically.
		 */
		try (Statement stm = conn.createStatement()) {
			String query = "show tables;";
			ResultSet rs = stm.executeQuery(query);
			// Store and return result of the query
			while (rs.next()) {
				msg.append(rs.getString("Tables_in_JCGExampleDB"));
```

```
                        }
                } catch (SQLException e) {
                        System.err.println(e.getMessage());
                } finally {
                        // Release connection back to the pool
                        if (conn != null) {
                                conn.close();
                        }
                        conn = null; // prevent any future access
                }

                model.addAttribute("msgArgument",
                                "Maven Java Web Application Project: Success! The show tables result is: "
                                                + msg.toString());

                return "index";
        }

        @RequestMapping(value = "/Print/{arg}", method = RequestMethod.GET)
        public String welcomeName(@PathVariable String arg, ModelMap model) {
                model.addAttribute("msgArgument",
                                "Maven Java Web Application Project, input variable: " + arg);

                return "index";
        }
}
```

The part that was added to the example controller code is the following:

ApplicationController.java:

```
...
                /**
                 * Get initial context that has references to all configurations and
                 * resources defined for this web application.
                 */
                Context initialContext = new InitialContext();

                /**
                 * Get Context object for all environment naming (JNDI), such as
                 * Resources configured for this web application.
                 */
                Context environmentContext = (Context) initialContext
                                .lookup("java:comp/env");
                /**
                 * Name of the Resource we want to access.
                 */
                String dataResourceName = "jdbc/JCGExampleDB";
                /**
                 * Get the data source for the MySQL to request a connection.
                 */
                DataSource dataSource = (DataSource) environmentContext
                                .lookup(dataResourceName);
                /**
                 * Request a Connection from the pool of connection threads.
                 */
                Connection conn = dataSource.getConnection();
                StringBuilder msg = new StringBuilder();
                /**
                 * Use Connection to query the database for a simple table listing.
```

```
                 * Statement will be closed automatically.
                 */
                try (Statement stm = conn.createStatement()) {
                        String query = "show tables;";
                        ResultSet rs = stm.executeQuery(query);
                        // Store and return result of the query
                        while (rs.next()) {
                                msg.append(rs.getString("Tables_in_JCGExampleDB"));
                        }
                } catch (SQLException e) {
                        System.err.println(e.getMessage());
                } finally {
                        // Release connection back to the pool
                        if (conn != null) {
                                conn.close();
                        }
                        conn = null; // prevent any future access
                }

                model.addAttribute("msgArgument",
                                "Maven Java Web Application Project: Success! The show ↩
                                    tables result is: "
                                                + msg.toString());
...
```

The above code demonstrates how to use JNDI to look up a resource by its name, and use resource's return class "DataSource" to get a "Connection" from the connection pool. Note, it is very important to release any borrowed resources, and close any resources that needs to be closed.

In the example above, Connection, Statement, and ResultSet needs to be closed. We closed Connection explicitly in the finally close of the try/catch block. The Statement is closed automatically as we used try with resources. This approach was introduced recently with Java 7 and allows to close classes automatically on exiting try/catch block, but such classes must implement AutoClosable interface.

Since Statement is closed automatically any open resources related to Statement, such as ResultSet are also closed automatically. By closing all borrowed resources, we avoid holding up connection, so that other connection requests are served faster.

Also note that we used two names to get two contexts. Those contexts have different scope. The first context retrieved with `java:comp/env` look up retrieves all configured resources for the web application. After having context with all resources, we need to select a specific "Resource" of the JDBC API abstraction class to manage connection to the database. Thus we had to call another look up with our pre configured resource called "jdbc/JCGExampleDB". Note how resource name matches the name specified in "res-ref-name" in web application specific web.xml configuration file.

We will not look in depth on how to use the java.sql.Connection. You can read more on how to use java.sql.Connection on oracle's official website here.

In our example, we closed java.sql.Connection once we were done using it. If we did not, the container would generate java.sql.SQLException after the wait time for the borrowed connection to be returned have expired. We can change this wait time in "Resource" entry as demonstrated above. Make sure that you close the connection only once. To achieve that you can encapsulate database related logic in a separate Java class, and declare appropriate open() and close() methods.

If you get connection time out java.sql.SQLException, it is more likely your "maxWait" attribute value in "Resource" is too small and needs to be increased to allow long queries to run to completion before container forcibly reclaims borrowed connection from the web application. The "maxWait" attribute value is recommended to be set between 10-15 sec as stated here.

9.11 Verify

To verify that all works you will need to do the following steps:

- Make sure that mysql server is running.

- Build web application project using `mvn package` command in the folder, where pom.xml is located.
- Copy the `testwebapp.war` to the `TOMCAT_ROOT_DIR\webapps` folder (delete old version if any).
- Start Tomcat service as explained here. The command is `tomcat8` and is run in the Tomcat's bin folder in cmd. In your favorite web browser (Google Chrome for me) go to this url `https://localhost:8080/testwebapp/Test`

In the end you should see the image below in the browser. Please feel free to contact me if anything goes wrong, so I could improve this post.

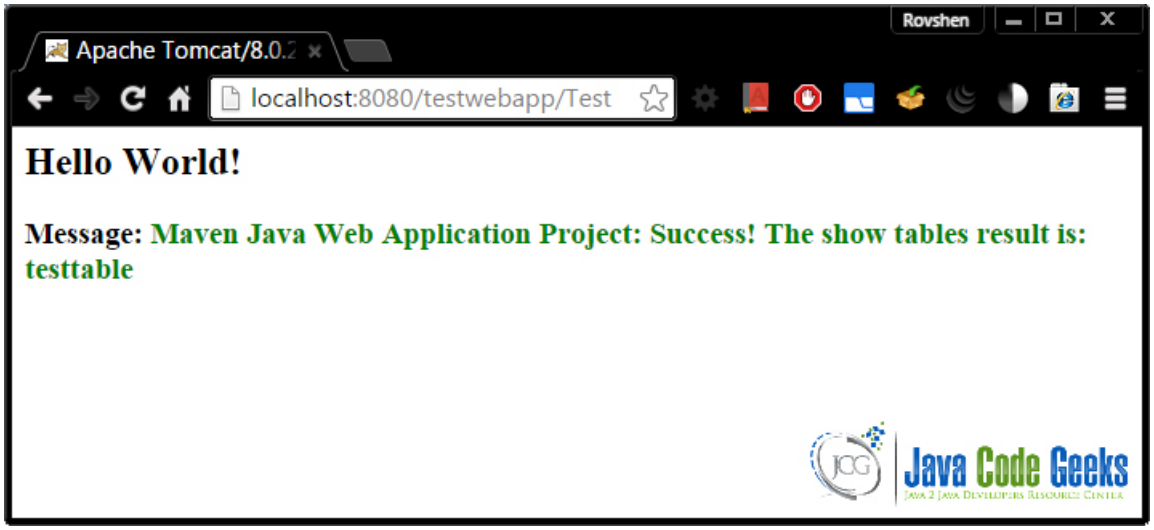

Figure 9.1: Database access result

9.12 Additional tips

If you forget to close a connection borrowed from the pool by forgetting to close ResultSet, Statement, or Connection the instance of connection you borrowed will never be returned back to the pool, creating connection pool "leak", which may result in database connection failure when there is no more available connections.

A proposed solution to this problem on the Apache Tomcat's official website here is to track and recover abandoned database connections. You can also configure DBCP to generate a stack trace to see which part of the web application opened resources and never closed them.

To remove and add back to the pool abandoned connection you can set `reomovedAbandoned="true"` attribute in the "Resource" element in the `server.xml`.

You can also set `logAbandoned="true"` to log a stack trace of the code segment that abandoned a database connection. Warning: use the logging with caution as it adds significant overhead.

Note that when you share a database connection pool, you may have many connections to the same database and thus will need to adjust necessary parameters in the "Resource" element entry in the server.xml file. Also in order to limit load on a particular database connection pool, you may need to move connection pool configuration from the server's configuration file to a web application specific configuration files located in `TOMCAT_ROOT_DIR\webapps\PROJECT_DIR\WEB-INF`.

Since The configurations used by Tomcat adhere to Java EE standards any web application with its configurations that you create on Tomcat could be ported to any other container that follows Java EE standards and provided JDBC facilities.

9.13 Conclusion

This example presents steps to configure global JDBC Resource to provide database access to the web application. The benefit of global Resource is that you can share one database by many web applications, and you can tune connection pool configuration

for those applications based on the usage by all of them.

Such sharing may be useful, when a dedicated database server is shared among many web applications located on the same Tomcat server. On the other hand, the drawback of global Resource configuration is that there will be more load on a single database, and per web application connection pool tuning will not be possible.

9.14 Download the Eclipse Project

Download

You can download the full source code of this example here: **TomcatConnectionPoolConfigurationExample**

9.15 Related posts

Tomcat MySQL Connection - Using JDBC to Connect Tomcat to MySQL

JNDI Datasource HOW-TO Tomcat 8

JNDI Resources HOW-TO Tomcat 8

JNDI Resources HOW-TO Tomcat 7

The Tomcat JDBC Connection Pool Tomcat 7.0.59

The Tomcat JDBC Connection Pool Tomcat 7.0.X

Connecting to MySQL Using the JDBC DriverManager Interface

Tomcat DataSource JNDI Example for Servlet Web Application

JDBC DataSource Example - Oracle, MySQL and Apache DBCP Tutorial

JDBC Example Tutorial - Drivers, Connection, Statement and ResultSet

How should I connect to JDBC database / datasource in a servlet based application?