Apache Tomcat

Administration

Agenda (1/2)

- Apache Tomcat
- Web Applications, JSP and Servlets
- Tomcat Installation
- Tomcat Architecture
- Basic Tomcat Configuration
- Advanced Tomcat Features
- Web Application Configuration
- Web Application Administration

Agenda (2/2)

- Tomcat and Apache Http Server
- JDBC Connectivity
- Tomcat Security
- Shared Tomcat Hosting
- Tomcat Monitoring and Management
- Clustering
- Logging
- Performance tuning

Apache Tomcat

What is Apache Tomcat?

- Apache Tomcat is a web Container.
- It comes from Apache Software Foundation(ASF).
- It is a reference implementation(RI) of Sun's (Oracle's) Java EE Web Container.
- Created initially by Sun as Java Web Server and later donated the code base to ASF.
- Distributed under Apache License.
- Supports JSP and Servlets.
- Tomcat started as a subproject of the Jakarta project, but now is independent of it.
- Tomcat can be freely used in any organization. It can be freely redistributed in any commercial project so long as its license is also included with the redistribution and proper recognition is given.

Tomcat Installation

Session Agenda

- Installing the Java Virtual Machine (JVM)
- Installing Tomcat on both Windows and Linux
- Understanding the Tomcat installation directory structure
- Troubleshooting typical problems encountered while installing Tomcat

Installing the Java Virtual Machine (JVM)

Get Java

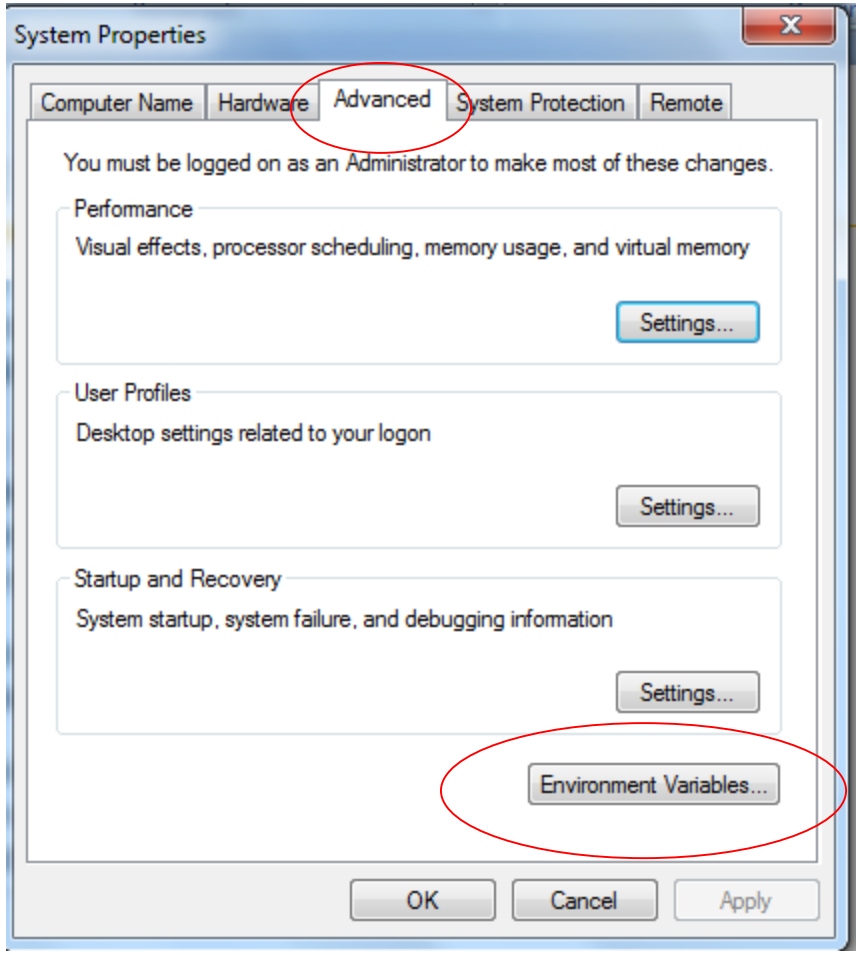
- Tomcat is based on Java language and hence we need JVM to run it.
- Tomcat 6.x requires Java 5 or later. We will use JDK6
- Get Java (JDK) from http://www.oracle.com/technetwork/java/javase/downloads/jdk6downloads-1902814.html
- Download JDK for your platform (Windows/Linux)

Install JDK on Windows

- We used http://download.oracle.com/otn-pub/java/jdk/6u43-b01/jdk-6u43-windows-x64.exe
- When you double click the .exe the installer will guide you through the installation process.
- On Windows, JDK by default will install it self in one of the following subdirectories:
  - In windows XP c:\Program Files\Java
  - In Windows 7 (32 bit) : c:\Program Files (x86)\Java
  - In Windows 7 (64 bit) : C:\Program Files\Java

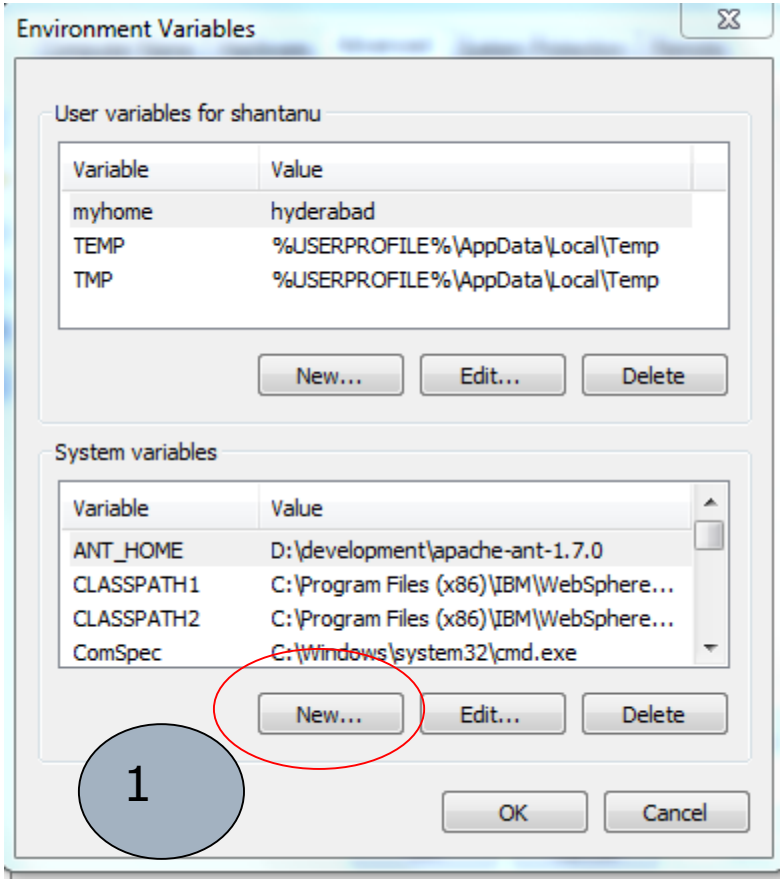
Add Java to your Environment

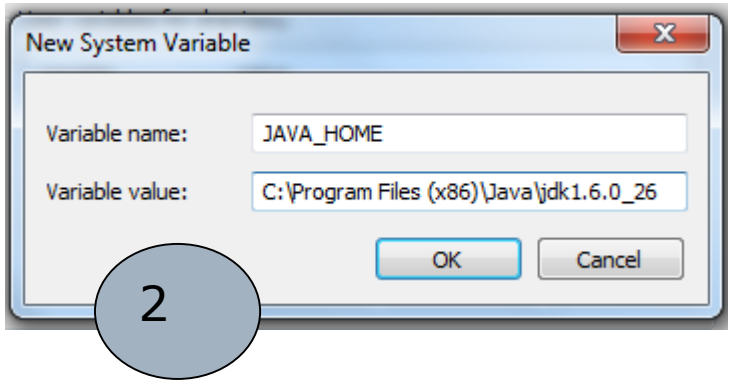
- To work with Tomcat properly you need to set **JAVA_HOME** env variable
- In Windows7 : start>control panel>system>Advanced System Settings
- Now System properties Dialog pops up
- Select Advanced>click on Environment Variables button



Add Java to your Environment

- Create a new System variable as described in the figure





Install JDK on Linux

- Download the appropriate JDK (e.g. jdk-6u35-linux-i586.bin)
- You should have root privileges.
- Copy the jdk installer to /usr/local/
- Run ./jdk-6u35-linux-i586.bin
- It will create a folder as /usr/local/jdk1.6.0_35
- Your JDk is now installed.

Add Java to your Environment (Linux)

- Create a file named javahome.sh in /etc/profile.d
- Make the following entries

  JAVA_HOME=/usr/local/jdk1.6.0_35

  PATH=$PATH:$JAVA_HOME/bin

  export JAVA_HOME PATH
- Save the file.
- Restart your OS
- Now your java environment is set in linux.

Download and Install Tomcat

Hands On Session

- Download Tomcat
- Install on windows
- Install as Windows Service
- Install on Linux

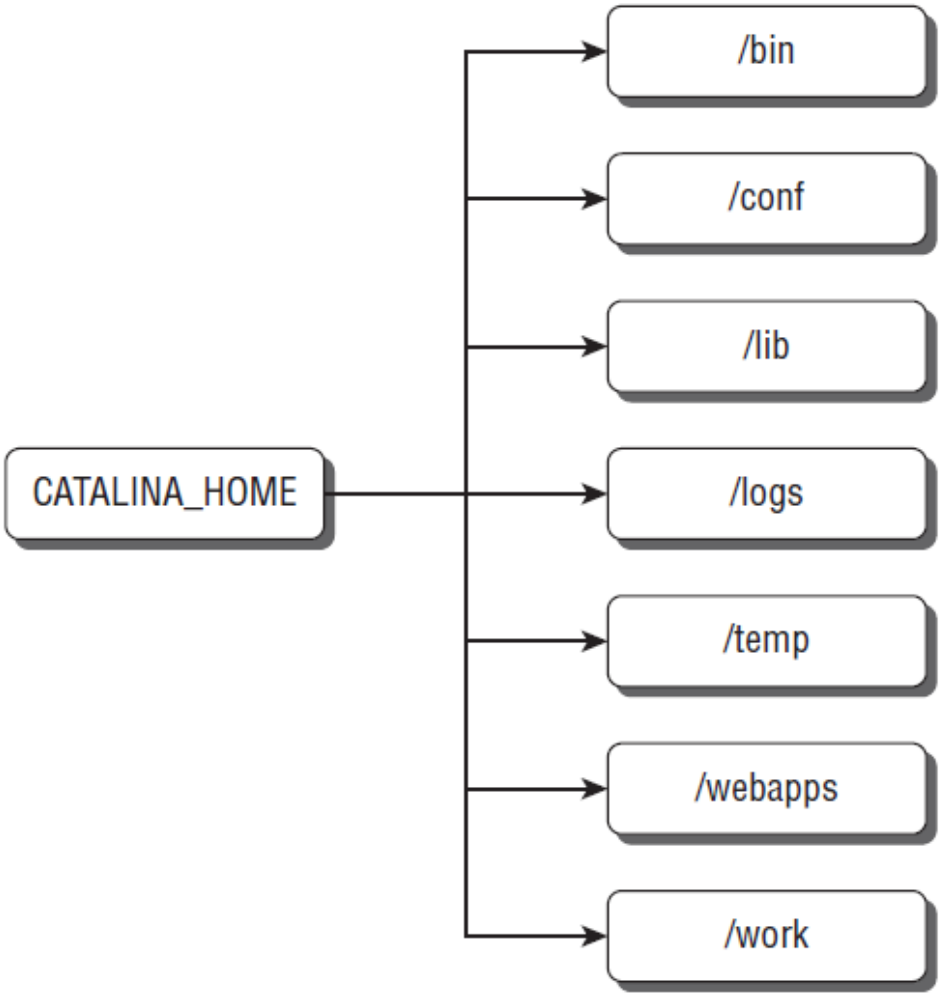
Tomcat Architecture

Session Agenda

- Tomcat directory structure
- Overview of the major Tomcat components
- Relationships between the components to make a full-service container

Tomcat Directory Structure

When you download and uncompress the Tomcat bundle, it places all of its contents in a folder or directory named apache-tomcat-6. *X.XX* where *X.XX represents the minor version numbers. This installation or top-level directory is what is* known as the CATALINA_HOME



Tomcat Directory Structure

- ***bin Directory***
  - The bin directory contains the scripts and code required to execute the server.
  - Depending on the version that you download, this directory may contain executable (.exe) files for a Windows service installation, or it will just contain the standard Java starting scripts.
  - Both Unix/Linux shell scripts (.sh) and Windows batch (.bat) scripts exist here for the standard Tomcat download.
  - Some JAR files also live in the bin directory, including bootstrap.jar , commons-daemon.jar , and tomcat-juli.jar .

Tomcat Directory Structure

- ***conf Directory***
  - The conf directory contains the files that are necessary to configure and set parameters for Tomcat when it executes.
  - When Tomcat launches, it investigates the files in this directory and alters/creates the necessary objects and settings to run.

Tomcat Directory Structure

- ***lib Directory***
  - The lib directory contains all of the JARs that are used by the container.
  - This includes the Tomcat JARs and the Servlet and JSP application programming interfaces (APIs).
  - This is the place where you place JARs that are shared across Web applications or JDBC JARs for connection pools.

Tomcat Directory Structure

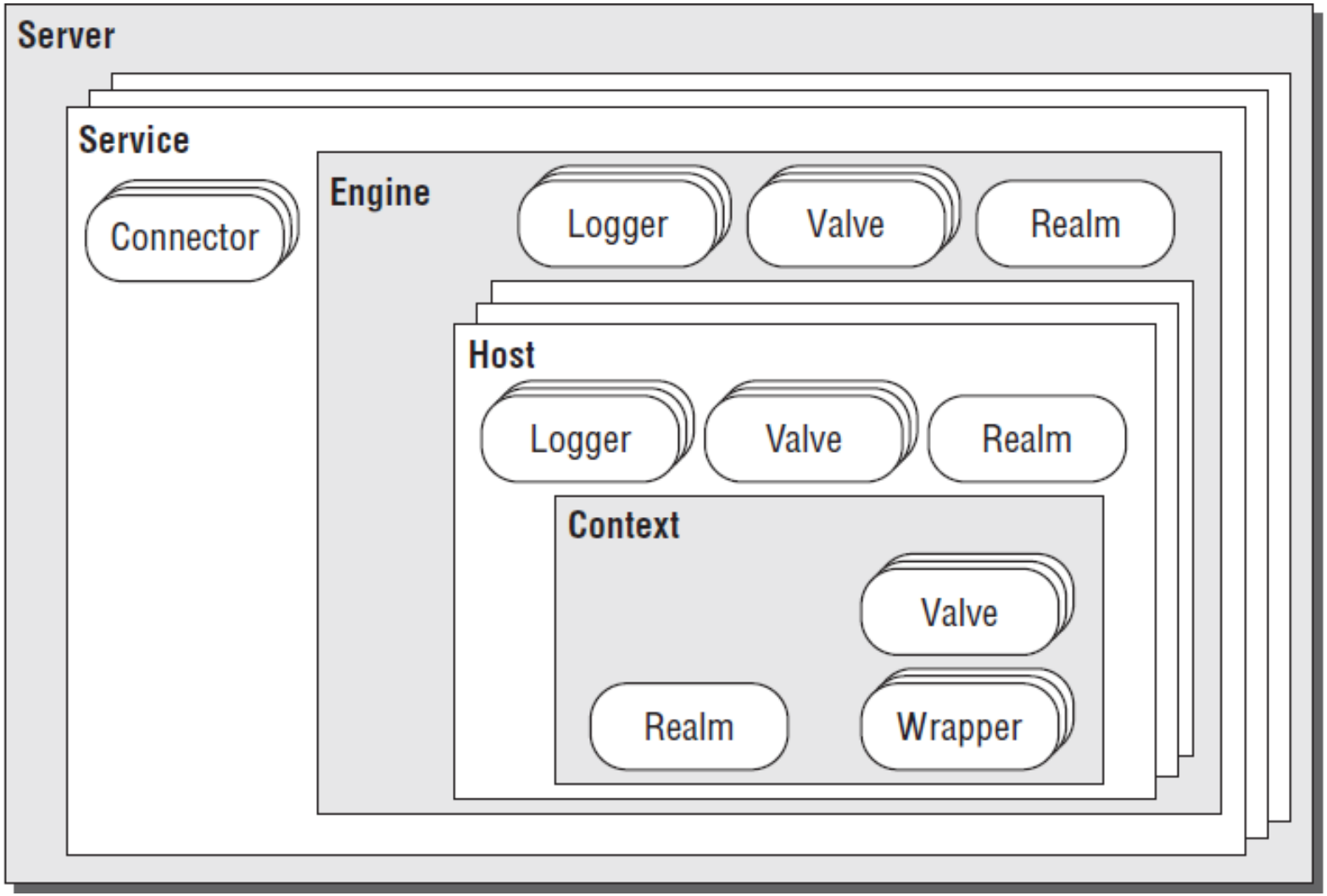
- ***logs Directory***
  - This directory is used for the logging files that are produced while Tomcat is running.
  - The JULI logging produces multiple files in this directory, and each log file is created for each day.
- ***temp Directory***
  - This is the temporary directory that Tomcat uses for scratch files and temporary use.
- ***work Directory***
  - Directory for temporary and working files.
  - This is heavily used during JSP compilation where the JSP is converted to a Java servlet and accessed through this directory.

Tomcat Directory Structure

- ***webapps Directory***
  - The webapps directory is where your Web applications ultimately will live.
  - If you are using an exploded WAR (a WAR file that is decompressed), it needs to be placed in this directory.
  - Placing a WAR file in this directory also causes Tomcat to deploy the file.
  - When you deploy a full WAR through the Manager console application or the Tomcat Client Deployer, your WAR file is also placed into this directory.

An Overview of Tomcat Architecture

- Tomcat 6 consists of a nested hierarchy of components.
- Some of these components are called *top-level components because they exist at the top of the component hierarchy in a rigid relationship with one* another.
- *Containers are components that can contain a collection of other components.*
- *Components that* can reside in containers, but cannot themselves contain other components, are called *nested components .*


Server
Service
Connector
Engine
Logger
Valve
Realm
Host
Logger
Valve
Realm
Context
Valve
Realm
Wrapper

The Server

- The *Server is Tomcat itself — an instance of the Web application server — and is a top-level component.*
- It owns a port that is used to shut down the server.
- *Only one instance of the Server can be created inside a given Java Virtual Machine (JVM).*

The Service

- A *Service groups a container (usually of type Engine) with a set of Connectors and is a top-level* component.
- *An Engine is a request-processing component that represents the Catalina Servlet engine.*
- *It examines the HTTP headers to determine the virtual host or context to which requests should be passed.*
- Each Service represents a grouping of Connectors (components that manage the connection between the client and server) and a single container, which accepts requests from the Connectors and processes the requests to present them to the appropriate Host.
- Each Service is named so that administrators can easily identify log messages sent from each Service.

The Connectors (1/2)

- *Connectors connect the applications to clients.*
- *They represent the point at which requests are received* from clients and are assigned a port on the server.
  - The default port for non-secure HTTP applications is kept as 8080 to avoid interference with any Web server running on the standard HTTP port, but there is no reason why this cannot be changed as long as the port is free.
- Multiple Connectors may be set up for a single Engine or Engine-level component, but they must have unique port numbers.

The Connectors (2/2)

- The default Connector is Coyote, which implements HTTP 1.1; Tomcat also comes with an AJP connector.
- In addition, the HTTP connector can be used with SSL as well.
- Both the HTTP and AJP connectors are fully supported by Tomcat.
- However, there are alternative Connectors, such as the old JServ and JK2, which work, but are no longer supported.

The Engine

- An Engine is a request-processing component that represents the Catalina Servlet engine.
- It examines the HTTP headers to determine the virtual host or context to which requests should be passed.
- An Engine may contain Hosts representing a group of Web applications and Contexts representing a single Web application.

The Realm (Under Engine)

- The Realm for an Engine manages user authentication and authorization.
- During the configuration of an application, the administrator sets the roles that are allowed for each resource or group of resources, and the Realm is used to enforce this policy.
- Realms can authenticate against text files, database tables, LDAP servers, and the Windows network identity of the user.

The Valves

- *Valves are components that enable Tomcat to intercept a request and preprocess it.*
- *They are similar to the* filter mechanism of the Servlet specifications, but are specific to Tomcat.
- Hosts, Contexts, and Engines may contain Valves. A Valve is essentially a super-filter, very similar to a servlet filter, but it intercepts and invokes at a much higher level.

Configuration By Architecture

```
<Server port="8005" shutdown="SHUTDOWN">
    <Listener className="org.apache.catalina.core.AprLifecycleListener"
        SSLEngine="on" />
    <Listener className="org.apache.catalina.core.JasperListener" />
    <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" />
    <Listener className="org.apache.catalina.mbeans.
        GlobalResourcesLifecycleListener" />
    <Service name="Catalina">
        <Connector port="8080" protocol="HTTP/1.1"
                maxThreads="150" connectionTimeout="20000"
                redirectPort="8443" />
   <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
   <Engine name="Catalina" defaultHost="localhost">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                resourceName="UserDatabase"/>
        <Host  name="localhost" appBase="webapps"
                     unpackWARs="true" autoDeploy="true"
               xmlValidation="false" xmlNamespaceAware="false"/>
      </Engine>
     </Service>
 </Server>
```

Basic Tomcat Configuration

Tomcat Configuration files

- A Tomcat 6 server instance reads a set of configuration XML files upon startup.
- To configure a Tomcat 6 server instance, it is necessary to modify these XML files.
- Files are:
  - server.xml
  - context.xml
  - web.xml
- Tomcat 6 looks for these configuration files in a specified configuration directory.
- This configuration directory is specified via an environment variable.
- Tomcat 6 first checks the $CATALINA_BASE (%CATALINA_BASE% on Windows) environment variable.

Configuration files and their roles

File Name	Description
`server.xml`	Primary configuration file for the Tomcat server components. This includes the configuration of Service, Connector, Engine, Realm, Valves, Hosts, and so on.
`context.xml`	Default version of the per-application configuration file for server components. Any components configured in this default file apply to all applications running on the server. An individual application can override the global configuration by defining its own `context.xml` file (placed in the `META-INF` directory of the application).
`web.xml`	Default version of the standard Java EE deployment descriptor for Web applications. This is used by Tomcat 6 for all automatically deployed Web applications, or applications without their own specific deployment descriptor. If a Web application has its own deployment descriptor, its content will always override the configuration settings specified in this default descriptor.

Mod_proxy in Apache

server.xml | httpd.conf

```
106  #LoadModule mem_cache_module modules/mod_mem_cache.so
107  LoadModule mime_module modules/mod_mime.so
108  #LoadModule mime_magic_module modules/mod_mime_magic.so
109  LoadModule negotiation_module modules/mod_negotiation.so
110  #LoadModule proxy_module modules/mod_proxy.so
111  #LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
112  #LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
113  #LoadModule proxy_connect_module modules/mod_proxy_connect.so
```

```
<Proxy balancer://testcluster stickysession=JSESSIONID>
BalancerMember ajp://127.0.0.1:8009 min=10 max=100 route=node1 loadfactor=1
BalancerMember ajp://127.0.0.1:8019 min=20 max=200 route=node2 loadfactor=1
</Proxy>

ProxyPass /examples balancer://testcluster/examples
```

- The ProxyPassReverse is used to change the headers sent by the app (appcluster) to Apache, before Apache sends it the browser. For example, if the app sits at http://localhost:9013/, and it tries to redirect the browser to, say, /new_location/, then it will respond with a redirect and location header of http://localhost:9013/new_location/, and Apache will take this and send it off to the browser. Problem is, the browser (assuming it's somewhere else) then tries to send a request to http://localhost:9013/new_location/, and gets an error.
- What ProxyPassReverse does is intercepts those headers, and rewrites them so that they match what the Apache server that's doing the proxying looks like. So if my apache server is hosting http://myhost.com/ and I have a ProxyPass that points / to http://localhost:9013/App, if the application sitting at localhost:9013 returns a redirect to http://localhost:9013/App/new_location/, I'll need to use ProxyPassReverse so that it gets rewritten to http://myhost.com/new_location/ by Apache before sending the request back to the browser.
- If you aren't issuing redirects, it's not going to be an issue, but it doesn't hurt to have it there in case a 301/302 redirect is returned. As far as mod_rewrite, the RewriteRule applies to the request going to the App, and not the response coming from the App. So they are mutually exclusive events.

Realms

- Overview
  - What is a Realm?
  - Configuring a Realm
- Common Features
  - Digested Passwords
  - Example Application
  - Manager Application
  - Realm Logging
- Standard Realm Implementations
  - JDBCRealm
  - DataSourceRealm
  - JNDIRealm
  - UserDatabaseRealm
  - MemoryRealm
  - JAASRealm
  - CombinedRealm
  - LockOutReal

What is a Realm?

- A **Realm** is a "database" of usernames and passwords that identify valid users of a web application (or set of web applications), plus an enumeration of the list of *roles* associated with each valid user.
- Tomcat 6 defines a Java interface (**org.apache.catalina.Realm**) that can be implemented by "plug in" components to establish this connection.

Monitor Tomcat with jconsole

- Edit CATALINA_HOME/bin/catalina.sh to include

```
if [ -z "$CATALINA_BASE" ] ; then
  CATALINA_BASE="$CATALINA_HOME"
fi
CATALINA_OPTS="-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=9004
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false"

if [ -z "$CATALINA_TMPDIR" ] ; then
  # Define the java.io.tmpdir to use for Catalina
  CATALINA_TMPDIR="$CATALINA_BASE"/temp
fi
```

Start jconsole and connect

- Go to a terminal window
- Type **jconsole**
- Select remote process
- Enter tomcat-host:jmx-port you enterd in the previous slide
- Done.