ApacheCon Europe 2000

Session T06:

Migrating Apache JServ Applications To Tomcat

24 October 2000 -- 14h00-15h00

Craig R. McClanahan

Sun Microsystems, Inc.

Craig.McClanahan@eng.sun.com

Prev Page Session Outline Next Page

- New Deployment Options
- Servlet API Versions
  - Changes From 2.0 to 2.1
  - Changes From 2.1 to 2.2
  - Changes from 2.2 to 2.3
- Web Applications
- Server Configuration
- References

Prev Page Session T06: Migrating Apache JServ Applications to Tomcat Next Page

Prev Page New Deployment Options Next Page

- Apache JServ has standard support only for running with Apache
- Tomcat can run with the following web servers:
  - Apache (1.3.x now, 2.x soon)
  - AOLServer
  - iPlanet (Netscape) Web Server
  - Microsoft IIS and PWS
- In-process versus out-of-process deployment
- Tomcat can run stand alone

Prev Page Session T06: Migrating Apache JServ Applications to Tomcat Next Page

Prev Page

Servlet API Versions

Next Page

- Apache JServ is based on **Version 2.0** of the Servlet API
- **Version 2.1** (11/98) - Significant API changes
- **Version 2.2** (08/99) - Introduces Web Applications
- **Version 2.3** (08/00 public draft) - Additional API enhancements
- Jason Hunter's *JavaWorld* articles (links on the References slide at the end of the presentation) on version 2.1 and 2.2 should be considered required reading for anyone planning a migration

Servlet API Version 2.1

Final Release - November, 1998

- Servlet context attributes for sharing global application objects:
  - ServletContext.getAttribute(String name)
  - ServletContext.getAttributeNames()
  - ServletContext.removeAttribute(String name)
  - ServletContext.setAttribute(String name, Object value)

```
In the init() method of a startup
servlet:

  ConnectionPool myPool =
     new ConnectionPool(...);
  ServletContext sc = getServletContext();
  sc.setAttribute("pool", myPool);

In a servlet that needs to use the
connection pool:

  ServletContext sc = getServletContext();
  ConnectionPool myPool = (ConnectionPool)
    sc.getAttribute("pool");
  Connection conn = myPool.allocate();
  ...
  conn.close();
```

- Servlet request attributes for passing per-request state information:
  - ServletRequest.getAttribute(String name) was already present
  - ServletRequest.getAttributeNames()
  - ServletRequest.removeAttribute(String name)
  - ServletRequest.setAttribute(String name, Object value)

```
In the servlet that processes a request and
forwards control to a presentation servlet:

  Hashtable hashtable =
    database.getResults(...);
  request.setAttribute("results", hashtable);

In the servlet that will use the results to
create the HTML response to the user:

  Hashtable hashtable = (Hashtable)
    request.getAttribute("results");
```

Prev Page Version 2.1 - Major Additions Next Page

- Servlet context resource abstraction:
  - ServletContext.getResource(String path)
  - ServletContext.getResourceAsStream(String path)

```
To load a properties file from within the
document root of this application:

  String name = "/myproperties.properties";
  ServletContext sc = getServletContext();
  InputStream is =
    sc.getResourceAsStream(name);
  Properties props = new Properties();
  props.load(is);
  is.close();
```

- RequestDispatcher for programmatic forwarding and nesting:
  - ServletContext.getResourceDispatcher(String path)
  - RequestDispatcher.forward(ServletRequest request, ServletResponse response)
  - RequestDispatcher.include(ServletRequest request, ServletResponse response)

```
In a business servlet that performs database
lookups:
    Hashtable hashtable = new Hashtable();
    hashtable.put(...); // Save results we need
    request.setAttribute("results", hashtable);
    RequestDispatcher rd = getServletContext().
      getRequestDispatcher("/servlet/showResults");
    rd.forward(request, response);

In a portal servlet that assembles its total
response from individual dynamic components:
    RequestDispatcher rd = getServletContext().
      getRequestDispatcher("/servlet/NewsTicker");
    rd.include(request, response);
```

Prev Page Version 2.1 - Deprecations Next Page

- The following methods have been deprecated and return `null` and empty `Enumerations`:
  - ServletContext.getServlet(String name)
  - ServletContext.getServletNames()
  - ServletContext.getServlets()
- How do you deal with this change? It depends on what you were using `getServlet()` for:
  - **Shared data access** - Extract the data into new Java objects stored as Servlet Context attributes
  - **Access to shared methods** - Extract the methods into new Java objects stored as Servlet Context attributes
  - **Merged input** - Use `RequestDispatcher.include()` for programmatic server-side includes
  - **Filtered output** - Pass data as request attributes to the next stage via `RequestDispatcher.forward()`

Prev Page Version 2.1 - Deprecations Next Page

- The following method has been deprectated and returns `null`:
  - HttpSession.getSessionContext()
- In addition, the entire `HttpSessionContext` class has been deprecated
- How do you deal with this change?
  - Create a class that implements `HttpSessionBindingListener`
  - Make sure an object of this class is added to every new session
  - When `valueBound()` is called, add the corresponding session to a private collection, perhaps stored as a servlet context attribute
  - When `valueUnbound()` is called, remove the corresponding session from the private collection
- In version 2.3, you can also use an Application Events Listener to manage your private session collection

Prev Page Session T06: Migrating Apache JServ Applications to Tomcat Next Page

Prev Page Version 2.1 - Minor Changes Next Page

- Minor API Additions:
  - Logging improvements
  - Multi-valued request parameters
  - Servlet API version values
  - ServletException takes an optional root cause argument
  - Programmatic control over session timeouts
- Minor API Changes:
  - Spelling changes (`Url` --> `URL`)
  - Session attribute methods (`getValue()` --> `getAttribute()`) for consistency

Prev Page Session T06: Migrating Apache JServ Applications to Tomcat Next Page

Prev Page Servlet API Version 2.2 Next Page

Servlet API Version 2.2

Final Release - August, 1999

Prev Page

Version 2.2 - Additions

Next Page

- ServletContext initialization parameters
  - ServletContext.getInitParameter(String name)
  - ServletContext.getInitParameterNames()
- Internationalization improvements
  - ServletRequest.getLocale()
  - ServletRequest.getLocales()
  - ServletResponse.getLocale()
  - ServletResponse.setLocale()
- Servlet response buffering support
  - ServletResponse.flushBuffer()
  - ServletResponse.getBufferSize()
  - ServletResponse.isCommitted()
  - ServletResponse.reset()
  - ServletResponse.setBufferSize()

Prev Page

Version 2.2 - Additions

Next Page

- Additional support for request dispatchers
  - ServletConfig.getServletName()
  - ServletContext.getNamedDispatcher(String name)
  - ServletRequest.getRequestDispatcher(String path)
- Additional request properties
  - ServletRequest.isSecure()
  - HttpServletRequest.getContextPath()
  - HttpServletRequest.getHeaders(String name)
  - HttpServletRequest.getUserPrincipal()
  - HttpServletRequest.isUserInRole(String role)
- Additional response methods
  - HttpServletResponse.addDateHeader(String name, long value)
  - HttpServletResponse.addHeader(String name, String value)
  - HttpServletResponse.addIntHeader(String name, int value)

Servlet API Version 2.3

Initial Public Draft - August, 2000

Prev Page

Version 2.3 - Additions

Next Page

- **WARNING** - Based on Public Draft 1 - Changes are possible before final version is released
- Requires a Java2 (JDK 1.2 or later) platform to run on
- Internationalization improvements
  - Defined rules for containers to calculate return values for `ServletRequest.getLocale()/getLocales()`
  - ServletRequest.setCharacterEncoding(String encoding)
- Request dispatcher improvements
  - New request and response wrapper classes are available
  - Arguments to `forward()` and `include()` can be the original objects, or wrappers around the original objects
  - Query string parameters on the request dispatcher path are aggregated with request parameters from the original request
- Additional request methods
  - ServletRequest.getParameterMap()
  - HttpServletRequest.getRequestURL()
- HttpUtils class is deprecated (but still available)

Prev Page

Next Page

Prev Page

Next Page

- A **Web Application** corresponds to a **ServletContext**, and will be discussed in more detail later
- You can register event listener classes that are notified on the following events related to the entire application (`ServletContext`):
  - Application startup (prior to the first request)
  - Application shutdown (after the last request)
  - Added, removed, or replaced servlet context attributes
- Application events are useful for initializing shared resources such as connection pools at application start time
- You can register event listener classes that are notified on the following events related to individual `HttpSession` instances:
  - Session creation
  - Session invalidation or timeout
  - Added, removed, or replaced session attributes

Prev Page

Next Page

Prev Page

Version 2.3 - Servlet Filtering

Next Page

- A **Filter** is an application-defined component that participates in request processing before and after the servlet that is ultimately invoked
  - Logging, auditing, and performance measurement
- Filters are mapped to request URI patterns (such as `/mydir/*` or `*.xml`) and/or individual servlets
- Filters can optionally complete the response and return, without invoking the next filter or the servlet mapped to this request URI
  - Application-level authentication
  - Remote IP address filtering
- Filters can optionally wrap the request and/or response objects passed on to the next filter or the servlet mapped to this request URI
  - Content transformation (XSLT) or tokenization (SAX/DOM)
  - Data compression and decompression
  - Encryption and decryption

Prev Page

Session T06: Migrating Apache JServ Applications to Tomcat

Next Page

- An example performance monitoring filter

```
public class ExampleFilter implements Filter {

  private FilterConfig filterConfig = null;
  public FilterConfig getFilterConfig() {
    return (this.filterConfig);
  }
  public void setFilterConfig
   (FilterConfig filterConfig) {
    this.filterConfig = filterConfig;
  }

  public void doFilter(ServletRequest request,
   ServletResponse response) throws
   IOException, ServletException {
    long startTime =
     System.currentTimeMills();
    Filter next = filterConfig.getNext();
    next.doFilter(request, response);
    long stopTime =
     System.currentTimeMillis();
    ServletContext sc =
     filterConfig.getServletConfig();
    sc.log("Request took " +
     (stopTime - startTime) +
     " milliseconds");
  }

}
```

Introduction To Web Applications

Introduced in Servlet API Version 2.2

Prev Page

Next Page

- A **Web Application** is a collection of servlets, JavaServer Pages (JSPs), HTML pages, Java classes and JAR files, and other resources that can be bundled and run on multiple containers from multiple vendors.
- Standard deployment format - the **Web Application Archive** (WAR) file
- Standard configuration mechanism - the **deployment descriptor** (web.xml) file
- Attached to a "context path" (request URI prefix), such as `/catalog`, by the servlet container administrator
- If the context path is "" (empty string), this web application represents an entire web site
- Corresponds one-to-one with a **ServletContext**

Prev Page

Next Page

Prev Page

Next Page

- A **Web Application Archive** (WAR) file is a JAR file with contents in a standardized directory layout:
  - `/WEB-INF/*` - Directory containing configuration files and resources for this application, not available to web clients
  - `/WEB-INF/web.xml` - The **deployment descriptor** for this web application
  - `/WEB-INF/classes/` - Java classes (including servlets) and resources that are not packaged in JAR files
  - `/WEB-INF/lib/*.jar` - Java classes (including servlets) and resources that have been packaged in JAR files
  - `/*` - Static content (such as HTML pages and images) that are visible at the "document root" of this application
- A typical web application will have a "welcome" file named `/index.html` or `/index.jsp` in the top level directory
- The servlet container **MUST** accept a WAR file to be deployed, and **MAY** expand it into the corresponding unpacked directory structure for execution (Tomcat currently does this)

Prev Page

Next Page

- Servlet containers now support the concept of **container managed security** so that applications do not need to deal with this issue
- Interface to user database is container specific (in Tomcat, it is a pluggable interface called a `Realm`)
- Four standard mechanisms for authenticating users
  - **HTTP BASIC Authentication** - Pop-up username/password prompt like protected areas unde Apache
  - **HTTP DIGEST Authentication** - Similar to BASIC Authentication but username and password are encrypted before transmission (requires Tomcat 4.0)
  - **Form Based Authentication** - Application provides login and error pages to be used by the container
  - **SSL Client Certificate Authentication** - Use client certificates provided on SSL connections (requires Tomcat 4.0)

- Subsets of the URI space for this application are protected by **security constraints** that can limit access based on any of the following criteria
  - Authenticated user
  - Authenticated user who possesses a specific **role**
  - Specified HTTP method (GET, POST, PUT, ...)
  - Require secure (SSL) connection
- Roles can have arbitrary meaning, but are typically mapped to the **Group** concept in the underlying security technology
- In addition to container-managed access to particular URIs, servlets can make security-related decisions about what content to present
  - **HttpServletRequest.getRemoteUser()** - Returns the username for the authenticated user
  - **HttpServletRequest.getUserPrincipal()** - Returns the `java.security.Principal` object for the authenticated user
  - **HttpServletRequest.isUserInRole(String role)** - Returns `true` if the authenticated user is associated with (possesses) the specified role

Prev Page

Next Page

- The servlet container selects a servlet to process each request by parsing the **Request URI** according to a standard set of rules
- First, it matches the **beginning** of the Request URI against the **context paths** of all registered applications
  - The longest possible match wins in case of overlaps
  - If there is no match based on context path, the "root" application (the one mapped to a context path of ""), if any, is selected
- Next, the container strips off the context path, and compares the remainder of the Request URI against the **Servlet Mapping** rules for the selected web application, as follows, until a match is found:
  - Check for an exact match against each of the registered servlet mappings ("exact match")
  - Check mappings that end with "/*" and match up to that point ("path match")
  - Check for mappings of the form "*.xxx", and match the **last** path component of the request URI ("extension match")
  - If no other match is found, give this request to the default servlet (if any) for this application, indicated by a mapping to "/" ("default match")

Prev Page

Next Page

Prev Page

Web Apps - Request Mapping

Next Page

- Example request mapping patterns and corresponding results

Request Mapping Patterns	
Path Pattern	**Servlet To Select**
/foo/bar/*	servlet1
/baz/*	servlet2
/catalog	servlet3
*.bop	servlet4

NOTE: The match for `/catalog` is exact because
no wildcard is specified

Request Mapping Results	
Incoming Path	**Selected Servlet**
/foo/bar/index.html	servlet1
/foo/bar/index.bop	servlet1
/baz	servlet2
/baz/index.html	servlet2
/catalog	servlet3
/catalog/index.html	"Default" servlet
/catalog/racecar.bop	servlet4
/index.bop	servlet4

Prev Page

Session T06: Migrating Apache JServ Applications to Tomcat

Next Page

- As part of the request mapping process, a servlet container divides the request URI into four portions, and provides access to those portions through method calls on **HttpServletRequest**:
  - **getContextPath()** - Returns the context path of the web application processing this request ("" for the root context)
  - **getServletPath()** - Returns the portion of the request URI, starting with the "/" immediately after the context path, that was used to select the servlet to process this request
  - **getPathInfo()** - Returns any portion of the request URI that is after the servlet path, and before the query string (if any)
  - **getQueryString()** - Returns any portion of the request URI after an unencoded "?" character (which may also be parsed into request parameters if the syntax is correct)

Prev Page

Request Mapping Results

Next Page

- Each type of request mapping described on the preceding slides results in the following values for servlet path and path info:
  - **Exact Match** - Servlet path will contain the entire request URI after the context path and before any query string, and path info will be `null`
  - **Path Match** - Servlet path will contain the same characters as the path pattern minus the trailing "/*", and path info will be anything after that or `null`
  - **Extenstion Match** - Servlet path will contain the entire request URI after the context path and before any query string, and path info will be `null` (because extension matching works only on the last path element)
  - **Default Match** - Servlet path will contain the entire request URI after the context path and before any query string, and path info will be `null`

Prev Page

Next Page

Prev Page

Tomcat Standard Mappings

Next Page

- Tomcat defines several default request mapping patterns (not required by the Servlet API Specification) to support standard facilities:
  - **/servlet/*** - Executes the "invoker" servlet, that allows you to execute servlets by their class name even if they are not registered in the configuration file
  - ***.jsp** - JSP pages are processed by the Jasper servlet that compiles and/or executes them
  - **/** - The default servlet for each web application (unless changed by the user) uses the specified path to locate a static resource (typically an HTML page or image file) that is returned as the response to this request
- How these mappings are defined, and how they can be changed, will be discussed in the Configuration section

Prev Page

Session T06: Migrating Apache JServ Applications to Tomcat

Next Page

Prev Page Configuration Next Page

Configuration

Web Application Configuration (Portable)

Tomcat Configuration (Specific)

Prev Page

Web Application Configuration

Next Page

- Web applications are configured using the **deployment descriptor** (/WEB-INF/web.xml) file, using XML syntax
- The document type description (DTD) for the deployment descriptor is listed in the Servlet API Specification, and included with Tomcat in the `$TOMCAT_HOME/conf` directory
- The deployment descriptor you prepare is "validated" against this DTD, so you must obey the rules described there - most easily accomplished by using an XML-aware editor
- The following configuration elements (among others) are included in the deployment descriptor:
  - Servlet context initialization parameters
  - Servlets (name, Java class, and initialization parameters)
  - Servlet mappings (request path patterns and the corresponding servlets to be executed)
  - Welcome file list

Prev Page

Session T06: Migrating Apache JServ Applications to Tomcat

Next Page

- The deployment descriptor follows the standard XML approach of declaring the DTD it conforms to, and uses the `web-app` top-level element

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>

  <!-- Configuration elements go here -->

</web-app>
```

- Examples for each configuration element will only show that element
- The order of elements we are walking through is the order required by the DTD (all elements of each type appear together)

- Context initialization parameters should be used for values that apply to the application as a whole, or to configure global resources:

```
<context-param>
  <param-name>dbpool.driver</param-name>
  <param-value>oracle.jdbc.driver.OracleDriver</param-value>
</context-param>

<context-param>
  <param-name>dbpool.url</param-name>
  <param-value>jdbc:oracle:thin:@dbserver:1521:ORCL</param-value>
</context-param>
```

- Servlets can read context initialization parameters like this:

```
ServletContext sc = getServletContext();
String className = sc.getInitParameter("dbpool.driver");
```

- Servlets and their initialization parameters are defined together:

```
<servlet>
  <servlet-name>myservlet</servlet-name>
  <servlet-class>com.mycompany.mypackage.MyServlet</servlet-class>
  <init-param>
    <param-name>configprops</param-name>
    <param-value>/WEB-INF/config.properties</param-value>
  </init-param>
  <!-- Ask the container to load this servlet immediately -->
  <load-on-startup>3</load-on-startup>
</servlet>
```

- Access to these parameters is common in the `init()` method:

```
String resourceName =
  getServletConfig().getInitParameter("configprops");
InputStream stream =
  getServletContext().getResourceAsStream(resourceName);
Properties configProps = new Properties();
configProps.load(stream);
stream.close();
```

Prev Page Servlet Mappings Next Page

- Next, we register request mapping patterns to map requests to the servlets that will be used to process them:

```
<servlet-mapping>
  <servlet-name>myservlet</servlet-name>
  <url-pattern>/store/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>cocoon</servlet-name>
  <url-pattern>*.xml</url-pattern>
</servlet-mapping>
```

- You must have defined each servlet named here with a `<servlet>` element previously
- For extra credit, which of the two servlets would handle a request for `/store/index.xml` given these mappings?

Prev Page Session T06: Migrating Apache JServ Applications to Tomcat Next Page

- You can specify the name of the file (or resource) that will be served when the request URI specifies a "directory" rather than a "file":

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

- This is similar in capability to Apache's `DirectoryIndex` configuration directive

- The deployment descriptor (web.xml) is portable to all servlet containers -- the following information is specific to Tomcat 3.2b4
- In the following slides, `$TOMCAT_HOME` represents the directory into which you have installed Tomcat
- We will focus on configuration necessary to set up Tomcat for stand alone use, and later extend it for use behind Apache
- In the simplest case, no configuration at all is required -- simply place your web application archive (WAR) file into the `$TOMCAT_HOME/webapps` directory and restart Tomcat
- Tomcat automatically expands the WAR file into an unpacked directory and installs it
- If you wish to update this application later, be sure you replace the WAR file **AND** delete the old directory and all of its contents

Prev Page Tomcat Server Configuration File Next Page

- Tomcat 3.2b4 configures itself based on the contents of `$TOMCAT_HOME/conf/server.xml`
- We will not cover all of the options in this file - our focus will be on setting up a web application with non-default properties
- A web application is configured by a `<Context>` element (optionally nested in a `<Host>` element for apps specific to a virtual host):

```
<Host name="www.mycompany.com">
  <Context path="/examples"
           docBase="webapps/examples"
           debug="0"
           reloadable="true"/>
</Host>
```

Prev Page Session T06: Migrating Apache JServ Applications to Tomcat Next Page

Prev Page Tomcat Server Configuration File Next Page

- The `<Context>` element (in Tomcat 3.2b4) recognizes the following XML attribute names:

Attribute	Description
debug	Select the amount of debugging detail information to be logged (higher numbers generally mean more detail)
docBase	Base directory of this web application - if relative, it is resolved from the $TOMCAT_HOME directory
path	Context path for this web application - the empty string identifies the default (root) application
reloadable	Set to `true` to enable application reloading if a class is changed
trusted	Set to `true` if this application should have access to Tomcat internals - normally only required for the administrative application

- Tomcat configuration for stand alone use is fairly simple - unless you really need Apache for functionality or performance, I suggest you consider deploying it this way
- For use with Apache, you must configure the `mod_jserv` or `mod_jk` module in Apache, as described in the documentation
- We will focus on the more useful case for Apache JServ users, which uses `mod_jserv`
- When Tomcat first starts, it generates a model set of configuration directives to be included in `httpd.conf`, in the file `$TOMCAT_HOME/conf/tomcat-apache.conf`
- Make a copy of this file for customization, and add an include directive to your Apache configuration file (httpd.conf):

```
Include /tomcat-home/my-tomcat-apache.conf
```

- The `tomcat-apache.conf` file starts with the following directives:

```
LoadModule jserv_module libexec/mod_jserv.so
ApJServManual on
ApJServDefaultProtocol ajpv12
ApJServSecretKey DISABLED
ApJServMountCopy on
ApJServLogLevel notice

ApJServDefaultPort 8007

AddType text/jsp .jsp
AddHandler jserv-servlet .jsp
```

- From these settings, we can see that the AJPV12 protocol has been selected, and that JSP pages (filenames that end with `.jsp`) will be forwarded to Tomcat for processing

- Next, a set of directives for each configured web application is created, with the following goals:
  - Map the context path of this application (in Apache's URI namespace) to the Tomcat document root
  - Pass all requests for JSP pages (filename extension ".jsp") to Tomcat
  - Pass all requests matching `/servlet/*` to Tomcat
  - Disallow clients from requesting files in the `WEB-INF` or `META-INF` directories of the web application
  - Handle all other requests (typically requests for static HTML files and images) within Apache itself
- In Tomcat 3.2b4, the generated directives are **not** based on the contents of your deployment descriptor (web.xml) file - only on the context path and document root

- Here are the directives generated for the standard "/examples" web application that ships with Tomcat:

```
Alias /examples "/tomcat-home/webapps/examples"
<Directory "/tomcat-home/webapps/examples">
  Options Indexes FollowSymLinks
</Directory>
ApJServMount /examples/servlet /examples
<Location "/examples/WEB-INF/">
  AllowOverride None
  deny from all
<Location>
<Location "/examples/META-INF/">
  AllowOverride None
  deny from all
<Location>
```

- A special directive is also included to make `/servlet` requests in the Apache server root call Tomcat's root application:

```
ApJServMount /servlet /ROOT
```

- How can we customize this configuration to accomplish more specialized goals? We will describe several common scenarios
- To use a Tomcat instance on a different server (default is localhost) or port (default is 8007):

```
ApJServMount /examples/servlet ajpv12://myhost:8008/examples
```

- Make Apache recognize session identifiers when using URL rewriting (you **must** have `mod_rewrite` configured)

```
RewriteEngine On
RewriteRule ^(/.*;jsessionid=.*)$ $1 [T=jserv-servlet]
```

- Support a JSP page as the directory index / welcome file:

```
<Directory "/tomcat-home/webapps/examples">
  Options Indexes FollowSymLinks
  DirectoryIndex index.jsp
</Directory>
```

- Forward requests for "Exact Match" mapping (/myservlet)

```
ApJServMount /examples/myservlet /examples
```

- Forward requests for "Path Match" mapping (/catalog/*)

```
ApJServMount /examples/catalog /examples
```

- Forward requests for "Extension Match" mapping (*.xml)

```
<LocationMatch "/examples/*.xml">
  SetHandler jserv-servlet
</LocationMatch>
```

- Forward **all** requests (even for static resources) within a particular web application

```
ApJServMount /cocoon /cocoon
```

- You can combine customizations as necessary - for instance, to run the Struts framework example application (http://jakarta.apache.org/struts):

```
Alias /struts-example "/tomcat-home/webapps/struts-example"
<LocationMatch "/struts-example/*.do">
  SetHandler jserv-servlet
</LocationMatch>
<Directory "/tomcat-home/webapps/struts-example">
    Options Indexes FollowSymLinks
    DirectoryIndex index.jsp
</Directory>
# ApJServMount /struts-example/servlet /struts-example
<Location "/struts-example/WEB-INF/">
    AllowOverride None
    deny from all
</Location>
<Location "/struts-example/META-INF/">
    AllowOverride None
    deny from all
</Location>
```

- As a final example of simple configuration, let's enable the use of `/servlets` instead of `/servlet` for accessing your servlets
- In your customized version of `tomcat-apache.conf`, add the following directive so that Apache will forward these requests:

```
ApJServMount /servlets /ROOT
```

- For Tomcat 3.1 or 4.0, add the following mapping to the deployment descriptor for the ROOT application ($TOMCAT_HOME/webapps/ROOT/WEB-INF/web.xml):

```
<servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlets/*<url-pattern>
</servlet-mapping>
```

- For Tomcat 3.2, the matched prefix is hard coded in the source code (`org.apache.tomcat.request.InvokerInterceptor`), so you have to change it there and rebuild Tomcat

Prev Page References Next Page

- Servlet API and JavaServer Pages (JSP) Specifications:
  - http://java.sun.com/products/servlet/download.html
  - http://java.sun.com/products/jsp/download.html
- Jason Hunter's JavaWorld Articles:
  - http://www.javaworld.com/jw-12-1998/jw-12-servletapi.html
  - http://www.javaworld.com/jw-10-1999/jw-10-servletapi.html
- Tomcat Project Page:
  - http://jakarta.apache.org/tomcat
- Apache JServ Project Page (historical documentation):
  - http://java.apache.org/jserv
- Author's Email Address:
  - Craig.McClanahan@eng.sun.com

Prev Page Session T06: Migrating Apache JServ Applications to Tomcat Next Page