

Advanced Topics for the Tomcat Servlet/JSP Reference Implementation



Noel J. Bergman

DevTech[®]





Session Overview

Tomcat is the official Reference Implementation of the JSP and Servlet specifications. The purpose for this session is to cover more advanced topics to help you deliver better performance, more reliable and more functional Web Apps.

Topics will include sessions that survive server reboots, the various Listener interfaces, handling additional HTTP features such as HTTP 304 response to reduce transfers and increase responsiveness, use of JNDI resources, Web server integration, and more.

PLEASE ASK QUESTIONS! 😊



Session Prerequisites

- Basic familiarity with Servlet and JSP specifications.
- Some experience with Tomcat and Apache HTTPd.



Configuration vs. Development

- Most of this presentation deals with development topics.
- However, developers can't expect most Web server administrators to know how to properly configure a web server to work with Tomcat. Nor can they expect web server administrators to know *anything* about configuring Tomcat or Web apps.
- The presentation covers some configuration areas to better help you:
 - Understand the configuration issues.
 - Communicate them to a Web server administrator
 - OK, **be** the Web server administrator if necessary.



Keys to Better Apps

- Look

- *"Use the Source, Duke."* There are behaviors whose implementation is unspecified. Use the source to see how best to tie into them.
- But plan for change. Internals can and will change!

- Listen

- The various Listener interfaces provide access to a lot of useful container behavior.

- Borrow

- Web applications are relatively immature compared to the experience within the web serving and database communities. Find best practices, and apply them.

Optimize Network/Server Use

- We can improve real and perceived performance by reducing the amount of transfer between the client and server without reducing the content.
 - Allow Client-side Caching (Conditional Get)
 - The client can cache resources for re-use so that it does not need to transfer resources that have not changed.
 - Not only reduced transfer size, but also reduced CPU.
 - Compression
 - We can compress the data shipped from the server to the client.
 - Significant reduction of transfer size.
 - CPU cost to compress, but balanced by less CPU cost to perform the transfer and pre-compression or caching.



Conditional Get

- Allow Client-side (and Proxy) Caching
 - Client requests resource
 - Server responds with resource and Last-Modified
 - Caching client requests with If-Modified-Since
 - Server response either with revised resource and new Last-Modified, or with HTTP 304 response code.
- A good web server will automatically handle Conditional Get for static resources, but we need to handle it for dynamic content.

Client-side Caching w/ Servlets

- Java Servlet (v2+) Specification provides `getLastModified` for servlets.
 - `long getLastModified(HttpServletRequest request)`
 - Returns milliseconds since 1JAN1970.
 - Default -1 → don't use Last-Modified.
- A servlet could look at timestamp(s) for the resource(s) making up the response, and derive the correct response.
- What about JSP pages?



Client-side Caching w/ JSP

- The JSP Specification disallows JSP pages to implement `getLastModified` as a servlet.
 - "A JSP page author may not (re)define servlet methods"
- We can get past this restriction, carefully, through the use of the `extends` attribute, and a carefully crafted implementation of `HttpJspPage`.

Conditional Get for JSP Pages

- Subclass Tomcat's `HttpJspBase` class to minimize impact with the container.
 - We don't want to implement everything, or interfere with internal optimizations. We just want to add Conditional Get behavior.
 - Implement Conditional Get behavior in the `service` method. Call `super.service` to do everything else.
- At the least, a page can be considered changed when its source changes, or one of its dependents changes.
 - Tomcat v4 and Tomcat v5 differ slightly:
 - Tomcat 4: `List getIncludes()`
 - Tomcat 5: `List getDependants()`
- Want to allow a page to tell us if it has changed.
- **Demo:** `LastModifiedJSP.java` and sample JSP pages.



Why Cache Dynamic Content?

- If the Web server is already handling static resources, and servlets & JSP pages are *dynamic* content, why bother?
- Dynamic is not the same as evanescent.
- Consider as examples:
 - Wiki
 - WebBoard
 - Blog
 - RSS Aggregator
 - Catalog
 - *etc.*
- Each is dynamic, but with reasonably significant valid cache lifetimes.



Compression

- Reduce Network Traffic with Compression
- Built into HTTP protocol
 - HTTP Request uses Accept-Encoding Header
 - HTTP Response uses Content-Encoding Header
- Supported by Apache HTTP Server
 - mod_gzip (Apache HTTP server v1.3)
 - mod_deflate (Apache HTTP server v2)
- Supported by Tomcat as well

Configuring Compression

- Supported by Apache HTTP Server
 - Automatic. Configure the web server, and let it handle compression for Tomcat.
 - Sample `gzip.conf` and `deflate.conf` config files on the Conference CD.
- Supported by Tomcat as well
 - Tomcat has a simple `CompressionFilter`
 - Configure `<Filter>` and `<Filter-Mapping>`
- Combine Conditional Get with a compression cache to optimize delivery to new clients.



Other Filter Applications

- We've just seen a filter used to apply compression, a data transform, on the response. There are many other uses for filters.
 - Performance Measurements
 - Hit Counter
 - Caching
 - Data Transformation
- Filters are useful and underutilized.



Filter Basics

- Use the Source.
 - Tomcat provides several example filters.
- Core of a filter:
 - ```
void doFilter(ServletRequest request,
 ServletResponse response,
 FilterChain chain)
 throws IOException, ServletException {
 // Do anything interesting with the Request
 chain.doFilter(request, response);
 // Do anything interesting with the Response
}
```
- Very simple, but very powerful.
- The compression filter acts on the response.
- Demo: sample filters



# Resilient URIs

---

- To quote Tim Berners-Lee, “Cool URIs don’t change.” – <http://www.w3.org/Provider/Style/URI.html>
- Just because you move some files, rename servlets, or change your technology doesn’t mean that the URIs should change.
- Few sites meet this ideal. We can do better.
- Don’t expose CGI, servlets, file extensions, or other implementation details in the URI.



# Valves

---

- Using a Filter or a Servlet to forward a request is not the same as the request appearing from the connector.
- Filters and Servlets have other limitations dictated by the specification, and their role.
- A Valve is Tomcat's internal filter-like mechanism.
- Tomcat-specific (see Session Title 😊), but is the only way to implement a "mod\_rewrite" within Tomcat, or do other very useful tricks.
- Demo: sample Valve



# Learning to Listen

---

- Listeners allow us to follow changes within the Servlet Container.
- Servlet 2.3 specification defines 6 Listener interfaces.
  - `ServletContextListener`
  - `ServletContextAttributeListener`
  - `HttpSessionListener`
  - `HttpSessionAttributeListener`
  - `HttpSessionBindingListener`
  - `HttpSessionActivationListener`
- Servlet 2.4 specification introduces 2 new ones.
  - `ServletRequestListener`
  - `ServletRequestAttributeListener`
- Although the Specification explains when the events occur, the best way to learn how to use them is to actually watch them happen.

■ Demo: `TomcatTestListener`





# JVM-spanning Sessions

---

- Sessions may span multiple instances of Tomcat across time. This may be due to load balancing or a persistent session across a server restart.
- The `HTTPSessionActivationListener` is used prior to serializing, and after de-serializing. It can be used to prepare/repair session content.
- The Servlet specification says that `HttpSession.setAttribute` **and** `putValue` **must throw** `IllegalArgumentException` **if the object provided is not** `Serializable` **or otherwise able to be migrated by the container.**

# Clustering/Load Balancing

- Tomcat with Apache 1.3 and mod\_jk worked with a Load Balancing worker, permitting multiple JVMs.
- Sessions could be locked to a specific JVM.
- With Tomcat 4 and 5, support for clustering is available.
  - Clustering for Tomcat 4: <http://cvs.apache.org/~fhanik/>
- Now that we have looked at Listeners and Valves, we have the background necessary to understand the new Tomcat clustering mechanism.
- I do *not*, however, plan to demonstrate clustering on my ThinkPad. 😊

# The `java:comp/env` JNDI Context

- A J2EE container makes a JNDI `InitialContext` available for components within the container. The `java:comp/env` context is the defined location.
- **Lookup:** `o = new InitialContext().lookup("java:comp/env/path");`
- J2EE defined types of JNDI entries:
  - `env-entry`
  - `ejb-ref`
  - `ejb-local-ref`
  - `resource-ref`
  - `resource-env-ref`
- New to Tomcat 5 (Servlet 2.4/J2EE 1.4)
  - `message-destination-ref`
- We'll look at a few of these.



# The `env-entry`

---

- A simple environment variable, JNDI-style
- Similar to `context-param`, but typed.
- Elements:
  - `<description>`
  - `<env-entry-name>`
  - `<env-entry-type>`
    - `String`, `Character`, `Byte`, `Short`, `Integer`, `Long`, `Boolean`, `Double`, or `Float`.
  - `<env-entry-value>`
- Each `Context.lookup("java:...")` call results in a semantically unique object.



# Objects in Context

---

- `resource-env-ref` is similar to `env-entry` except there is no value, and we specify an interface.
- Elements:
  - `<description>`
  - `<resource-env-ref-name>`
  - `<resource-env-ref-type>`
- Now we can access objects, such as a named bean or other resource, defined for our context.
- Resources are declared in `web.xml`, but the definition of them is server specific.



# Objects in Context

---

- `resource-ref` is similar to `resource-env-ref` except the name is defined to refer to a connection factory for connections of `res-type`.
- Elements:
  - `<description>`
  - `<res-ref-name>`
  - `<res-type>`
  - `<res-auth>`
  - `<res-sharing-scope>`
- `resource-env` was created as a simpler form of this element when authentication isn't necessary.



# Defining `java:comp/env` Context

---

- Resources are declared in `web.xml`. The definition of them is server specific.
- Tomcat defines resources within a `Context` element, or globally in the `Server` element.
- Resources are defined using
  - `Environment` – equivalent to `env-entry`
  - `Resource` – used to define *both* `resource-ref` and `resource-env-ref` entries
- Object factories are used to handle both of the resource entries.
- Common uses: JDBC DataSources (`/jdbc`) and JavaMail Session (`/mail`), custom resources.
- Demo: JNDI resources



# Apache Integration

---

- Why put Apache HTTP server in front of Tomcat?
  - Robustness.
    - httpd has had years of work to make it a non-stop server, and is designed with that focus. For example, worker processes can be periodically replaced with fresh ones. Tomcat lacks the infrastructure to do that by itself, although clustering provides some interesting possibilities.
  - Performance
    - Without `java.nio`, Tomcat uses one thread per connection.
    - Without `sendfile()`, Tomcat cannot achieve the same level of efficiency.
  - Flexibility
    - Run separate Tomcat instances configured and privileged as necessary, *e.g.*, as different users for different virtual hosts.

# Kernel Level Data Transfer

- Without `sendfile`
  - while there is data to be transferred
    - read data from source to a buffer
    - write data from buffer to destination
- With `sendfile`
  - `sendfile(dest, src, ...)`
  - No ring transitions, redundant data copying, JNI boundaries, *etc.*
  - OS specific, but APR provides a portable layer.

# Tomcat *via* Proxy

- Explicitly keep for Apache HTTP server
  - RewriteRule *regular-expression* "\$0" [L]
  - Might keep GIF, JPEG, HTML, and/or specific directories
- Forbid WEB-INF/
  - RewriteRule /WEB-INF/ "\$0" [NC,F,L]
- Give everything else to Tomcat
  - RewriteRule "^/(.\*)" "http://host:port/\$1" [P]
  - ProxyPassReverse / http://host:port/
- Could proxy specific resources. It all depends upon what you want Tomcat to handle.

# Tomcat *via* `mod_jk2`

- Similar in effect, but we explicitly map resources for Tomcat to handle.
- I use UNIX Domain Sockets instead of TCP/IP to connect with Tomcat.
  - Don't have to track list of sockets assigned to instances.
  - Easier to secure.
- Caveat: as this is written in August, there is a bit of an inconvenience dealing with server aliases. If that remains at the time of the conference, you'll see each mapping replicated for each alias.
- Demo: `httpd.conf` and `server.xml`

# “HEADLESS” Graphics on \*NIX

- Problem:
  - I wanted to use a graphical password challenge to forbid robots, but permit easy access to people.
  - I don't install a GUI on my servers. [Windows users: ignore this 😊]
- < JRE 1.4:
  - The machine must be running an X Server (may be Xvfb)
  - The user the servlet engine runs as must be able to communicate with the X Server
  - The DISPLAY environment variable must be passed to the servlet engine process.
- >=JRE 1.4: printing, 2D, images enabled
  - `-Djava.awt.headless=true`
  - <http://java.sun.com/j2se/1.4.2/docs/guide/awt/AWTChanges.html#headless>



# Authentication

---

- You will likely want to use the authentication provided by Tomcat, especially if you are not authenticating in a front-end Web server.
- Declarative Security Constraints
  - XML analogue to those in Apache HTTP server
  - Container managed authentication
  - Authentication plug-in called a `Realm`.
- Optional User-provided Authentication Form
  - Login Form

```
<form method="POST" action="j_security_check">
<input type="text" name="j_username">
<input type="password" name="j_password">
</form>
```
- Demo: Access control to a URL



# Related Sessions

---

- “What’s New in the Servlet and JSP Specifications” (Bryan Basham)
- “JavaServer Pages and the Expression Language” (Bryan Basham)
- “Localizing and Customizing JavaServer Pages” (Paul Tremblett)
- “Portlets (JSR-168)” (Dave Landers)
- “Struts Controller in Action” (Gary Ashley Jr.)
- “Struts View Assembly and Validation” (Gary Ashley Jr.)
- “Tiles: Creating a Highly Flexible Dynamic Website” (Hermod Opstvedt)



# Wrap-up

---

- Are there any questions?
- Please remember to turn in your speaker evaluation forms.
- Thank you for coming. I hope that you've enjoyed the session.



# Links

---

- “Writing Servlet Filters”  
[http://javaboutique.internet.com/tutorials/Servlet\\_Filters/](http://javaboutique.internet.com/tutorials/Servlet_Filters/)
- Pre-compile JSP Pages  
<http://cvs.apache.org/~fhanik/precompile.html>
- Useful Things To Know About Compiling Pages  
<http://jakarta.apache.org/tomcat/tomcat-4.1-doc/jasper-howto.html>
- Tomcat Clustering / Load Balancing  
<http://cvs.apache.org/~fhanik/>
- A Collection of Useful Tomcat Articles  
<http://www.apache.org/~jefft/forrest/samples/wikirenderer-site/wiki/ApacheModProxy.html>