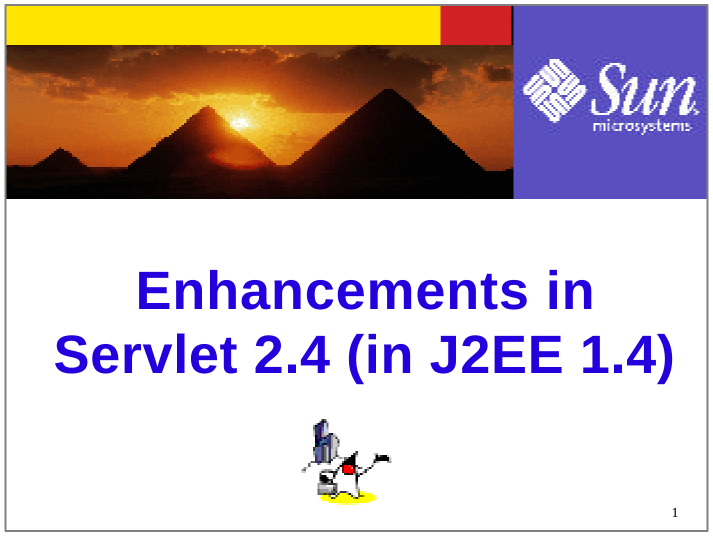


Enhancements in Servlet 2.4 (in J2EE 1.4)

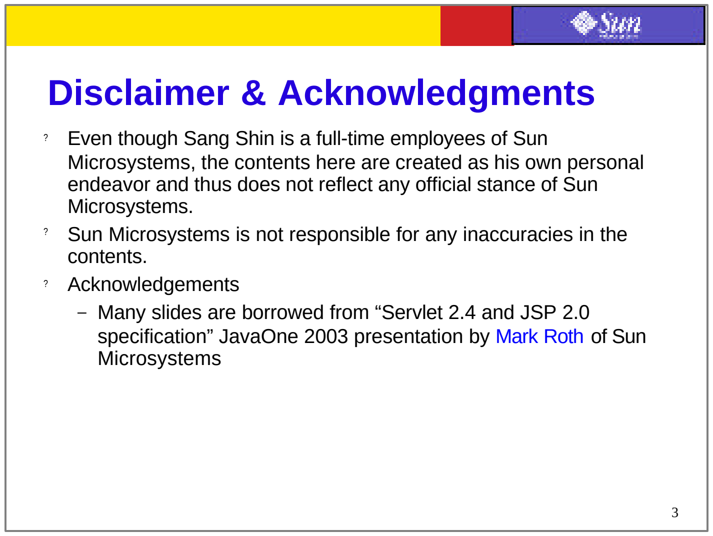
1

Let's talk about enhancements made in Servlet 2.4. I must say upfront, the things that are added to 2.4 are not as significant as the ones added in Servlet 2.3. But there are a few things you should be aware of.

Sang Shin

sang.shin@sun.com
www.javapassion.com
Java™ Technology Evangelist
Sun Microsystems, Inc.

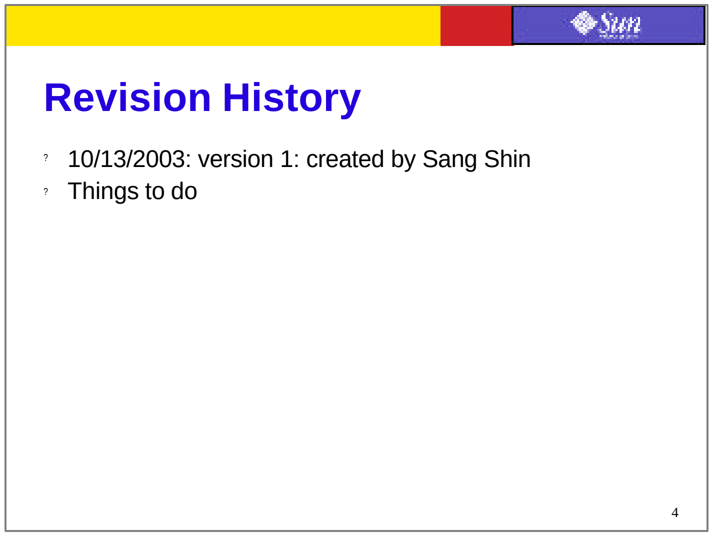
2



Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employees of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents.
- Acknowledgements
  - Many slides are borrowed from “Servlet 2.4 and JSP 2.0 specification” JavaOne 2003 presentation by Mark Roth of Sun Microsystems

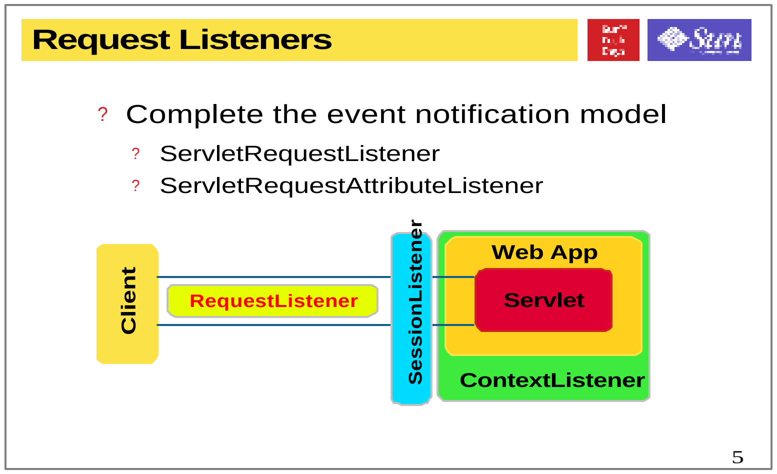
3



Revision History

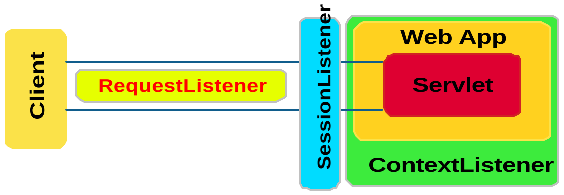
- 10/13/2003: version 1: created by Sang Shin
- Things to do

4



Request Listeners

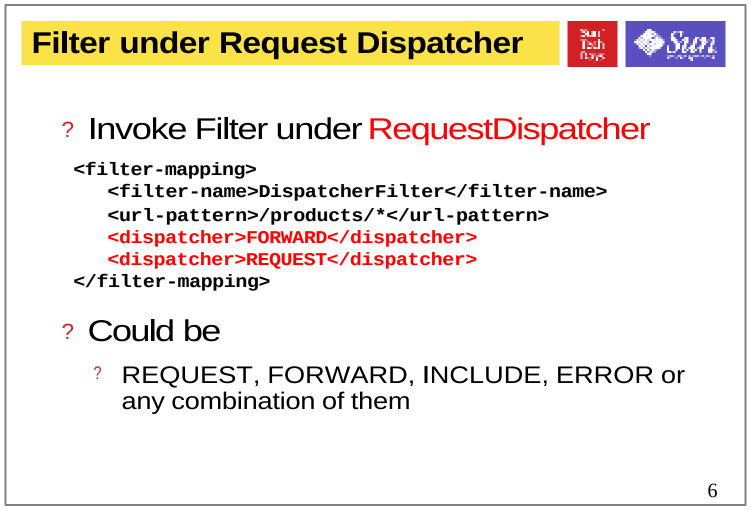
? Complete the event notification model
  - ? ServletRequestListener
  - ? ServletRequestAttributeListener



5

Servlet 2.3, which is a pervious version of Servlet, introduced the idea of context and session listeners, classes that could observe when a context or session was initialized or about to be destroyed, and when attributes were added or removed to the context or session. Servlet 2.4 expands the model to add request listeners, allowing developers (or more likely tool vendors) to observe as requests are created and destroyed, and as attributes are added and removed from a request.

So now there could be 3 different listeners in your web-tier application, context listener, sessionlistener, and now servletrequest listener.



Filter under Request Dispatcher

? Invoke Filter under RequestDispatcher

```
<filter-mapping>
   <filter-name>DispatcherFilter</filter-name>
   <url-pattern>/products/*</url-pattern>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

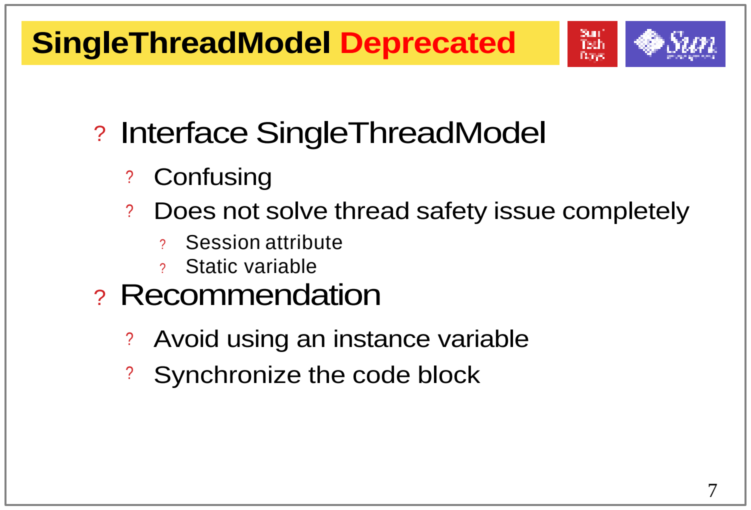
? Could be

  ? REQUEST, FORWARD, INCLUDE, ERROR or any combination of them

One of the ambiguous areas in previous version of Servlet is in regard to the interaction between the RequestDispatcher and filters. Should filters invoke for forwarded requests? Included requests? What about for URIs invoked via the <error-page> mechanism? Before Servlet 2.4, these questions were left as open issues.

Now Servlet 2.4 makes it a developer's choice. There's a new <dispatcher> element in the deployment descriptor with possible values REQUEST, FORWARD, INCLUDE, and ERROR. You can add any number of <dispatcher> entries to a <filter-mapping> like above.

This example indicates the filter should be applied to requests directly from the client as well as forward requests. Adding the INCLUDE and ERROR values also indicates that the filter should additionally be applied for include requests and <error-page> requests. Mix and match for what you want. If you don't specify any <dispatcher> elements, the default is REQUEST.



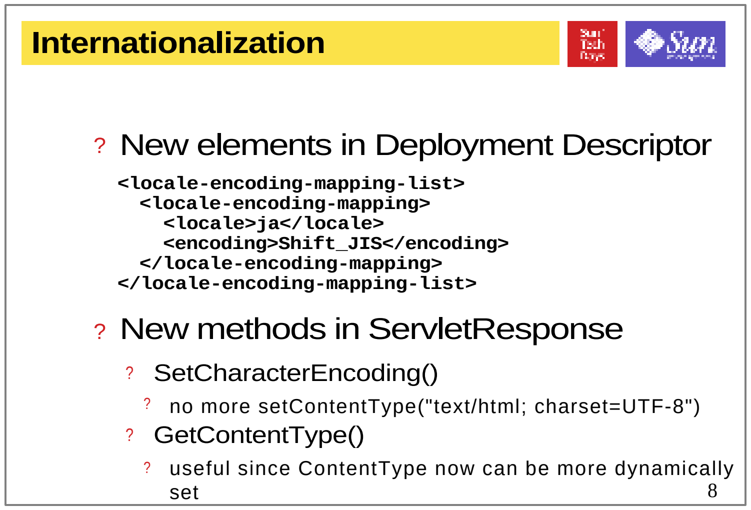
SingleThreadModel Deprecated

- Interface SingleThreadModel
  - Confusing
  - Does not solve thread safety issue completely
    - Session attribute
    - Static variable
- Recommendation
  - Avoid using an instance variable
  - Synchronize the code block

7

In Servlet 2.4, SingleThreadModel is deprecated for a good reason mostly because it does not really provide thread safety. For example, you still have to manage synchronized access to session attribute and static variables.

So recommendation is avoid the usage of instance variables and if you do, use sychronized code block.



Internationalization

? New elements in Deployment Descriptor

```
<locale-encoding-mapping-list>
  <locale-encoding-mapping>
    <locale>ja</locale>
    <encoding>Shift_JIS</encoding>
  </locale-encoding-mapping>
</locale-encoding-mapping-list>
```

? New methods in ServletResponse
  ? SetCharacterEncoding()
    ? no more setContentType("text/html; charset=UTF-8")
  ? GetContentType()
    ? useful since ContentType now can be more dynamically set

8

Also in Servlet 2.4, the ServletResponse interface (and the ServletResponseWrapper) adds two new methods:
* setCharacterEncoding(String encoding): Sets the response's character encoding. This method provides an alternative to passing a charset parameter to setContentType(String) or passing a Locale to setLocale(Locale). This method has no effect if called after getWriter() has been called or if the response has committed. For a list of acceptable Internet charsets, see Resources.
* getContentType(): Returns the response's content type. This may include a charset parameter set by either setContentType(), setLocale(), or setCharacterEncoding(). If no type has been specified, the method returns null.

The setCharacterEncoding() method pairs with the preexisting getCharacterEncoding() method to provide an easy way to manipulate and view the response's character encoding (charset). You can now avoid setting the charset via the awkward setContentType("text/html; charset=UTF-8") call.

The new getContentType() method pairs with the preexisting setContentType() method to expose the content type you've assigned. Formerly, this wouldn't have been too interesting, but now the type might be dynamically set with a combination of setContentType(), setLocale(), and setCharacterEncoding() calls, and this method provides a way to view the generated type string.

So which is better, setLocale() or setCharacterEncoding()? It depends. The former lets you specify a locale like ja for Japanese and lets the container handle the work of determining an appropriate charset. That's convenient, but, of course, many charsets might work for a given locale, and the developer has no choice in the matter. The latter method provides a new, easy way to choose a specific charset, letting you override the container's choice of Shift_JIS with EUC-JP, for example.

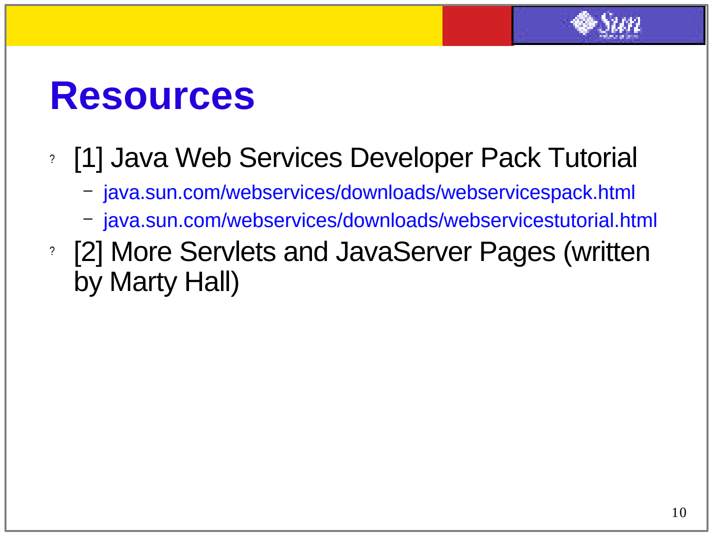
However, the story doesn't end there. Servlet 2.4 also introduces a new <locale-encoding-mapping-list> element in the web.xml deployment descriptor to let the deployer assign locale-to-charset mappings outside servlet code. It looks like this:

```
<locale-encoding-mapping-list>
  <locale-encoding-mapping>
    <locale>ja</locale>
    <encoding>Shift_JIS</encoding>
  </locale-encoding-mapping>
  <locale-encoding-mapping>
    <locale>zh_TW</locale>
    <encoding>Big5</encoding>
  </locale-encoding-mapping>
</locale-encoding-mapping-list>
```

Now within this Web application, any response assigned to the ja locale uses the Shift_JIS charset, and any assigned to the zh_TW Chinese/Taiwan locale uses the Big5 charset. These values could later be changed to UTF-8 when it grows more popular among clients. Any locales not mentioned in the list will use the container-specific defaults as before.

Resources

9



Resources

- [1] Java Web Services Developer Pack Tutorial
  - java.sun.com/webservices/downloads/webservicespack.html
  - java.sun.com/webservices/downloads/webservicestutorial.html
- [2] More Servlets and JavaServer Pages (written by Marty Hall)

10

This is the resource page.

Live your life with Passion!