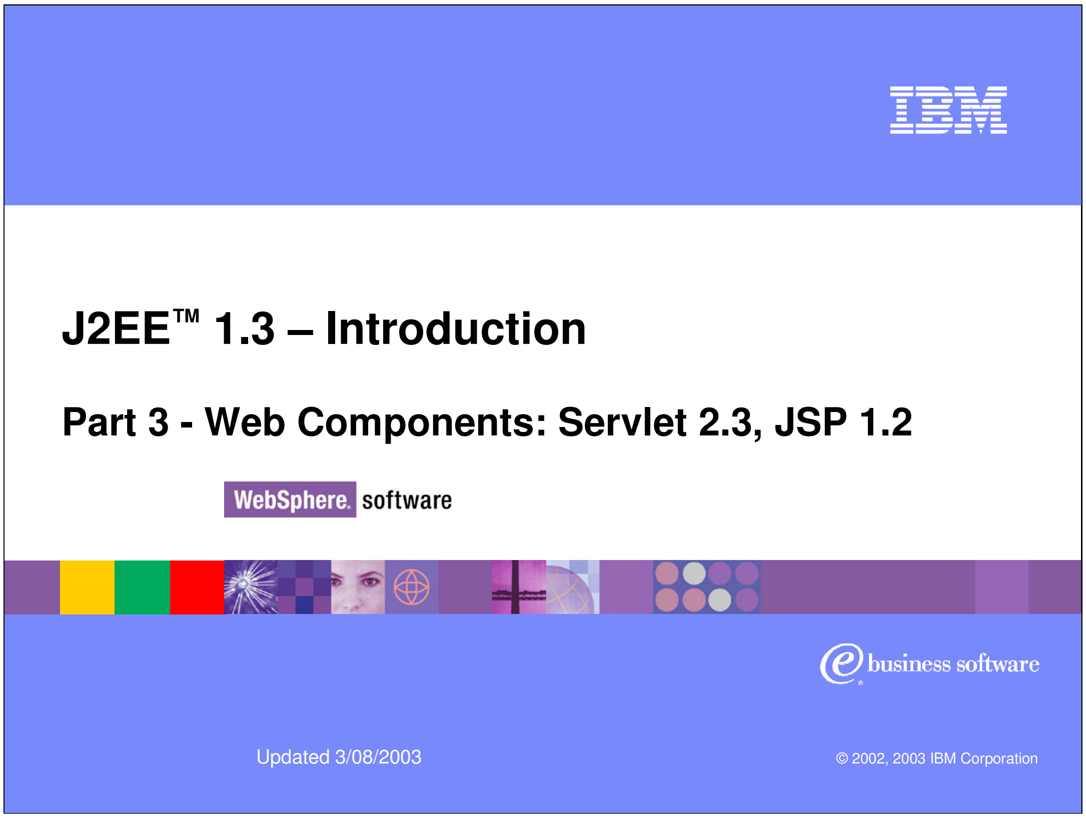
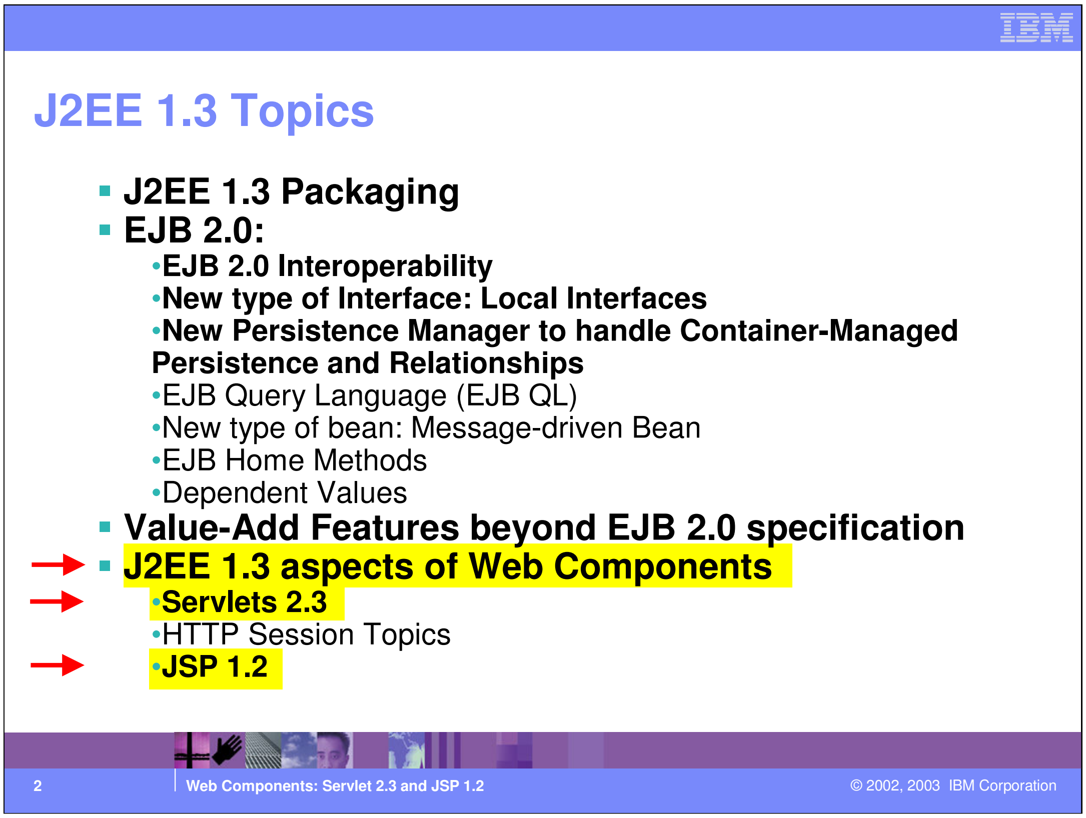


J2EE™ 1.3 – Introduction

Part 3 - Web Components: Servlet 2.3, JSP 1.2

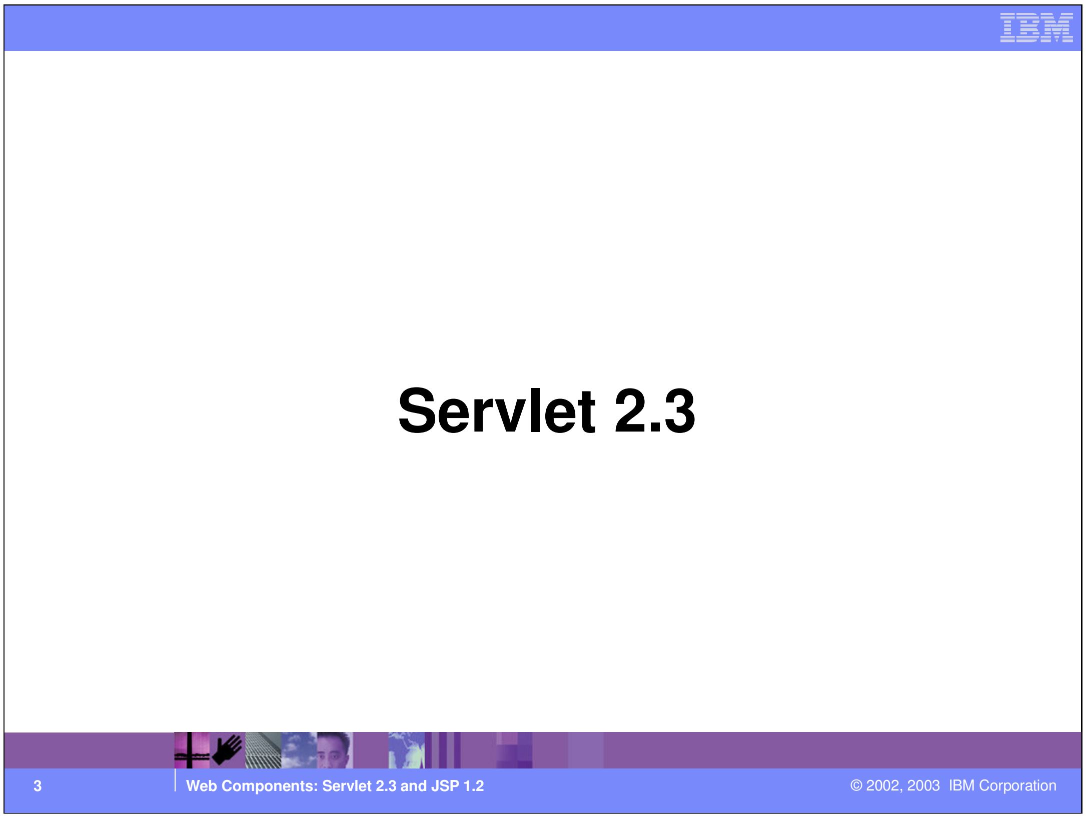
IBM
WebSphere. software
e business software

Updated 3/08/2003

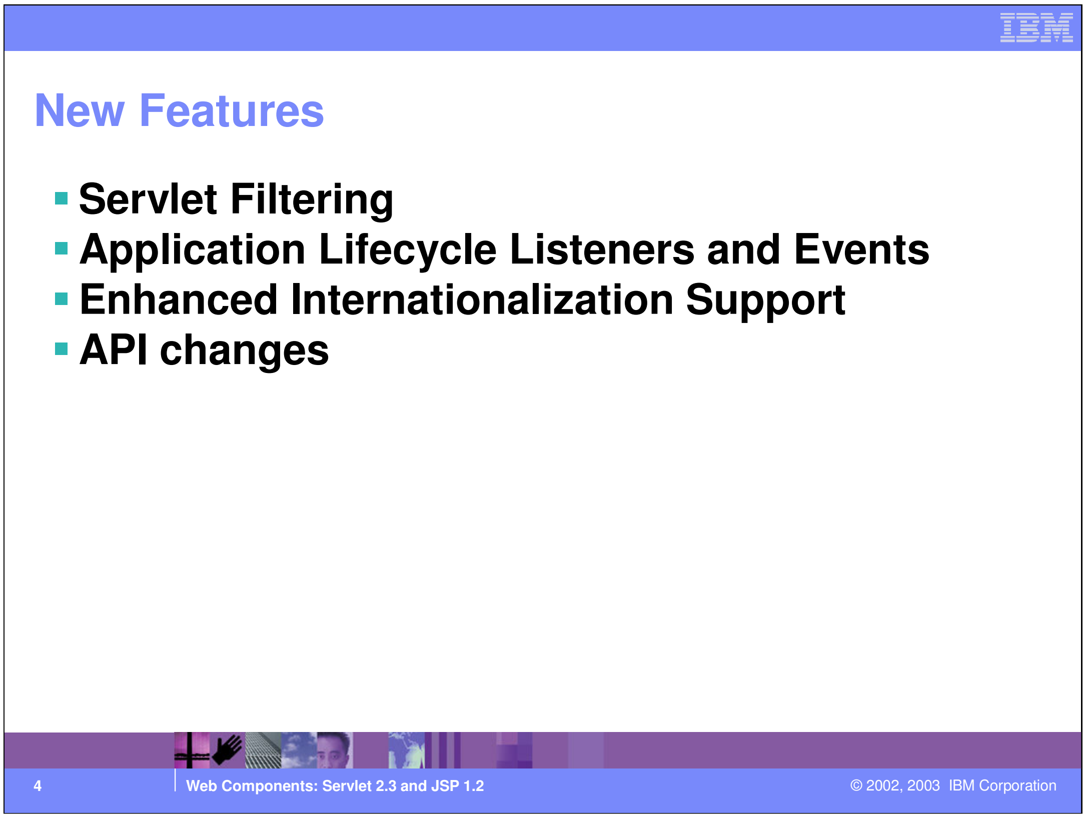


J2EE 1.3 Topics

- **J2EE 1.3 Packaging**
- **EJB 2.0:**
  - **EJB 2.0 Interoperability**
  - **New type of Interface: Local Interfaces**
  - **New Persistence Manager to handle Container-Managed Persistence and Relationships**
  - EJB Query Language (EJB QL)
  - New type of bean: Message-driven Bean
  - EJB Home Methods
  - Dependent Values
- **Value-Add Features beyond EJB 2.0 specification**
- → **J2EE 1.3 aspects of Web Components**
  - → **Servlets 2.3**
  - HTTP Session Topics
  - → **JSP 1.2**



Servlet 2.3



New Features

- Servlet Filtering
- Application Lifecycle Listeners and Events
- Enhanced Internationalization Support
- API changes

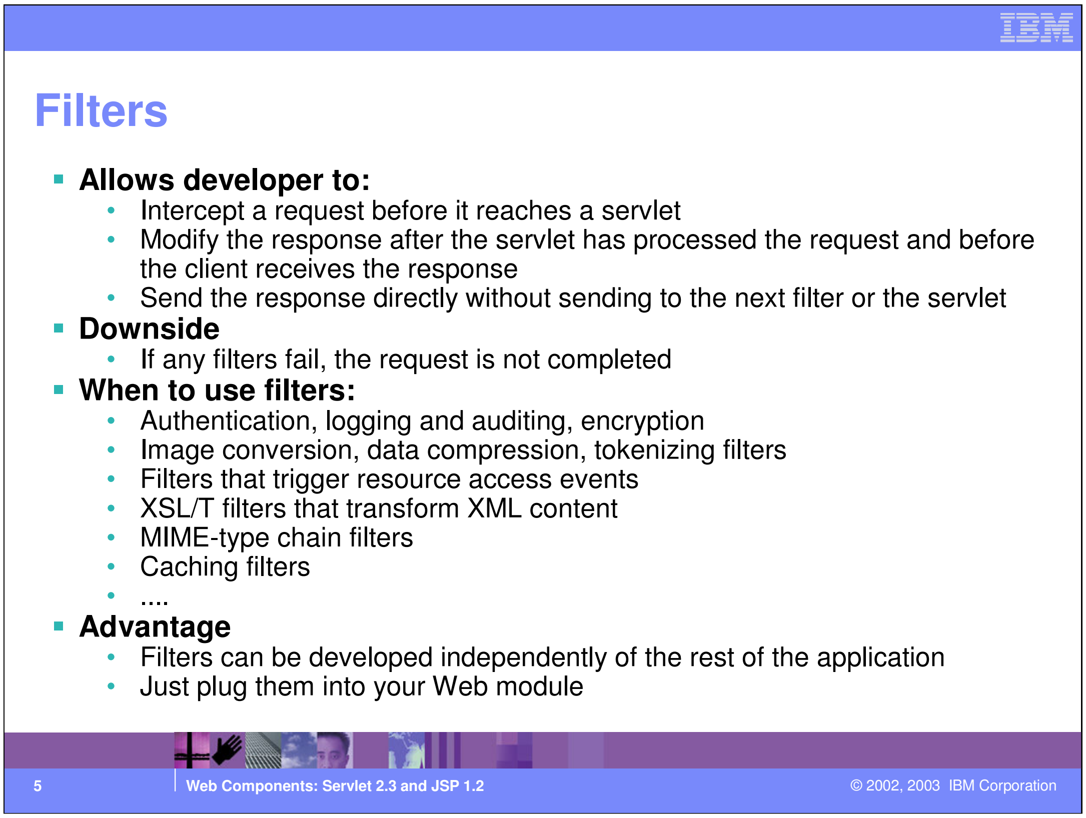
4 Web Components: Servlet 2.3 and JSP 1.2

There are a few interesting additions to the web container specifications.

Definitely the major addition is represented by the introduction of filters.

Also, a new range of listeners can be defined on web applications allowing the developer better control over the events that occur in the web container.

Internationalization is also an area where some enhancements were introduced as well as a number of API changes.



Filters

- **Allows developer to:**
  - Intercept a request before it reaches a servlet
  - Modify the response after the servlet has processed the request and before the client receives the response
  - Send the response directly without sending to the next filter or the servlet
- **Downside**
  - If any filters fail, the request is not completed
- **When to use filters:**
  - Authentication, logging and auditing, encryption
  - Image conversion, data compression, tokenizing filters
  - Filters that trigger resource access events
  - XSL/T filters that transform XML content
  - MIME-type chain filters
  - Caching filters
  - ....
- **Advantage**
  - Filters can be developed independently of the rest of the application
  - Just plug them into your Web module

5 Web Components: Servlet 2.3 and JSP 1.2

All filters implement the javax.filter.filter interface, which contains the following methods:

init(FilterConfig): Called by the Web container when the filter initializes. Only one instance per filter declaration in the deployment descriptor

destroy(): Called by the Web container when the filter closes. Allows filter to free resources obtained in init()

getConfig()

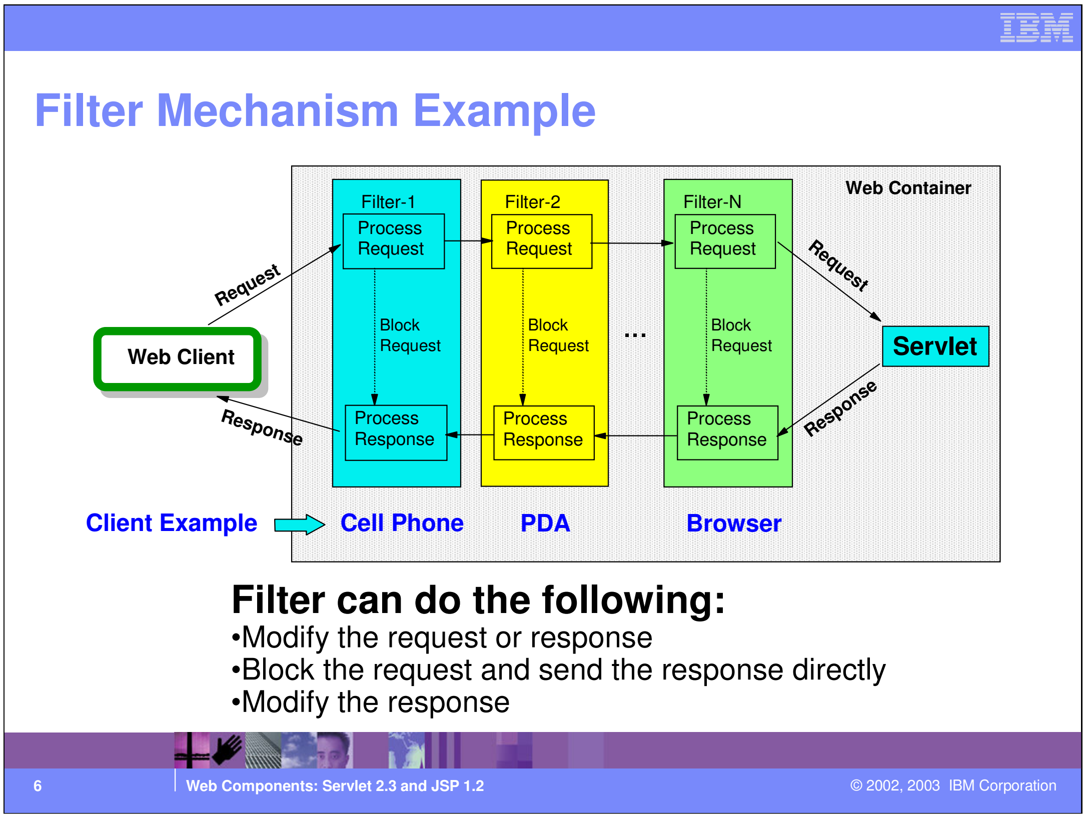
setFilterConfig()

doFilter(): Represents the main function that can modify the request and the response. This filter can implement the following pattern, or a subset of this pattern:

•Create customized implementation of the Request object (ServletRequest or HttpServletRequest) to modify the request and header

•Invoke the next entity in the filter chain, which can include the next filter, or the target Web resource (if this is the last filter, as defined in the deployment descriptor). Calling the doFilter method on the FilterChain affects the invocation of the next entity object, passing in the request and response it was called with, or wrapped versions it created. The filter can block the request, by not calling the doFilter method and sending the response back instead.

•Examine the response and wrap the Response object passed in to its doFilter method, with a customized implementation of ServletResponse or HttpServletResponse, to modify response headers or data.



Filter Mechanism Example

Web Container

Web Client

Request

Response

Filter-1

Process Request

Block Request

Process Response

Filter-2

Process Request

Block Request

Process Response

...

Filter-N

Process Request

Block Request

Process Response

Request

Response

Servlet

Client Example

Cell Phone

PDA

Browser

Filter can do the following:

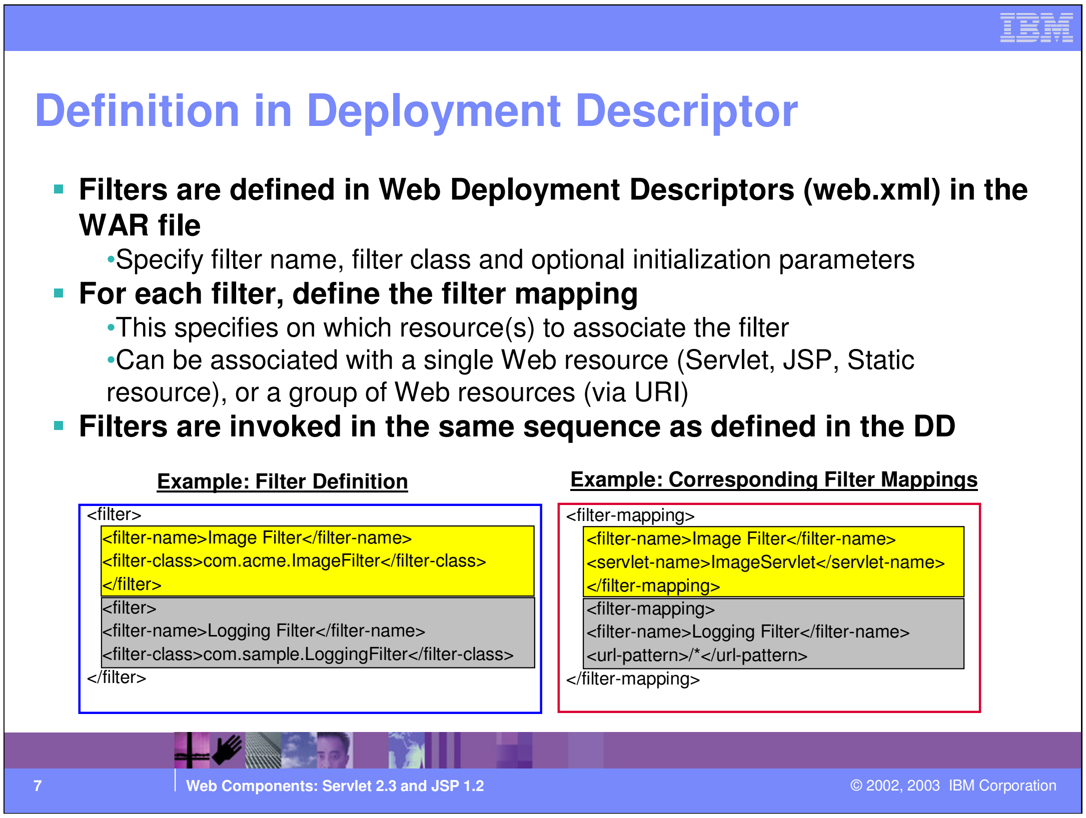
- Modify the request or response
- Block the request and send the response directly
- Modify the response

This chart shows how filters can be chained to provide for a cascaded processing of the HTTP request and response.

The chart also shows that a software provider may customize the application by adding filters depending on the clients that are going to be used.

For example, when installing the application at a customer's site where the web browser is going to be used, only the right-most filter is going to be needed.

When installing the application for a customer that needs access through a cell phone, we may configure additional filters - but the logic of the application doesn't change and the filters can be developed and configured independently.



Definition in Deployment Descriptor

- **Filters are defined in Web Deployment Descriptors (web.xml) in the WAR file**
  - Specify filter name, filter class and optional initialization parameters
- **For each filter, define the filter mapping**
  - This specifies on which resource(s) to associate the filter
  - Can be associated with a single Web resource (Servlet, JSP, Static resource), or a group of Web resources (via URI)
- **Filters are invoked in the same sequence as defined in the DD**

Example: Filter Definition

```
<filter>
   <filter-name>Image Filter</filter-name>
   <filter-class>com.acme.ImageFilter</filter-class>
   </filter>
   <filter>
   <filter-name>Logging Filter</filter-name>
   <filter-class>com.sample.LoggingFilter</filter-class>
</filter>
```

Example: Corresponding Filter Mappings

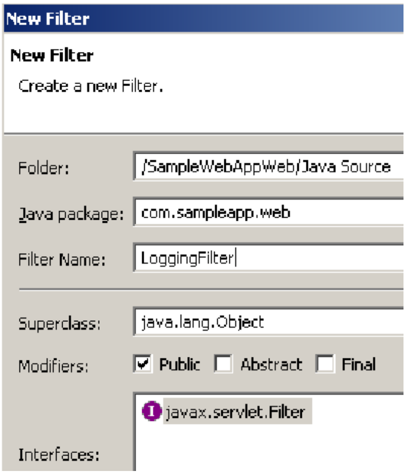
```
<filter-mapping>
   <filter-name>Image Filter</filter-name>
   <servlet-name>ImageServlet</servlet-name>
   </filter-mapping>
   <filter-mapping>
   <filter-name>Logging Filter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

7 Web Components: Servlet 2.3 and JSP 1.2

Define filter configuration in a Web application in the deployment descriptor, using the filter element, specifying the filter name, class and initialization parameters

Associate filters with a single Web resource (servlet, JSP, static resource), or a group of Web resources. Make this association using the servlet name, or using the URL-pattern. This approach uses a filter-mapping element in the deployment descriptor.

Create Filters in Application Developer



New Filter

New Filter

Create a new Filter.

Folder: /SampleWebAppWeb/Java Source

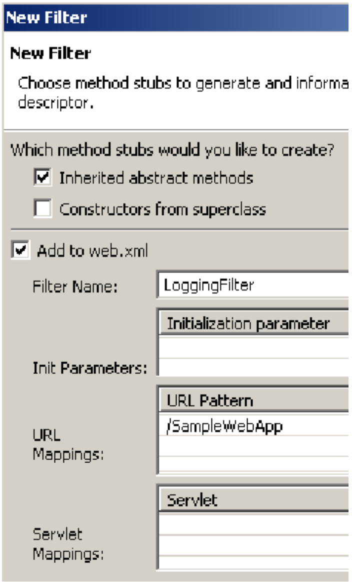
Java package: com.sampleapp.web

Filter Name: LoggingFilter

Superclass: java.lang.Object

Modifiers: ☑ Public □ Abstract □ Final

Interfaces: javax.servlet.Filter



New Filter

New Filter

Choose method stubs to generate and informa
descriptor.

Which method stubs would you like to create?

☑ Inherited abstract methods

□ Constructors from superclass

☑ Add to web.xml

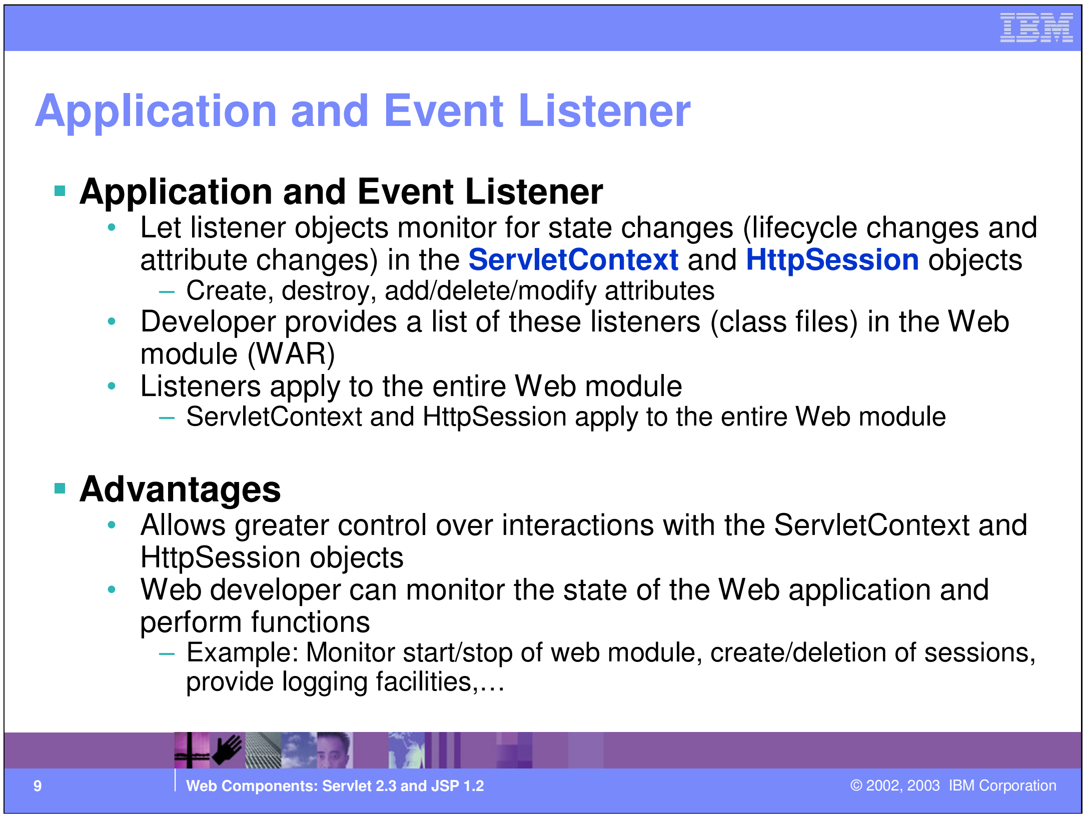
Filter Name: LoggingFilter

Init Parameters: Initialization parameter

URL Mappings: URL Pattern /SampleWebApp

Servlet Mappings: Servlet

WebSphere Studio App Developer facilitates the creation of filters thanks to a wizard that creates a skeleton for you and configures the filter in the Web Application deployment descriptor(web.xml).



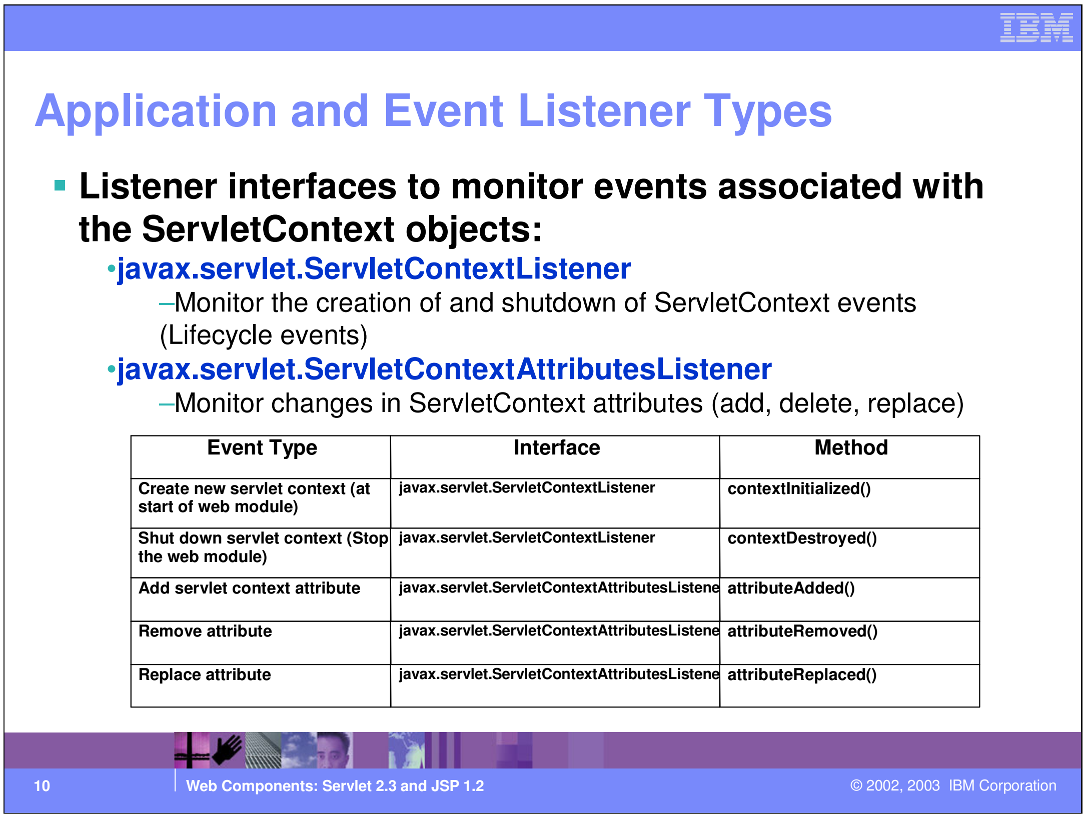
Application and Event Listener

- **Application and Event Listener**
  - Let listener objects monitor for state changes (lifecycle changes and attribute changes) in the **ServletContext** and **HttpSession** objects
    - Create, destroy, add/delete/modify attributes
  - Developer provides a list of these listeners (class files) in the Web module (WAR)
  - Listeners apply to the entire Web module
    - ServletContext and HttpSession apply to the entire Web module
- **Advantages**
  - Allows greater control over interactions with the ServletContext and HttpSession objects
  - Web developer can monitor the state of the Web application and perform functions
    - Example: Monitor start/stop of web module, create/deletion of sessions, provide logging facilities,…

The Web container instantiates and registers these listeners at the time of Web application deployment, prior to the start of the first request into the Web application.

You can use these listener to provide a very close control over the events that occur in the web container, such as the activation or deactivation of a web application, or events occurring to the http session.

The listeners can implement a number of actions, that may include initialization, cleanup, logging, auditing, and so on.

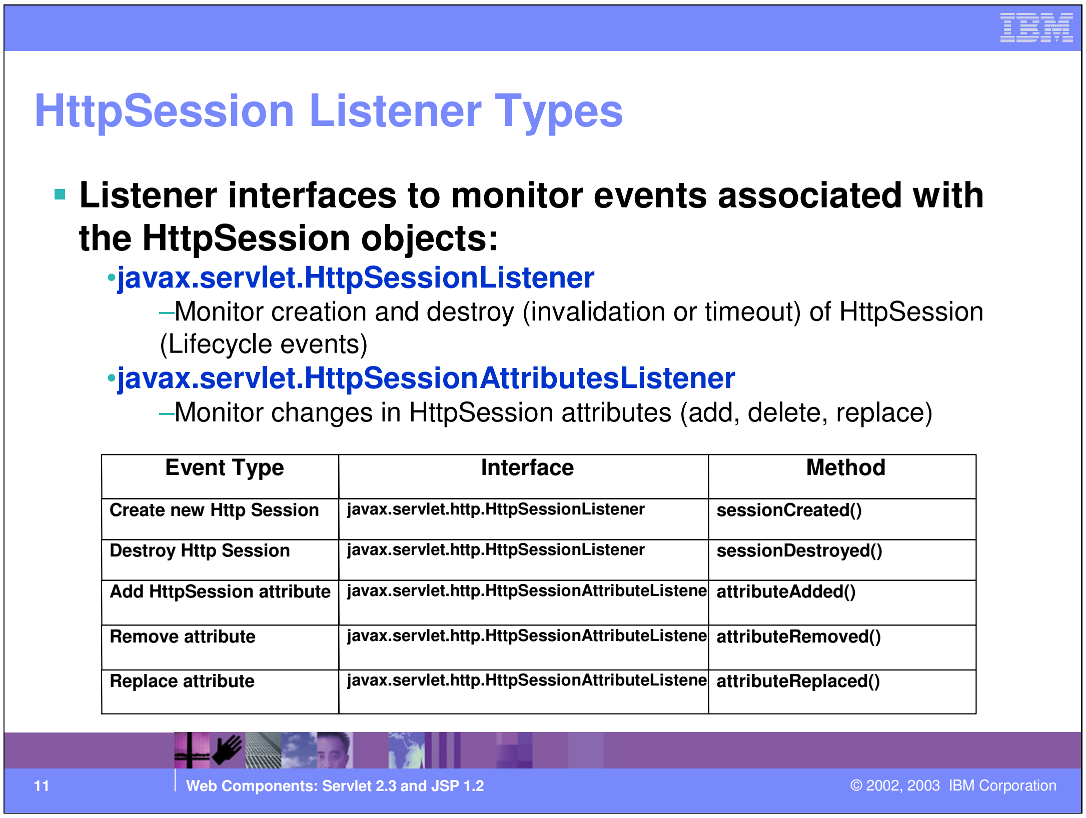


Application and Event Listener Types

- **Listener interfaces to monitor events associated with the ServletContext objects:**
  - **javax.servlet.ServletContextListener**
    - Monitor the creation of and shutdown of ServletContext events (Lifecycle events)
  - **javax.servlet.ServletContextAttributesListener**
    - Monitor changes in ServletContext attributes (add, delete, replace)

Event Type	Interface	Method
Create new servlet context (at start of web module)	javax.servlet.ServletContextListener	contextInitialized()
Shut down servlet context (Stop the web module)	javax.servlet.ServletContextListener	contextDestroyed()
Add servlet context attribute	javax.servlet.ServletContextAttributesListene	attributeAdded()
Remove attribute	javax.servlet.ServletContextAttributesListene	attributeRemoved()
Replace attribute	javax.servlet.ServletContextAttributesListene	attributeReplaced()

These are the interfaces that can be implemented by an Application/Event listener. You may create a listener that implements a number of these.

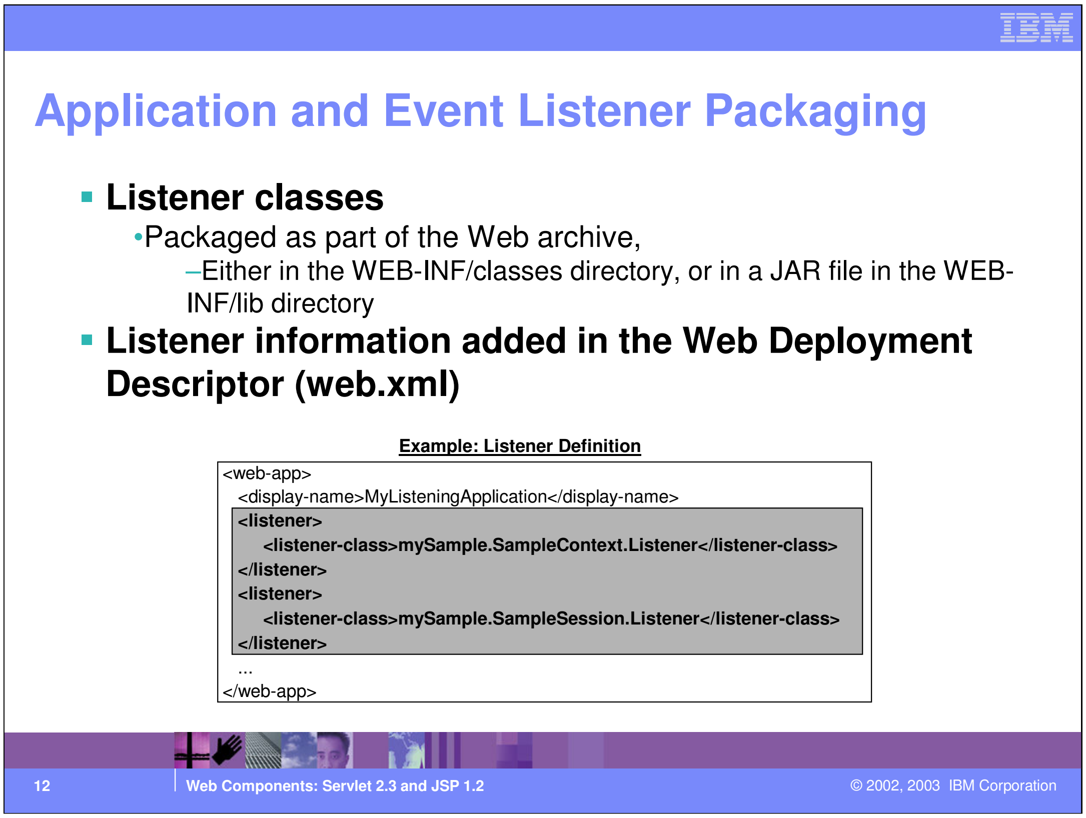


HttpSession Listener Types

- **Listener interfaces to monitor events associated with the HttpSession objects:**
  - **javax.servlet.HttpSessionListener**
    - Monitor creation and destroy (invalidation or timeout) of HttpSession (Lifecycle events)
  - **javax.servlet.HttpSessionAttributesListener**
    - Monitor changes in HttpSession attributes (add, delete, replace)

Event Type	Interface	Method
Create new Http Session	javax.servlet.http.HttpSessionListener	sessionCreated()
Destroy Http Session	javax.servlet.http.HttpSessionListener	sessionDestroyed()
Add HttpSession attribute	javax.servlet.http.HttpSessionAttributeListene	attributeAdded()
Remove attribute	javax.servlet.http.HttpSessionAttributeListene	attributeRemoved()
Replace attribute	javax.servlet.http.HttpSessionAttributeListene	attributeReplaced()

These interfaces are available for monitoring the events associated to the HTTP session.



Application and Event Listener Packaging

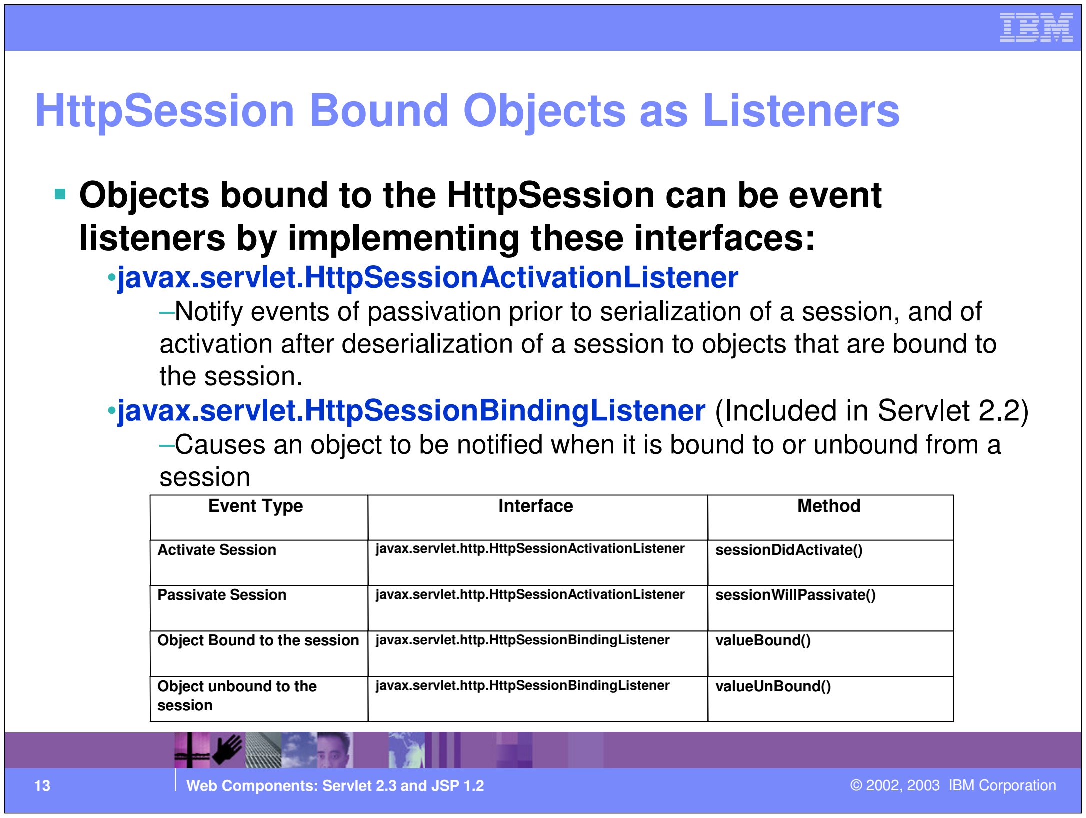
- **Listener classes**
  - Packaged as part of the Web archive,
    - Either in the WEB-INF/classes directory, or in a JAR file in the WEB-INF/lib directory
- **Listener information added in the Web Deployment Descriptor (web.xml)**

Example: Listener Definition

```
<web-app>
  <display-name>MyListeningApplication</display-name>
  <listener>
      <listener-class>mySample.SampleContext.Listener</listener-class>
  </listener>
  <listener>
      <listener-class>mySample.SampleSession.Listener</listener-class>
  </listener>
  ...
</web-app>
```

12 Web Components: Servlet 2.3 and JSP 1.2 © 2002, 2003 IBM Corporation

Listeners are defined in Deployment Descriptor using "listener" elements. The Listeners can have multiple listeners defined in a Web module. The web container handles the instantiation and registration of the listeners, prior to the start of the first request into the Web application.



HttpSession Bound Objects as Listeners

- **Objects bound to the HttpSession can be event listeners by implementing these interfaces:**
  - **javax.servlet.HttpSessionActivationListener**
    - Notify events of passivation prior to serialization of a session, and of activation after deserialization of a session to objects that are bound to the session.
  - **javax.servlet.HttpSessionBindingListener** (Included in Servlet 2.2)
    - Causes an object to be notified when it is bound to or unbound from a session

Event Type	Interface	Method
Activate Session	javax.servlet.http.HttpSessionActivationListener	sessionDidActivate()
Passivate Session	javax.servlet.http.HttpSessionActivationListener	sessionWillPassivate()
Object Bound to the session	javax.servlet.http.HttpSessionBindingListener	valueBound()
Object unbound to the session	javax.servlet.http.HttpSessionBindingListener	valueUnBound()

13 Web Components: Servlet 2.3 and JSP 1.2 © 2002, 2003 IBM Corporation

javax.servlet.http.HttpSessionActivationListener

> Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated. A container that migrates session between VMs or persists sessions is required to notify all attributes bound to sessions implementing HttpSessionActivationListener

javax.servlet.http.HttpSessionBindingListener

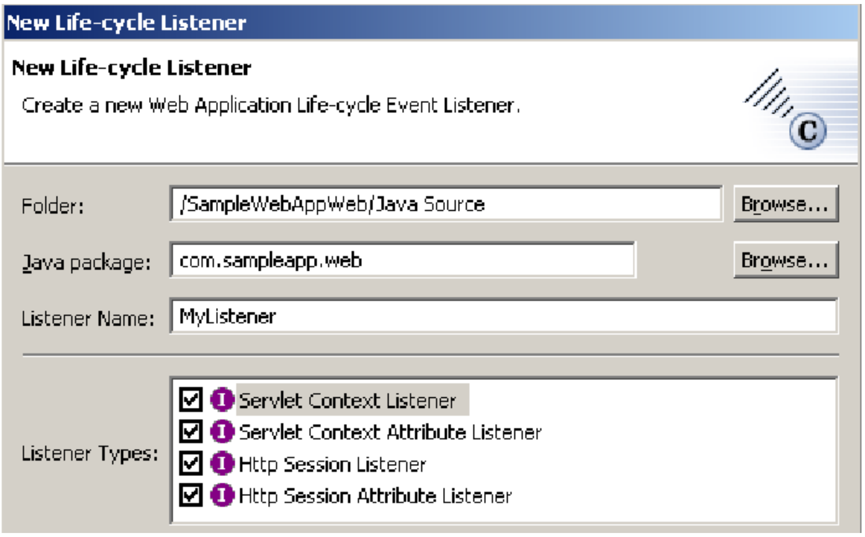
> Causes an object to be notified when it is bound to or unbound from a session. The object is notified by an HttpSessionBindingEvent object. This may be as a result of a servlet programmer explicitly unbinding an attribute from a session, due to a session being invalidated, or due to a session timing out.

In both the above cases, the Object that is bound to the HttpSession is the one implementing the interface.

When an application stores an object in or removes an object from a session, the session checks whether the object implements HttpSessionBindingListener. If it does, the servlet notifies the object that it has been bound to or unbound from the session. Notifications are sent after the binding methods complete. For session that are invalidated or expire, notifications are sent after the session has been invalidated or expired.

When container migrates a session between VMs in a distributed container setting, all session attributes implementing the HttpSessionActivationListener interface are notified.

Create Listeners Application Developer



New Life-cycle Listener

New Life-cycle Listener

Create a new Web Application Life-cycle Event Listener.

Folder: /SampleWebAppWeb/Java Source

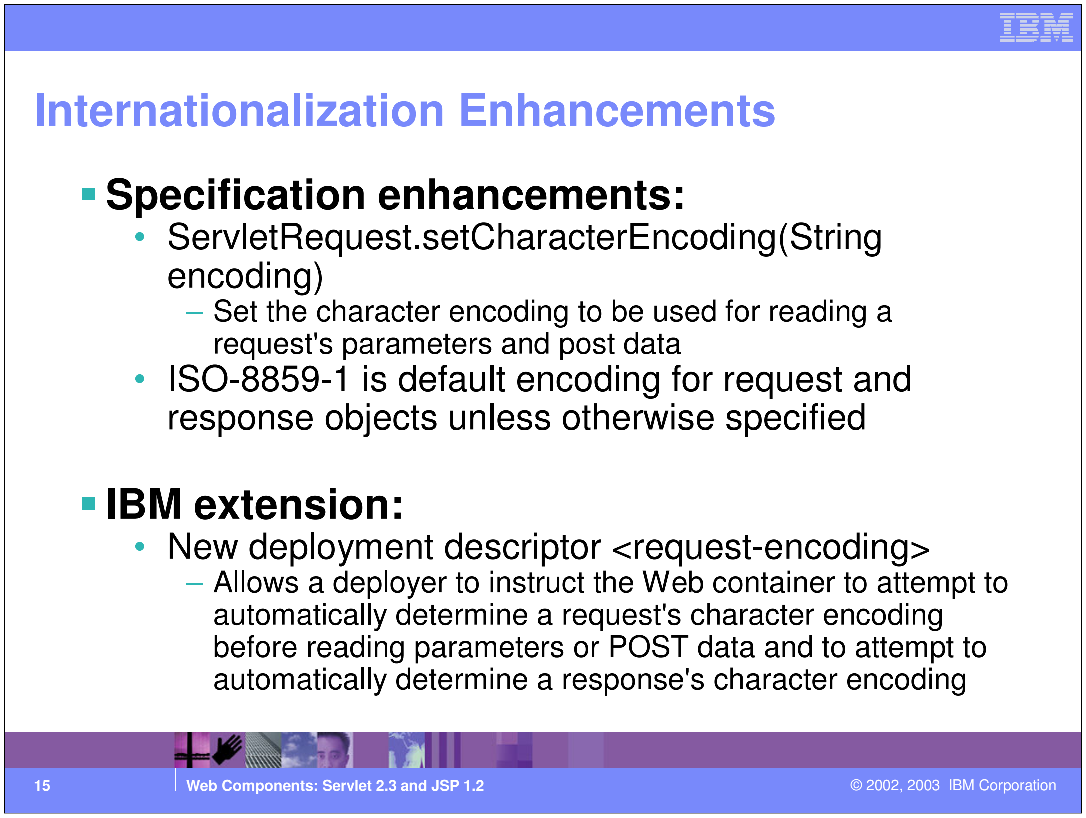
Java package: com.sampleapp.web

Listener Name: MyListener

Listener Types:

- ☑ Servlet Context Listener
- ☑ Servlet Context Attribute Listener
- ☑ Http Session Listener
- ☑ Http Session Attribute Listener

WebSphere Studio Application Developer provides a wizard for the lifecycle listeners that creates a skeleton of the desired listener and adds the listener, if needed, to the deployment descriptor of the web application.



Internationalization Enhancements

- **Specification enhancements:**
  - ServletRequest.setCharacterEncoding(String encoding)
    - Set the character encoding to be used for reading a request's parameters and post data
  - ISO-8859-1 is default encoding for request and response objects unless otherwise specified
- **IBM extension:**
  - New deployment descriptor <request-encoding>
    - Allows a deployer to instruct the Web container to attempt to automatically determine a request's character encoding before reading parameters or POST data and to attempt to automatically determine a response's character encoding

15 Web Components: Servlet 2.3 and JSP 1.2 © 2002, 2003 IBM Corporation

Request Encoding allows to control the encoding scheme to be used to interpret the character strings in an HTTP request. The ISO-8859-1 (Latin-1) encoding scheme covers most of the Western European languages, including English and it's the most commonly used scheme in the Western world. That's why it's chosen as the default to fall back to.

Here are the steps that the container follows:

- if <request-encoding=j2ee>

  Request's character encoding will be set according to the Servlet 2.3 specification (a default encoding of ISO-8859-1 if none specified)

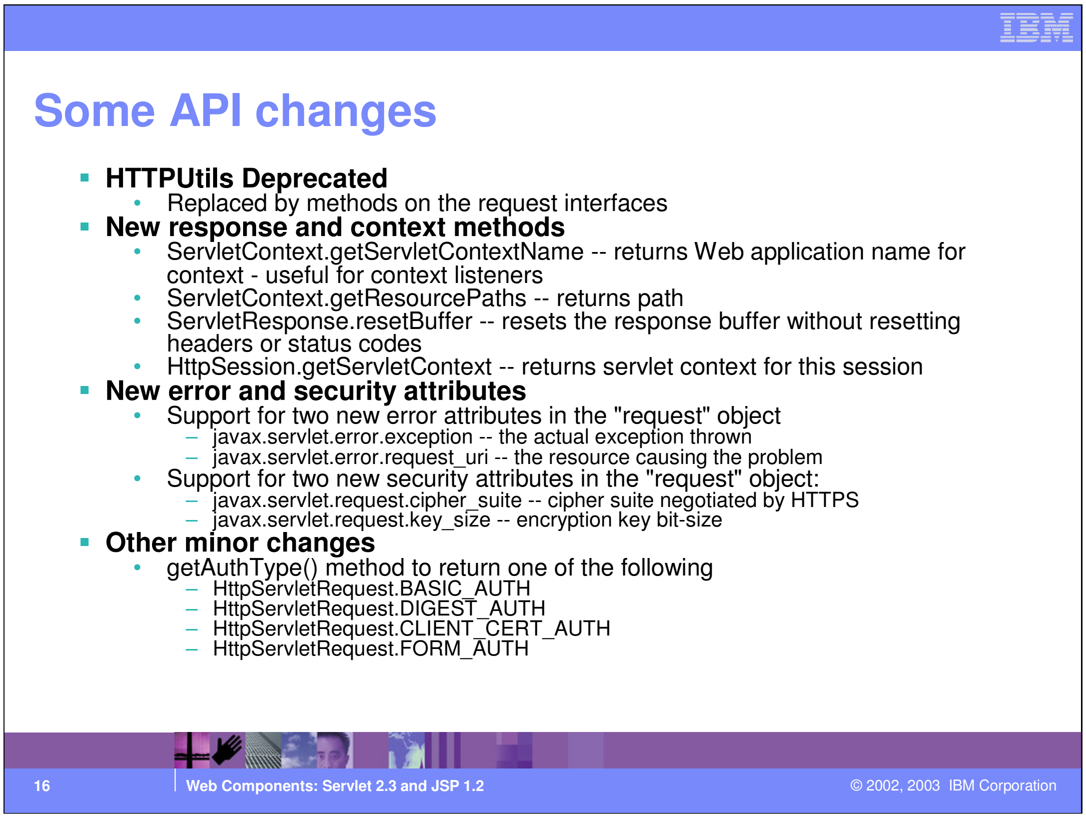
- if <request-encoding=automatic> and the client does not set the character encoding in the request header and that the servlet writer does not invoke the setCharacterEncoding(String) method before accessing request parameters, then Web Container will perform the following steps in order to ascertain the correct character encoding for the request parameters and data:

  Look for the charset in the Content-Type header

  Failing the above, try to map the server's locale to a character set using defined properties

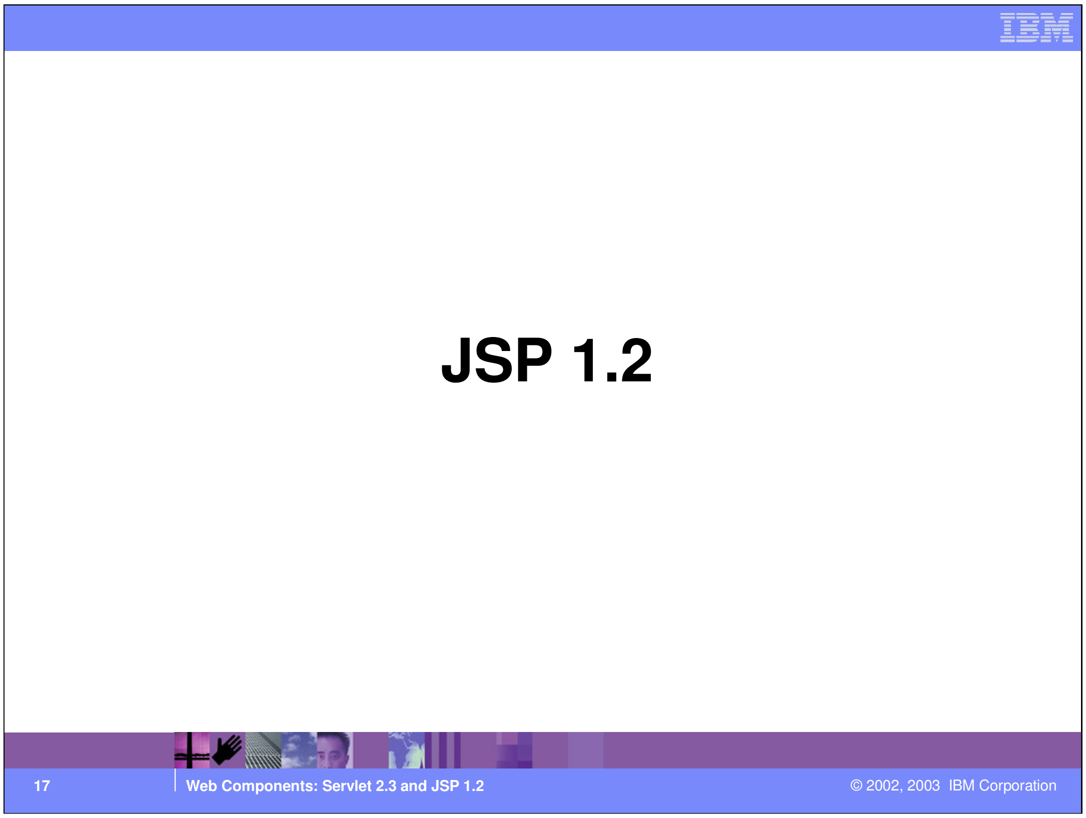
  Failing the above, try to use the DEFAULT_CLIENT_ENCODING system property, if set

  Finally, use the ISO-8859-1 character encoding

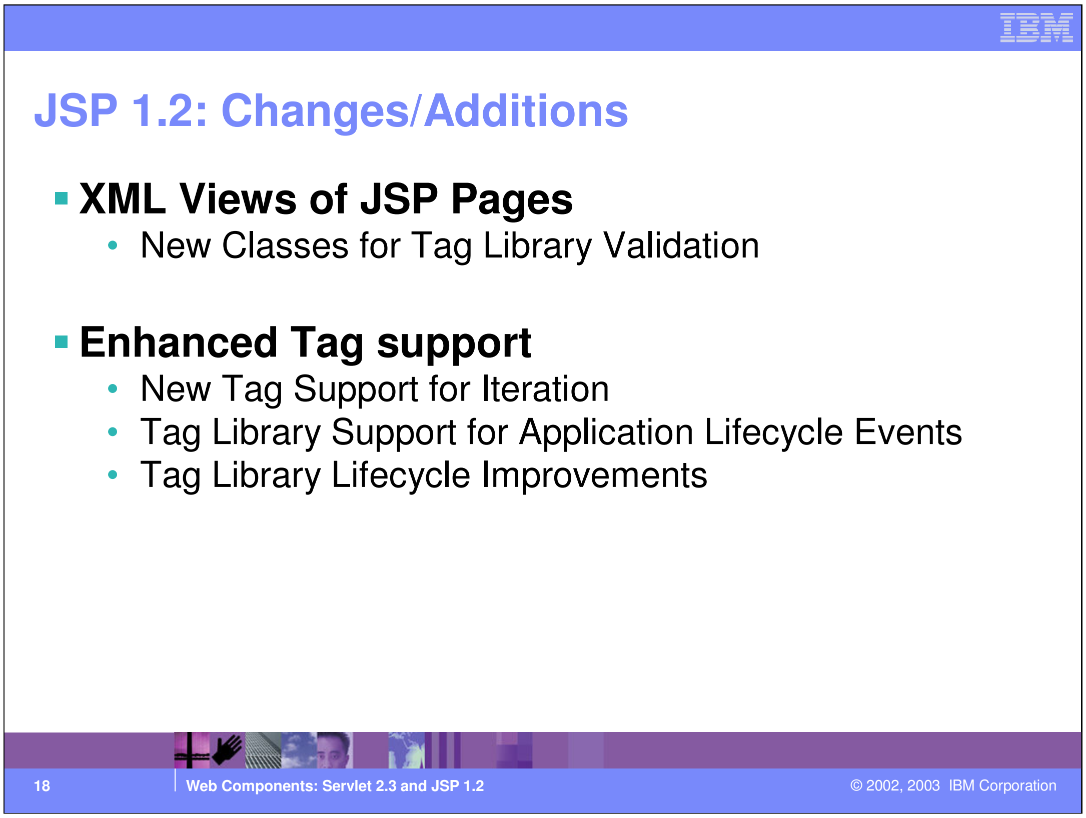


Some API changes

- **HTTPUtils Deprecated**
  - Replaced by methods on the request interfaces
- **New response and context methods**
  - ServletContext.getServletContextName -- returns Web application name for context - useful for context listeners
  - ServletContext.getResourcePaths -- returns path
  - ServletResponse.resetBuffer -- resets the response buffer without resetting headers or status codes
  - HttpSession.getServletContext -- returns servlet context for this session
- **New error and security attributes**
  - Support for two new error attributes in the "request" object
    - javax.servlet.error.exception -- the actual exception thrown
    - javax.servlet.error.request_uri -- the resource causing the problem
  - Support for two new security attributes in the "request" object:
    - javax.servlet.request.cipher_suite -- cipher suite negotiated by HTTPS
    - javax.servlet.request.key_size -- encryption key bit-size
- **Other minor changes**
  - getAuthType() method to return one of the following
    - HttpServletRequest.BASIC_AUTH
    - HttpServletRequest.DIGEST_AUTH
    - HttpServletRequest.CLIENT_CERT_AUTH
    - HttpServletRequest.FORM_AUTH



JSP 1.2

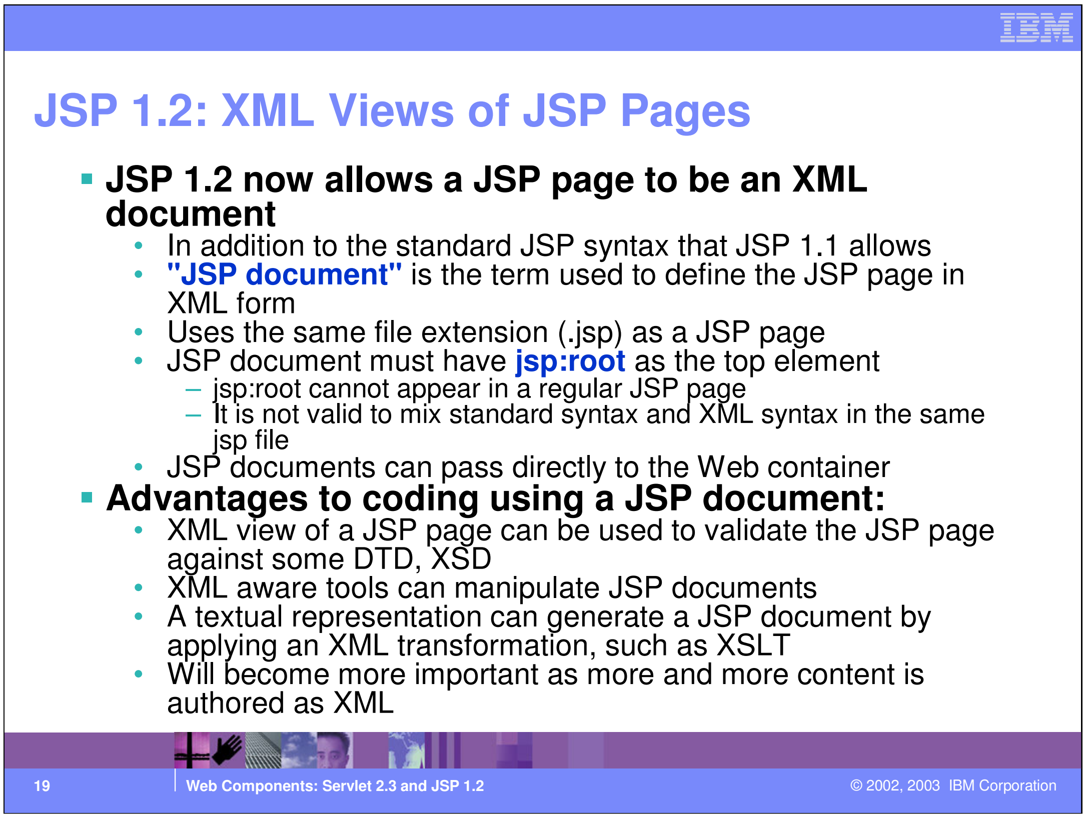


JSP 1.2: Changes/Additions

- **XML Views of JSP Pages**
  - New Classes for Tag Library Validation
- **Enhanced Tag support**
  - New Tag Support for Iteration
  - Tag Library Support for Application Lifecycle Events
  - Tag Library Lifecycle Improvements

18 Web Components: Servlet 2.3 and JSP 1.2

The major change for the JSP 1.2 spec is the ability to encode the JSP in pure XML format. There are also additional classes for the validation of tag libraries and new tags that facilitate iteration, and handling lifecycle events.



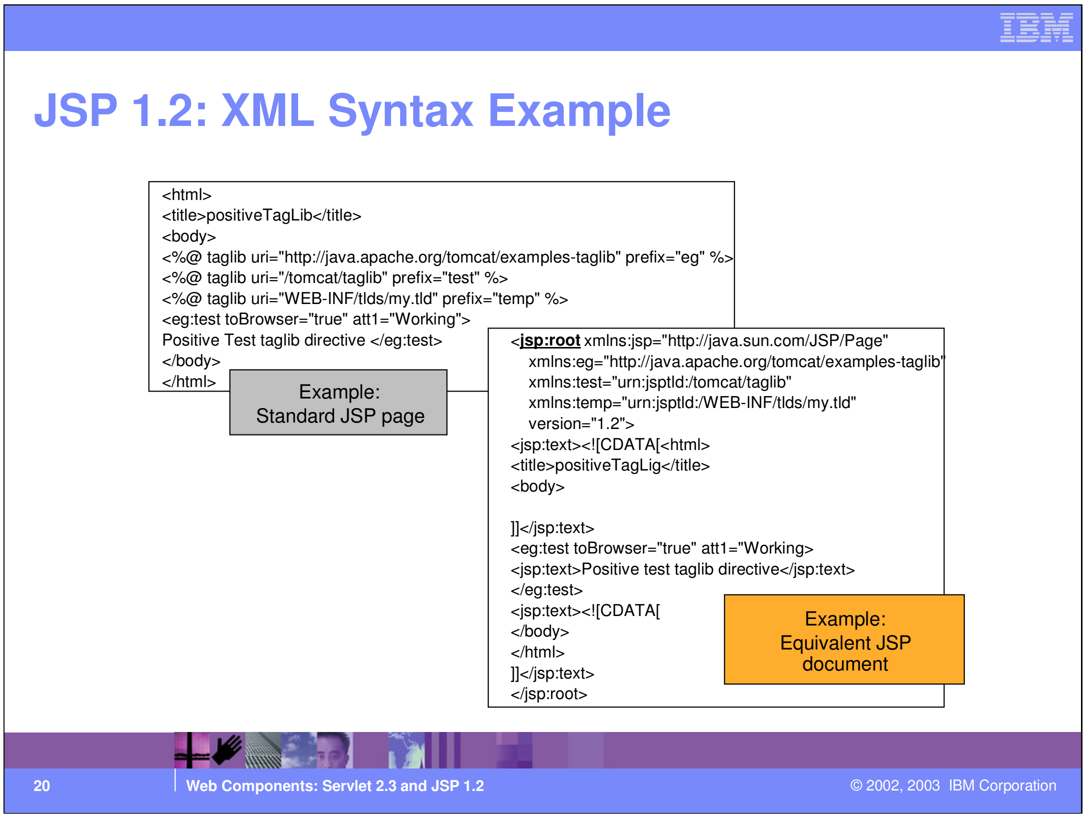
JSP 1.2: XML Views of JSP Pages

- **JSP 1.2 now allows a JSP page to be an XML document**
  - In addition to the standard JSP syntax that JSP 1.1 allows
  - **"JSP document"** is the term used to define the JSP page in XML form
  - Uses the same file extension (.jsp) as a JSP page
  - JSP document must have **jsp:root** as the top element
    - jsp:root cannot appear in a regular JSP page
    - It is not valid to mix standard syntax and XML syntax in the same jsp file
  - JSP documents can pass directly to the Web container
- **Advantages to coding using a JSP document:**
  - XML view of a JSP page can be used to validate the JSP page against some DTD, XSD
  - XML aware tools can manipulate JSP documents
  - A textual representation can generate a JSP document by applying an XML transformation, such as XSLT
  - Will become more important as more and more content is authored as XML

A synonym for JSP document is XML view of a JSP page.

The main benefit here is that every tool that is capable of manipulating XML can now be used with JSPs. Also, you can easily validate a JSP against a DTD or an XSD.

Another interesting possibility is provided by the ability to perform code transformation through XSL/T.



JSP 1.2: XML Syntax Example

```
<html>
<title>positiveTagLib</title>
<body>
<%@ taglib uri="http://java.apache.org/tomcat/examples-taglib" prefix="eg" %>
<%@ taglib uri="/tomcat/taglib" prefix="test" %>
<%@ taglib uri="WEB-INF/tlds/my.tld" prefix="temp" %>
<eg:test toBrowser="true" att1="Working">
Positive Test taglib directive </eg:test>
</body>
</html>
```

Example:
Standard JSP page

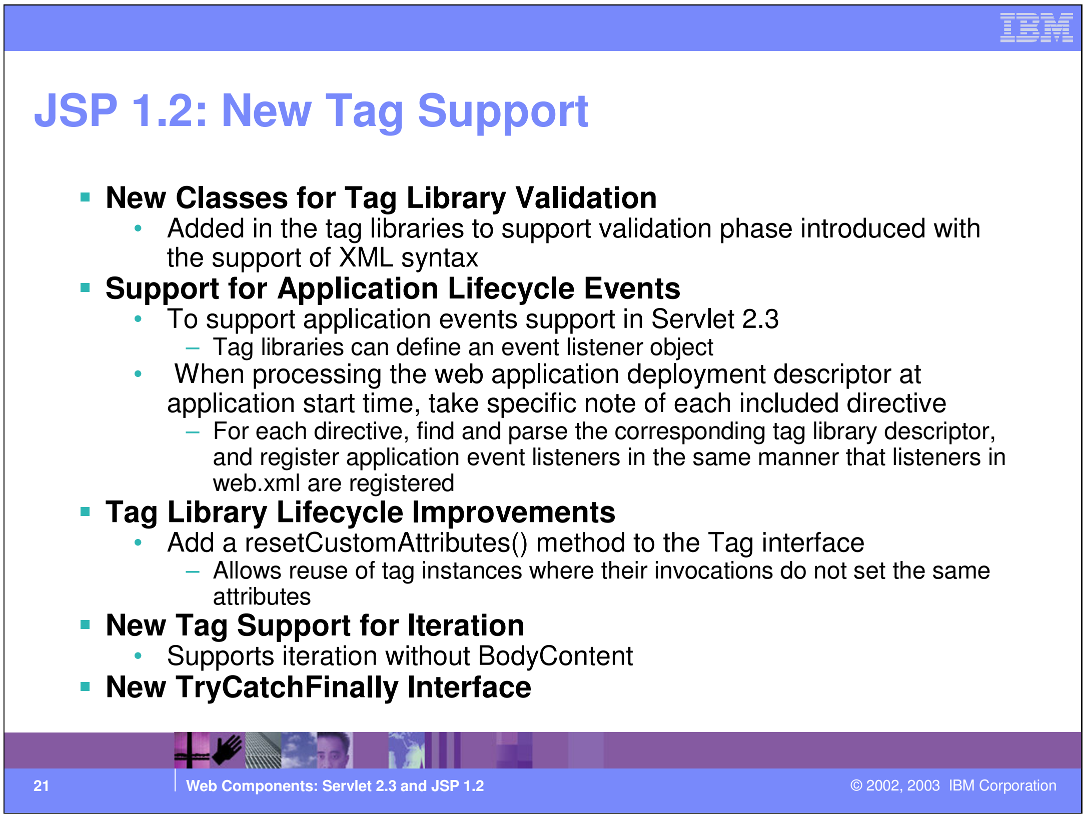
```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
   xmlns:eg="http://java.apache.org/tomcat/examples-taglib"
   xmlns:test="urn:jsptld:/tomcat/taglib"
   xmlns:temp="urn:jsptld:/WEB-INF/tlds/my.tld"
   version="1.2">
<jsp:text><![CDATA[<html>
<title>positiveTagLig</title>
<body>

]]</jsp:text>
<eg:test toBrowser="true" att1="Working>
<jsp:text>Positive test taglib directive</jsp:text>
</eg:test>
<jsp:text><![CDATA[
</body>
</html>
]]</jsp:text>
</jsp:root>
```

Example:
Equivalent JSP
document

20 | Web Components: Servlet 2.3 and JSP 1.2 © 2002, 2003 IBM Corporation

Here is an example of a JSP "old style" and of the corresponding version in XML.

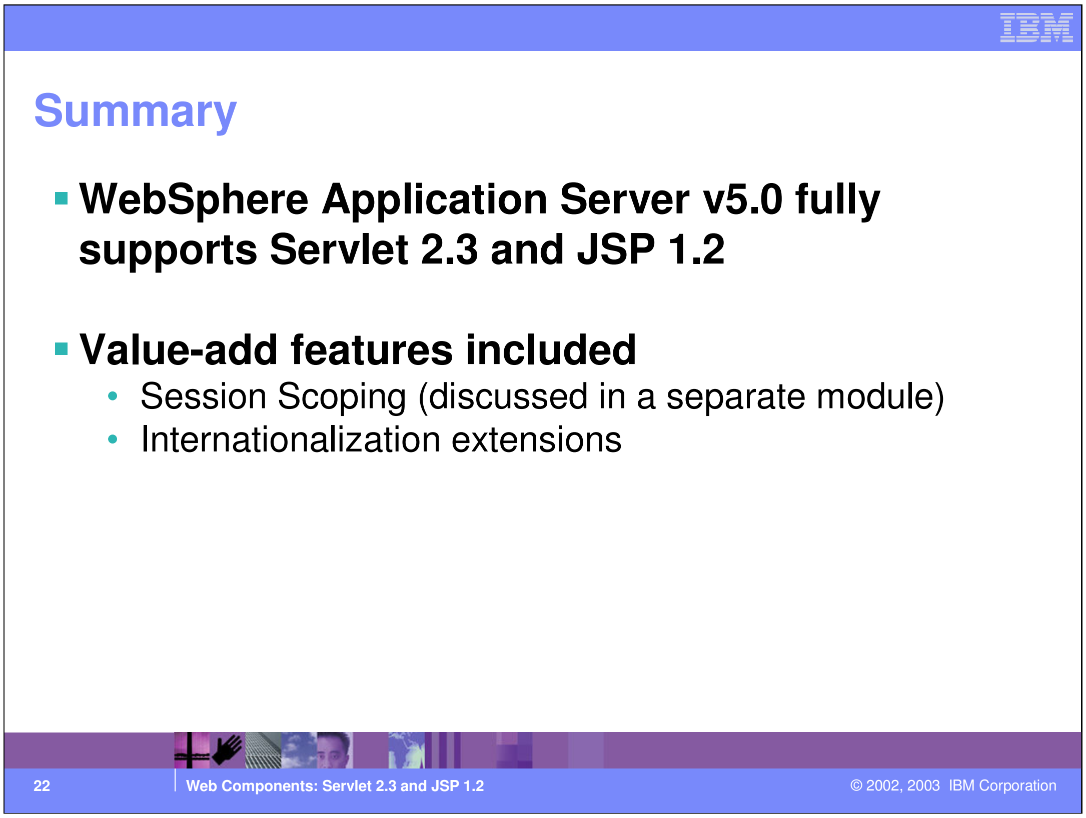


JSP 1.2: New Tag Support

- **New Classes for Tag Library Validation**
  - Added in the tag libraries to support validation phase introduced with the support of XML syntax
- **Support for Application Lifecycle Events**
  - To support application events support in Servlet 2.3
    - Tag libraries can define an event listener object
  - When processing the web application deployment descriptor at application start time, take specific note of each included directive
    - For each directive, find and parse the corresponding tag library descriptor, and register application event listeners in the same manner that listeners in web.xml are registered
- **Tag Library Lifecycle Improvements**
  - Add a resetCustomAttributes() method to the Tag interface
    - Allows reuse of tag instances where their invocations do not set the same attributes
- **New Tag Support for Iteration**
  - Supports iteration without BodyContent
- **New TryCatchFinally Interface**

The JSP 1.2 specifications include support for validating tag libraries - as part of the validation process that is introduced by the new XML syntax.

Tags can also be defined to include event listener objects as per the Servlet 2.3 specs. Conceptually, these special directives work the same as the corresponding entries in the web.xml deployment descriptor. It's up to the container to process these directives at application startup and to register the listeners.



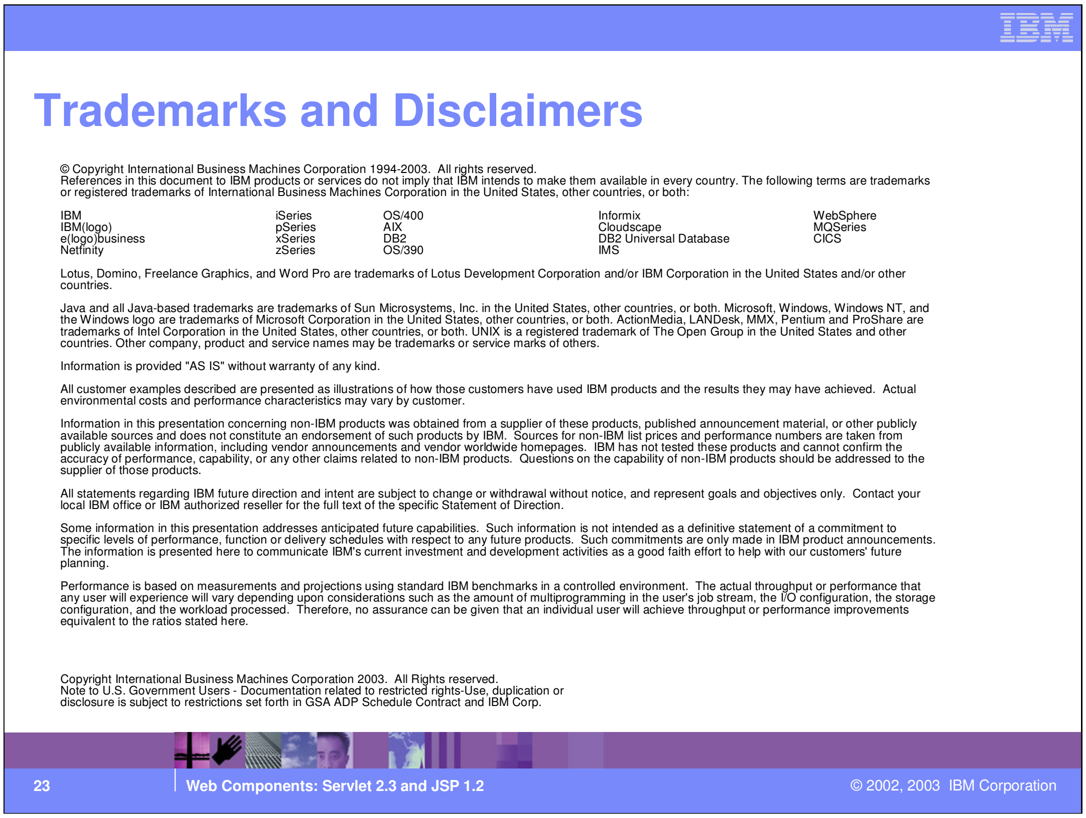
Summary

- **WebSphere Application Server v5.0 fully supports Servlet 2.3 and JSP 1.2**
- **Value-add features included**
  - Session Scoping (discussed in a separate module)
  - Internationalization extensions

WebSphere fully supports the new specs and in the WebSphere Enterprise product extends the specifications to provide broader support in areas such Internationalization.

Also Session Scoping is an important part of the specifications. The J2EE 1.3 specs mandate that the container scopes the HTTP Session at the servlet context level (in practice, session information cannot span multiple WAR files).

But WebSphere Application Server extends the specs allowing more flexible forms of session sharing - see the module on HTTP Session Management.



Trademarks and Disclaimers

References in this document to IBM products or services do not imply that IBM intends to make them available in every country. The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
IBM(logo)
e(logo)business
Netfinity
iSeries
pSeries
xSeries
zSeries
OS/400
AIX
DB2
OS/390
Informix
Cloudscape
DB2 Universal Database
IMS
WebSphere
MQSeries
CICS

Lotus, Domino, Freelance Graphics, and Word Pro are trademarks of Lotus Development Corporation and/or IBM Corporation in the United States and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both. ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Other company, product and service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information in this presentation concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information in this presentation addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.