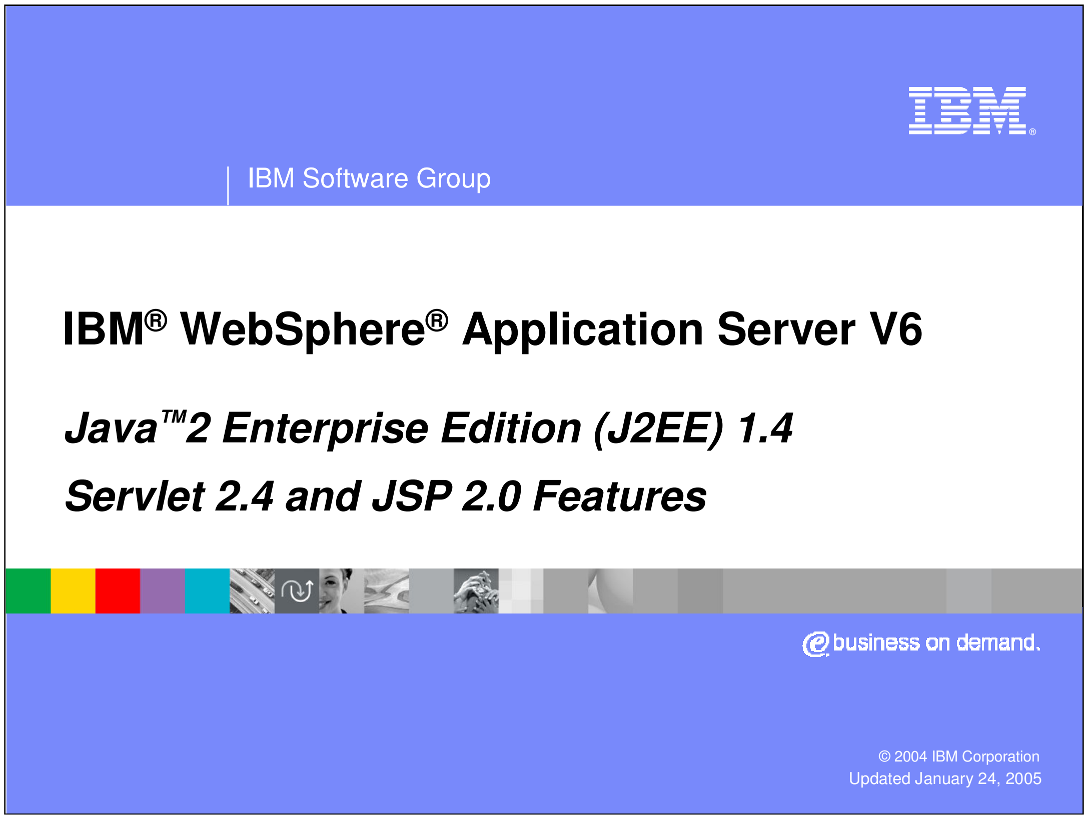


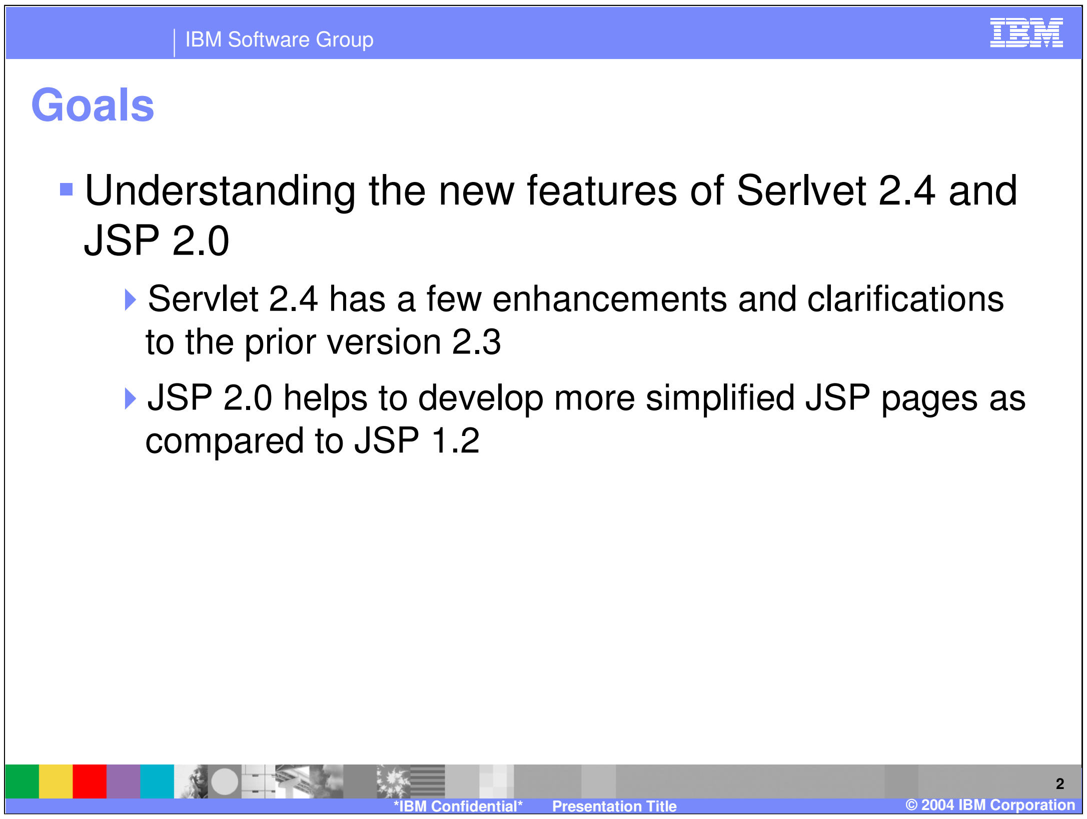
IBM Software Group

# IBM® WebSphere® Application Server V6

## *Java™2 Enterprise Edition (J2EE) 1.4*
## *Servlet 2.4 and JSP 2.0 Features*

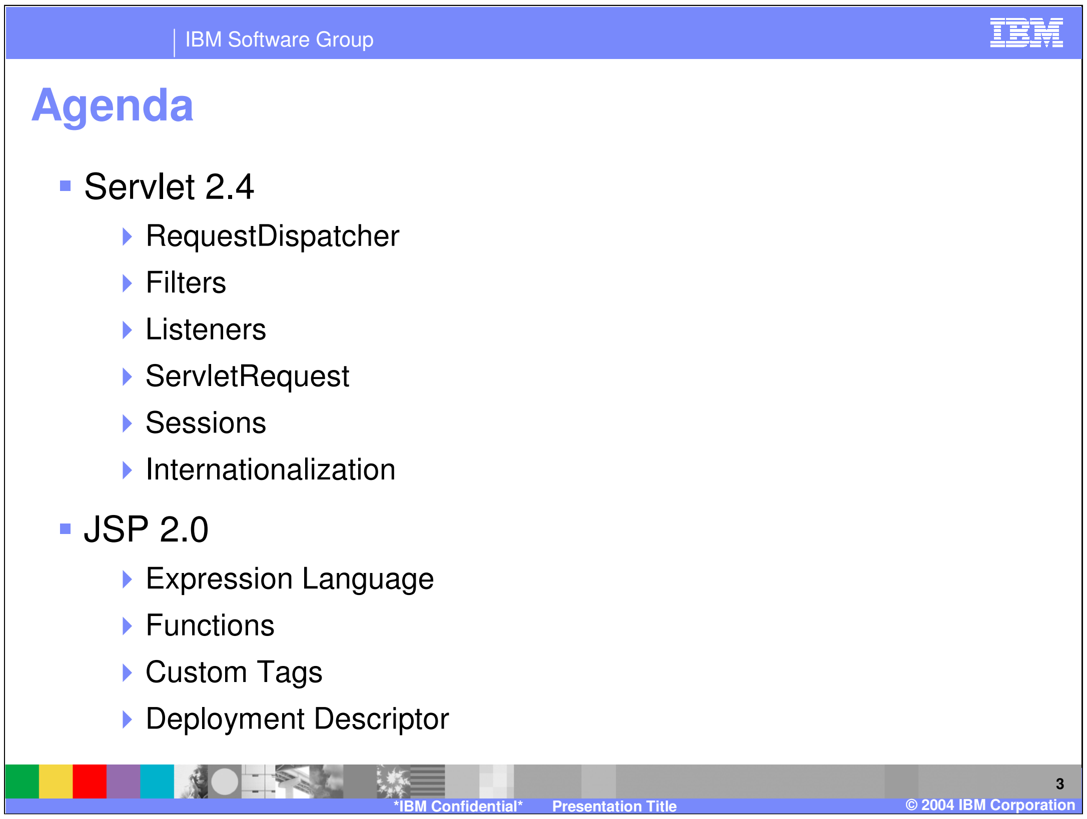
business on demand.

Updated January 24, 2005



## Goals

- Understanding the new features of Serlvet 2.4 and JSP 2.0
  - Servlet 2.4 has a few enhancements and clarifications to the prior version 2.3
  - JSP 2.0 helps to develop more simplified JSP pages as compared to JSP 1.2

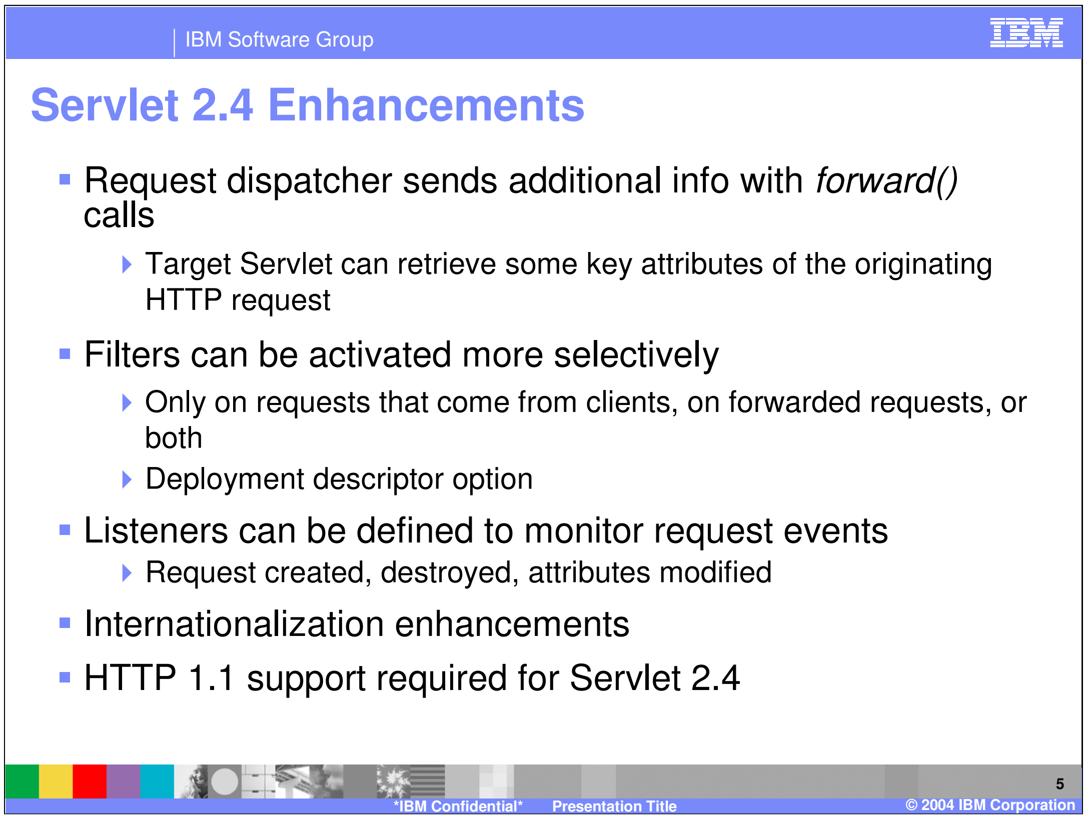


## Agenda

- Servlet 2.4
  - RequestDispatcher
  - Filters
  - Listeners
  - ServletRequest
  - Sessions
  - Internationalization
- JSP 2.0
  - Expression Language
  - Functions
  - Custom Tags
  - Deployment Descriptor

## Section

***Servlet 2.4***


IBM Software Group

## Servlet 2.4 Enhancements

- Request dispatcher sends additional info with *forward()* calls
  - Target Servlet can retrieve some key attributes of the originating HTTP request
- Filters can be activated more selectively
  - Only on requests that come from clients, on forwarded requests, or both
  - Deployment descriptor option
- Listeners can be defined to monitor request events
  - Request created, destroyed, attributes modified
- Internationalization enhancements
- HTTP 1.1 support required for Servlet 2.4

5

*IBM Confidential* Presentation Title © 2004 IBM Corporation

**Request Dispatcher**

Servlet 2.4 adds five new request attributes to provide extra information during a RequestDispatcher forward() call. Servles that have been invoked by another servlet using the forward method of RequestDispatcher, have access to the path of the original request. One exception to this is a servlet obtained by using the getNamedDispatcher method.

**Filters**

This 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. Different combinations of these values can be specified. If you don't specify any <dispatcher> elements, the default is REQUEST. Theres 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>

**Listeners**

Servlet 2.3 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 to observe as requests are created and destroyed, and as attributes are added and removed from a request.

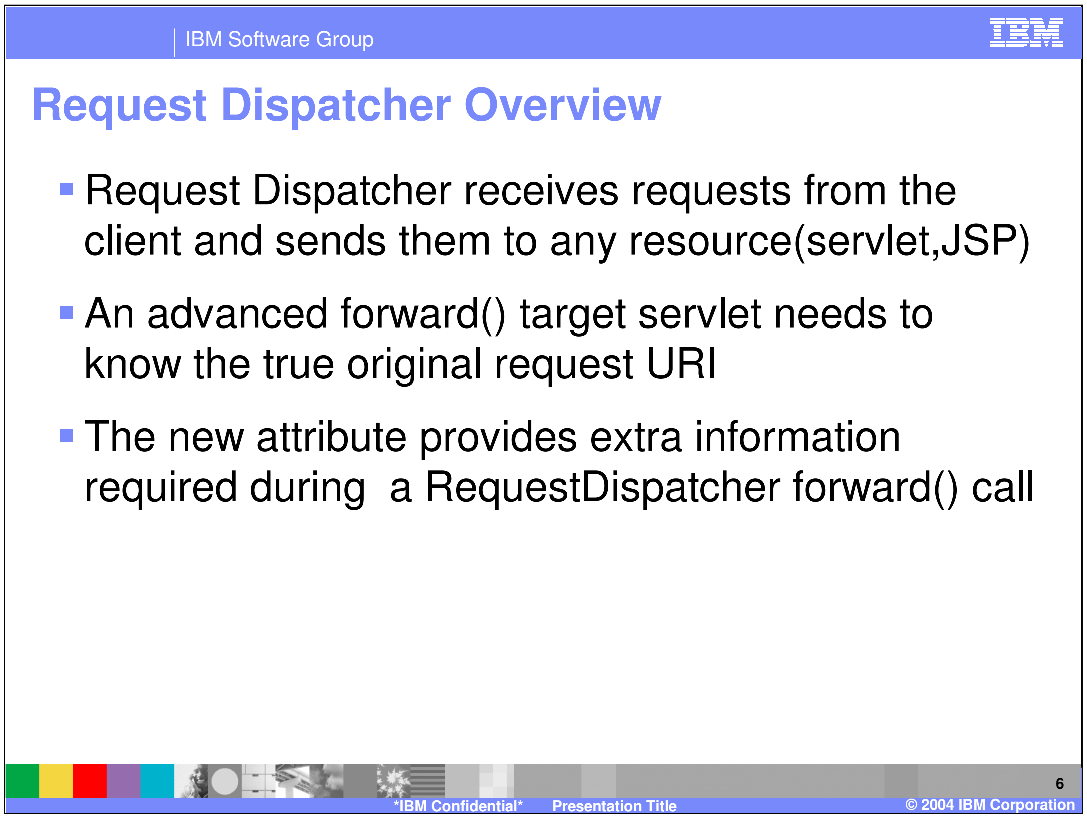
***Internationalization***

Also in Servlet 2.4, the ServletResponse interface (and the ServletResponseWrapper) adds two new methods:

**setCharacterEncoding(String encoding)**

**getContentType()**

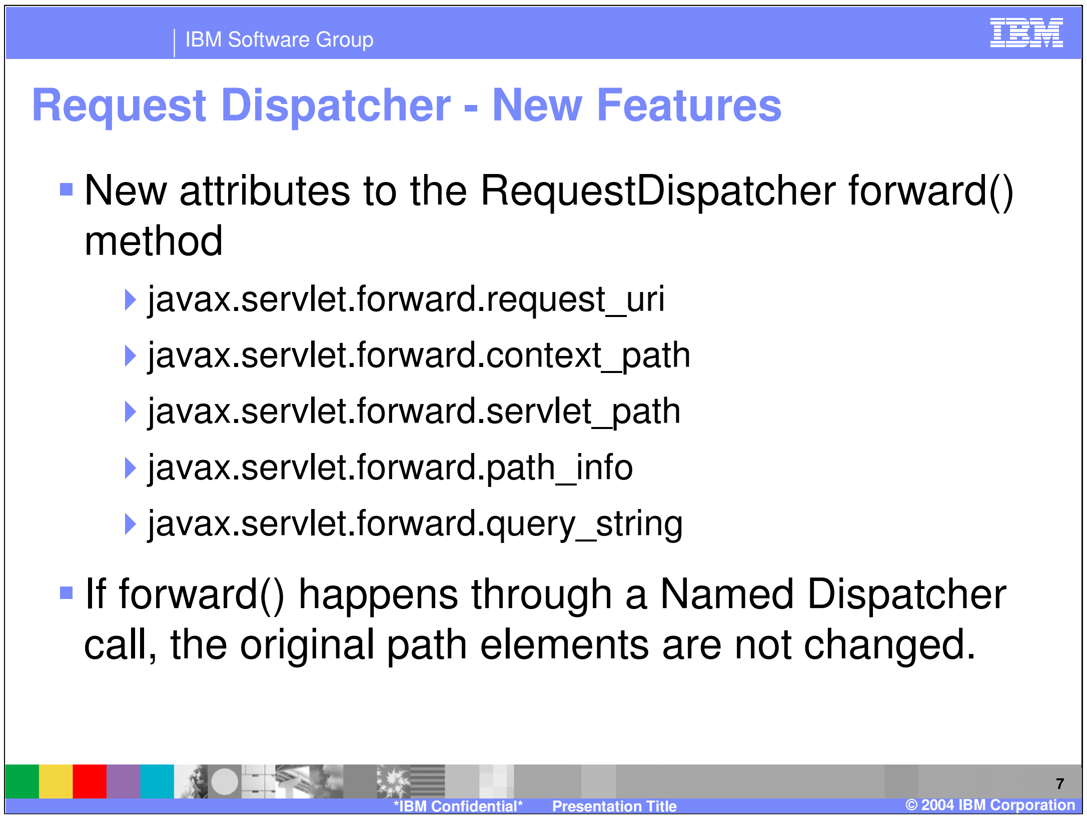
In the servlet set the locale and the character encoding of a response. The locale is set using the ServletResponse.setLocale method, and communicated to the client using the Content-Language header. The character encoding can be set explicitly using the ServletResponse methods setCharacterEncoding and setContentType, or implicitly using the ServletResponse.setLocale method, and is communicated to the client using the charset parameter of the Content-Type header.



## Request Dispatcher Overview

- Request Dispatcher receives requests from the client and sends them to any resource(servlet,JSP)
- An advanced forward() target servlet needs to know the true original request URI
- The new attribute provides extra information required during  a RequestDispatcher forward() call

When you code the forward() method in a servlet, the servlet container changes the target servlet's path environment as if it were the first servlet being invoked. The methods getRequestURI(), getContextPath(), getServletPath(), getPathInfo(), and getQueryString() all return information based on the URI (Uniform Resource Identifier) passed to the getRequestDispatcher() method. However, sometimes an advanced forward() target servlet might like to know the true original request URI. Servlet 2.4 adds five new request attributes to provide extra information during a RequestDispatcher forward() call.

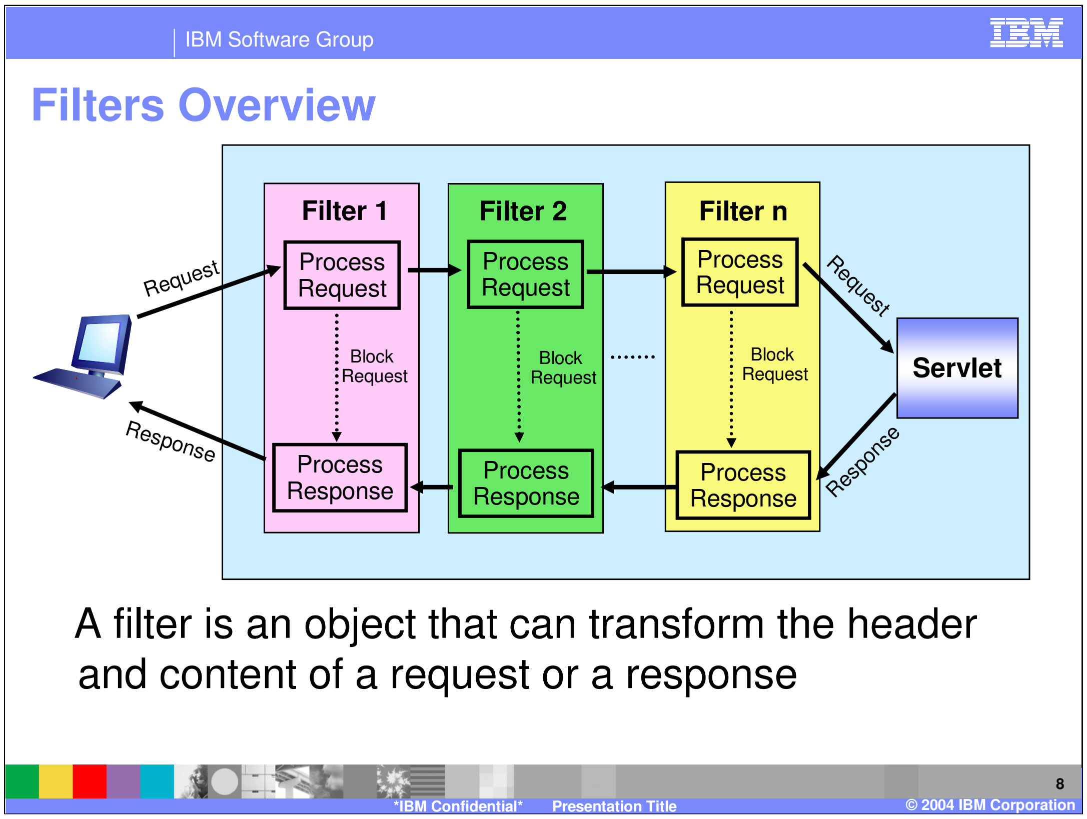


## Request Dispatcher - New Features

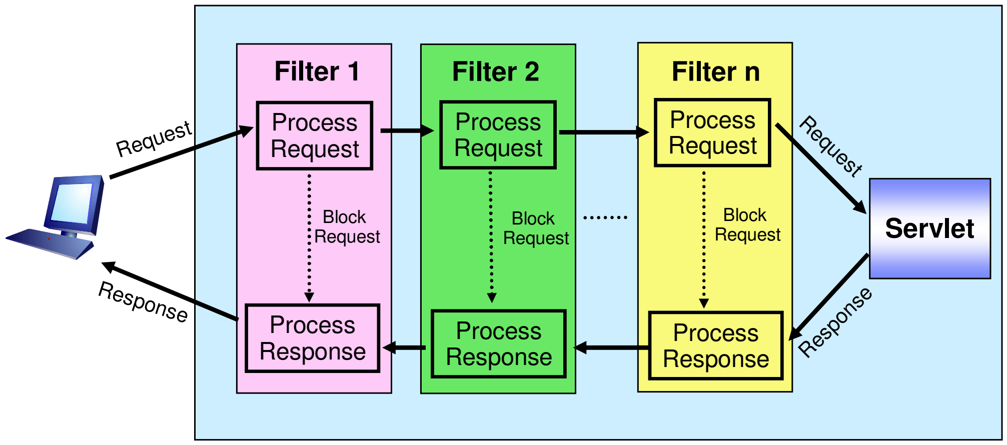
- New attributes to the RequestDispatcher forward() method
  - javax.servlet.forward.request_uri
  - javax.servlet.forward.context_path
  - javax.servlet.forward.servlet_path
  - javax.servlet.forward.path_info
  - javax.servlet.forward.query_string
- If forward() happens through a Named Dispatcher call, the original path elements are not changed.

Inside a forwarded servlet getRequestURI() will return the path to the target servlet, however to get at the original path, in the application you can request request.getAttribute("javax.servlet.forward.request_uri").

If forward() occurs through a getNamedDispatcher() call, the above attributes are not set because the original path elements are not changed.



## Filters Overview



A filter is an object that can transform the header and content of a request or a response

A filter is a reusable piece of code that can transform the content of HTTP requests, responses, and header information. Filters differ from other web components because they do not create their own response. Filters do not generally create a response or respond to a request as servlets do, rather they modify or adapt the requests for a resource, and modify or adapt responses from a resource.

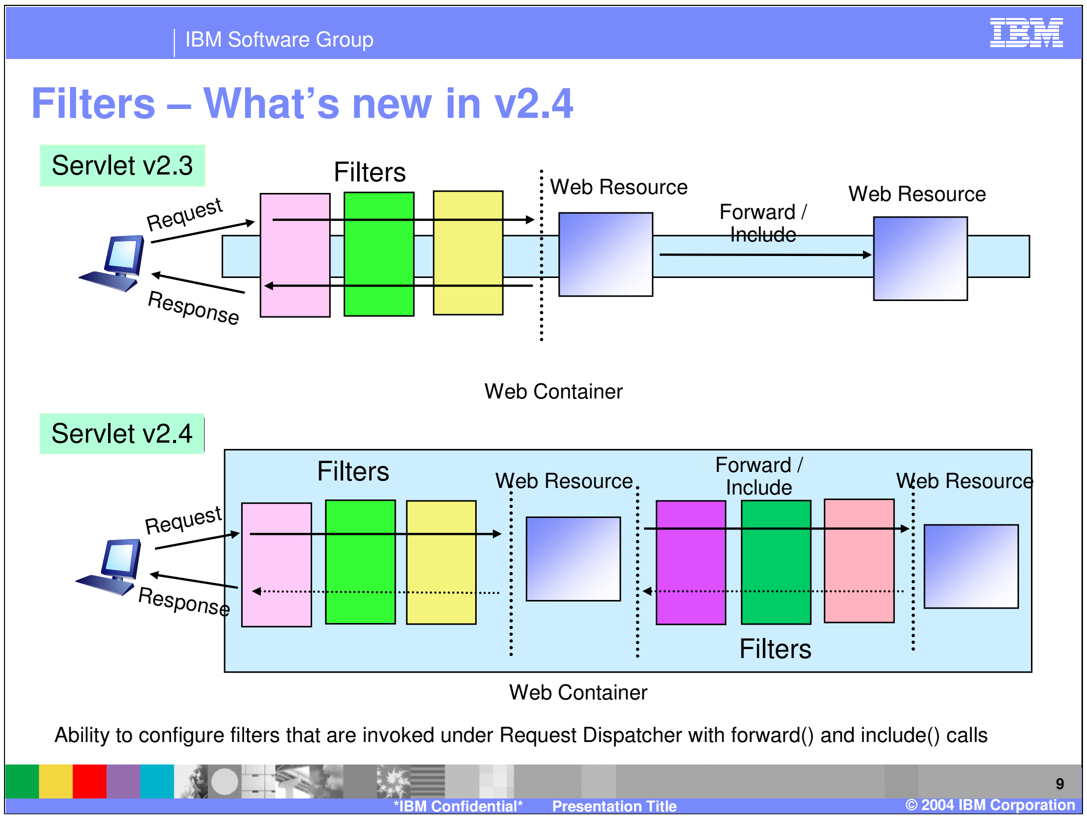
The main tasks that a filter can perform are as follows:

•Modify the request or response

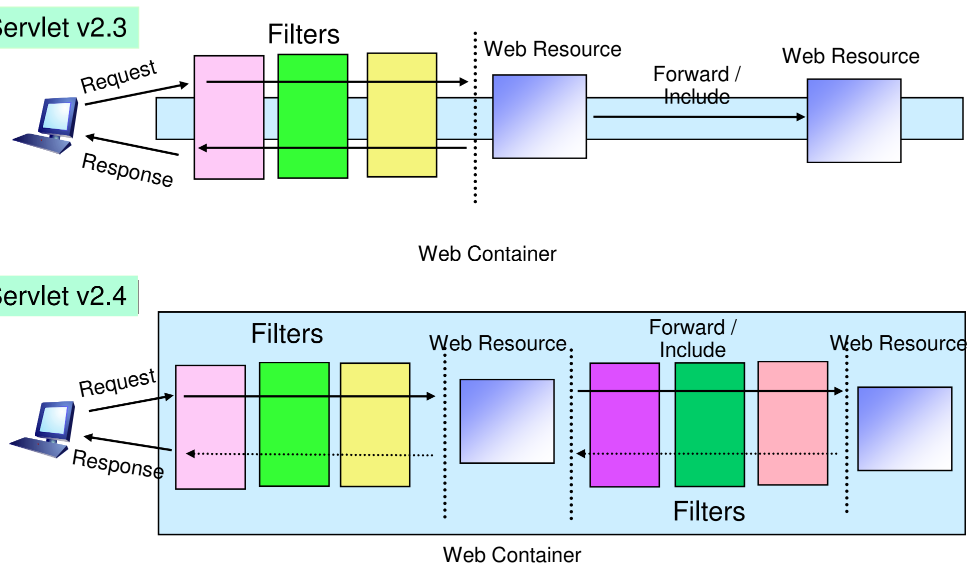
•Block the request and send the response directly

•Interact with external resources.

Applications of filters include authentication, logging, image conversion, data compression, encryption, tokenizing streams, and XML transformations.


IBM Software Group

## Filters – What's new in v2.4



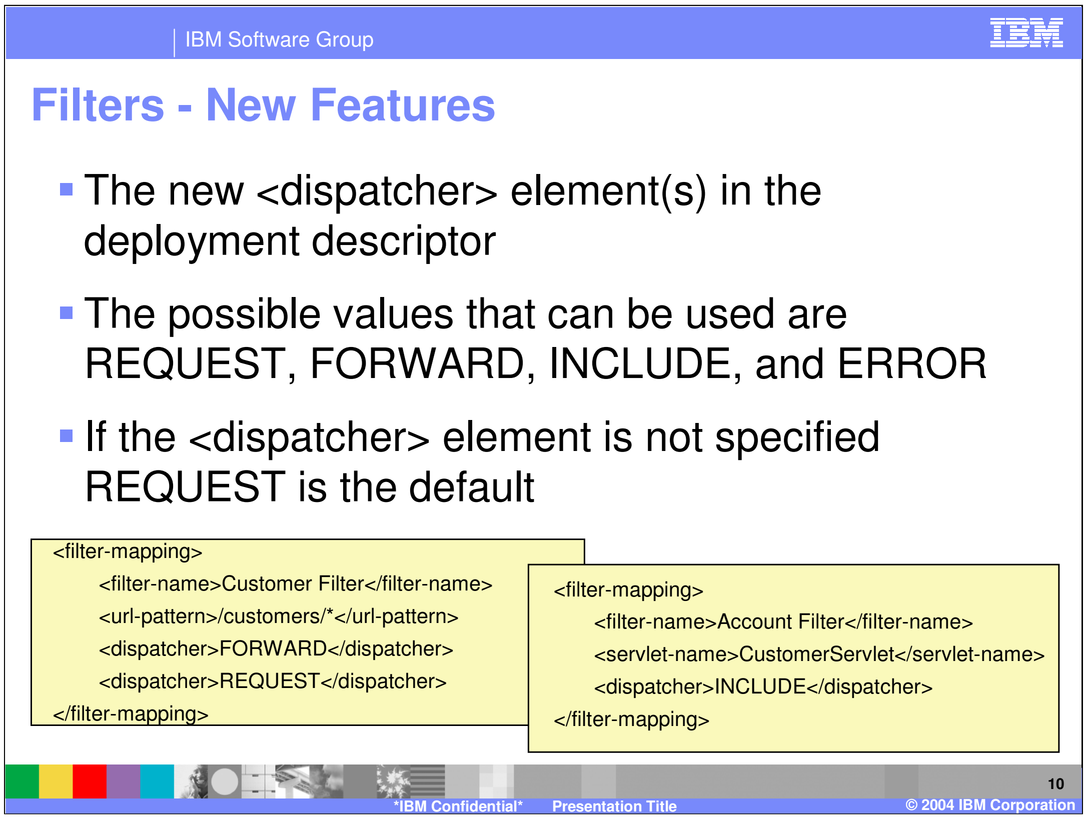
Ability to configure filters that are invoked under Request Dispatcher with forward() and include() calls

9 *IBM Confidential* Presentation Title © 2004 IBM Corporation

Previous versions of the servlet specification did not make it clear whether a filter should be invoked with a RequestDispatcher or whether filters could be invoked using forward() or include() calls.

Servlet 2.4 gives an ability to configure filters that are invoked under Request Dispatcher with forward() and include() calls.

This can be done by using the new <dispatcher> element(s) in the deployment descriptor


IBM Software Group

## Filters - New Features

- The new <dispatcher> element(s) in the deployment descriptor
- The possible values that can be used are REQUEST, FORWARD, INCLUDE, and ERROR
- If the <dispatcher> element is not specified REQUEST is the default

```
<filter-mapping>
        <filter-name>Customer Filter</filter-name>
        <url-pattern>/customers/*</url-pattern>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

```
<filter-mapping>
        <filter-name>Account Filter</filter-name>
        <servlet-name>CustomerServlet</servlet-name>
        <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

10
*IBM Confidential* Presentation Title © 2004 IBM Corporation

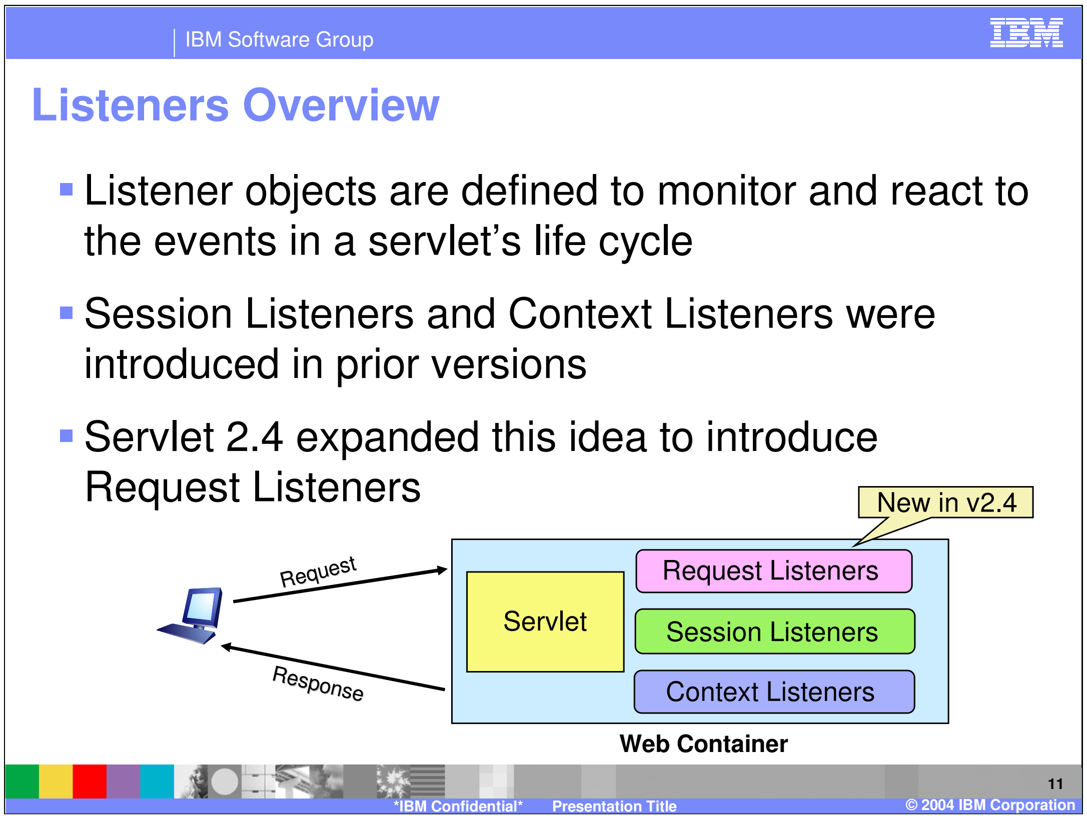
Here is an example of defining filter using the dispatcher elements.

If the request comes directly from the client, it is indicated by a <dispatcher> element with value *REQUEST*, or by the absence of any <dispatcher> elements.

If the request is being processed under a request dispatcher representing the Web component matching the <url-pattern> or <servlet-name> using a forward() call, then it is indicated by a <dispatcher> element with value *FORWARD* or if it is by using an include() call, then the <dispatcher> element will be *INCLUDE.*

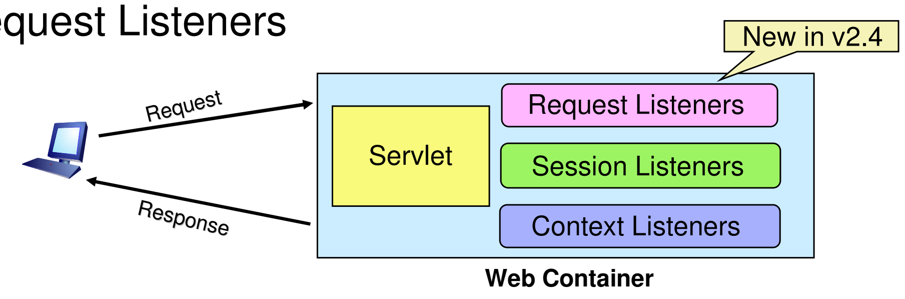
If the request is being processed with the error page mechanism matching the <url-pattern>, then it is indicated by a <dispatcher> element with the value *ERROR.*

The error page mechanism described does not intervene when errors occur when invoked using the RequestDispatcher or filter.doFilter method. In this way, a filter or servlet using the RequestDispatcher has the opportunity to handle errors generated.



## Listeners Overview

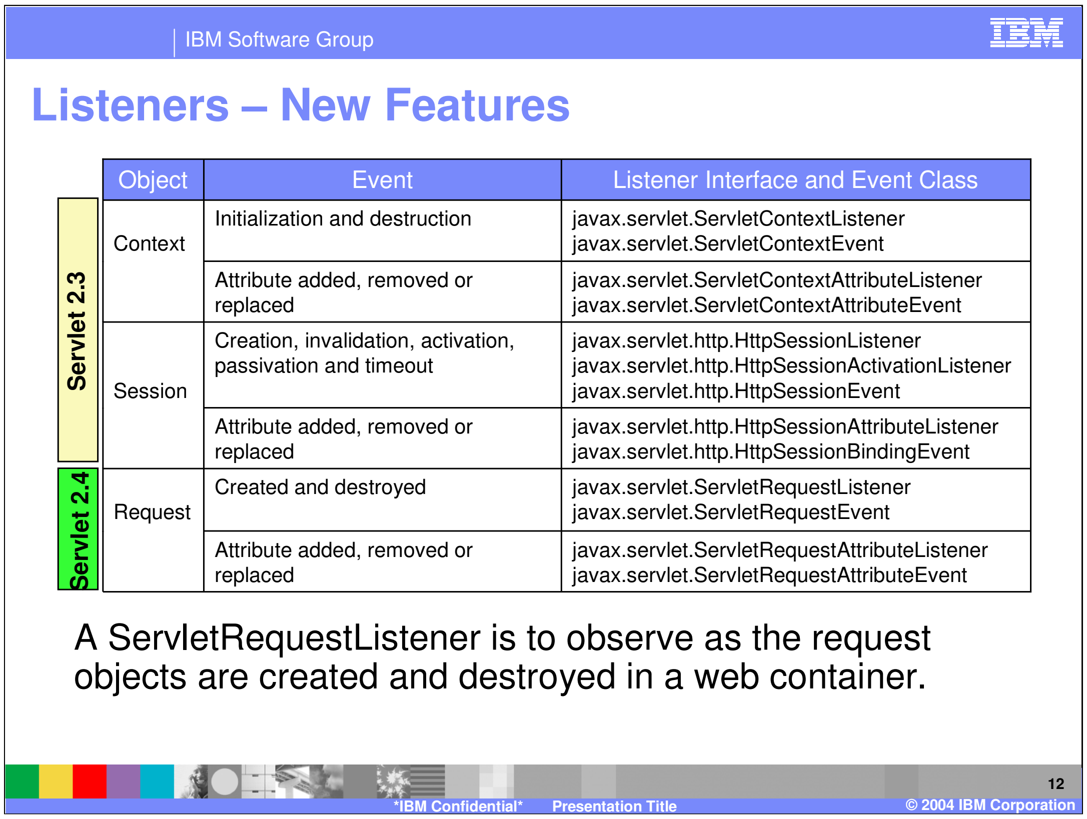
- Listener objects are defined to monitor and react to the events in a servlet's life cycle
- Session Listeners and Context Listeners were introduced in prior versions
- Servlet 2.4 expanded this idea to introduce Request Listeners



Servlet 2.4 expands the idea of context and session listeners to add request listeners, allowing developers to observe as requests are created and destroyed, and as attributes are added and removed from a request.

When a listener method is invoked, it is passed an event that contains information appropriate to the event. For Example: Methods in the HttpSessionListener interface are passed an HttpSessionEvent, which contains an HttpSession.

ServletRequestListener can be used to track the request start and end.



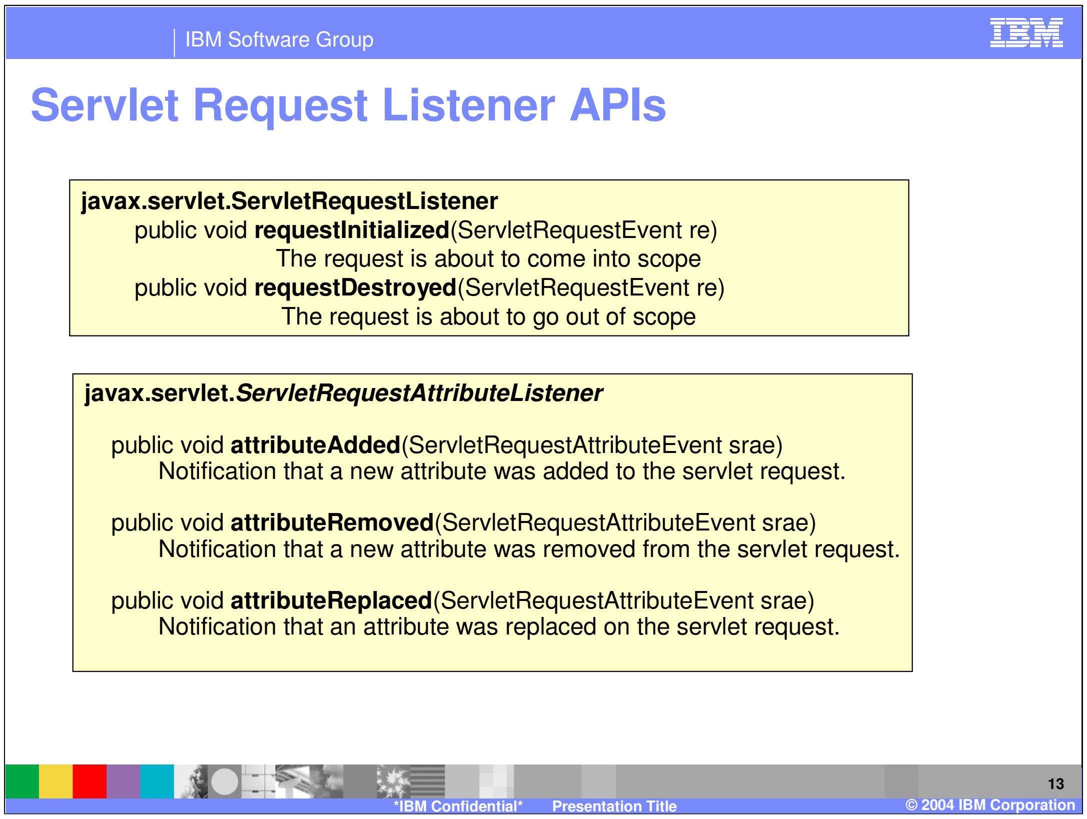
## Listeners – New Features

| | Object | Event | Listener Interface and Event Class |
|---|---|---|---|
| Servlet 2.3 | Context | Initialization and destruction | javax.servlet.ServletContextListener javax.servlet.ServletContextEvent |
| | | Attribute added, removed or replaced | javax.servlet.ServletContextAttributeListener javax.servlet.ServletContextAttributeEvent |
| | Session | Creation, invalidation, activation, passivation and timeout | javax.servlet.http.HttpSessionListener javax.servlet.http.HttpSessionActivationListener javax.servlet.http.HttpSessionEvent |
| | | Attribute added, removed or replaced | javax.servlet.http.HttpSessionAttributeListener javax.servlet.http.HttpSessionBindingEvent |
| Servlet 2.4 | Request | Created and destroyed | javax.servlet.ServletRequestListener javax.servlet.ServletRequestEvent |
| | | Attribute added, removed or replaced | javax.servlet.ServletRequestAttributeListener javax.servlet.ServletRequestAttributeEvent |

A ServletRequestListener is to observe as the request objects are created and destroyed in a web container.

A ServletRequestListener can be implemented by the developer interested in getting notified of requests coming in and out of scope in a web component. A request is defined as coming into scope when it is about to enter the first filter in the Filter chain that will process it, and as going out of scope when it exits the last filter in its filter chain.

A ServletRequestAttributeListener can be implemented by the developer interested in being notified of request attribute changes. Notifications will be generated while the request is within the scope of the web application in which the listener is registered. A request is defined as coming into scope when it is about to enter the first servlet or filter in each web application, as going out of scope when it exits the last servlet or the first filter in the chain.



## Servlet Request Listener APIs

**javax.servlet.ServletRequestListener**

public void **requestInitialized**(ServletRequestEvent re)

The request is about to come into scope

public void **requestDestroyed**(ServletRequestEvent re)

The request is about to go out of scope

**javax.servlet.*ServletRequestAttributeListener***

public void **attributeAdded**(ServletRequestAttributeEvent srae)

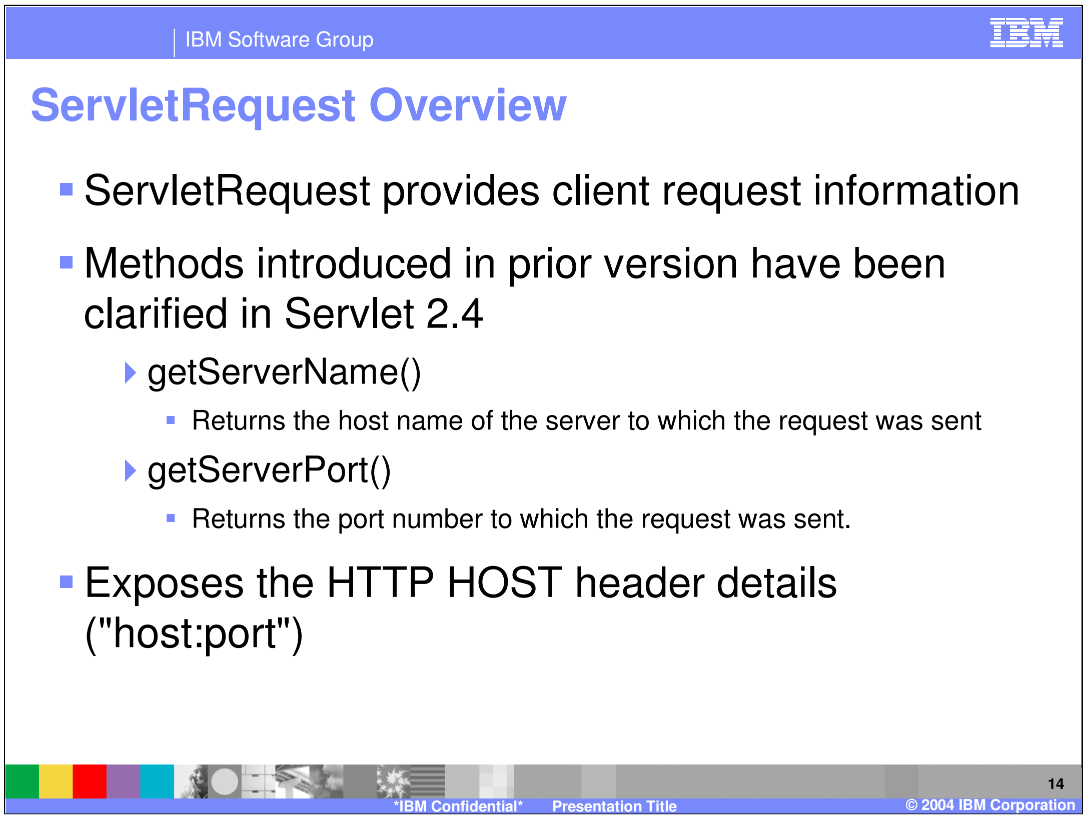
Notification that a new attribute was added to the servlet request.

public void **attributeRemoved**(ServletRequestAttributeEvent srae)

Notification that a new attribute was removed from the servlet request.

public void **attributeReplaced**(ServletRequestAttributeEvent srae)

Notification that an attribute was replaced on the servlet request.


IBM Software Group

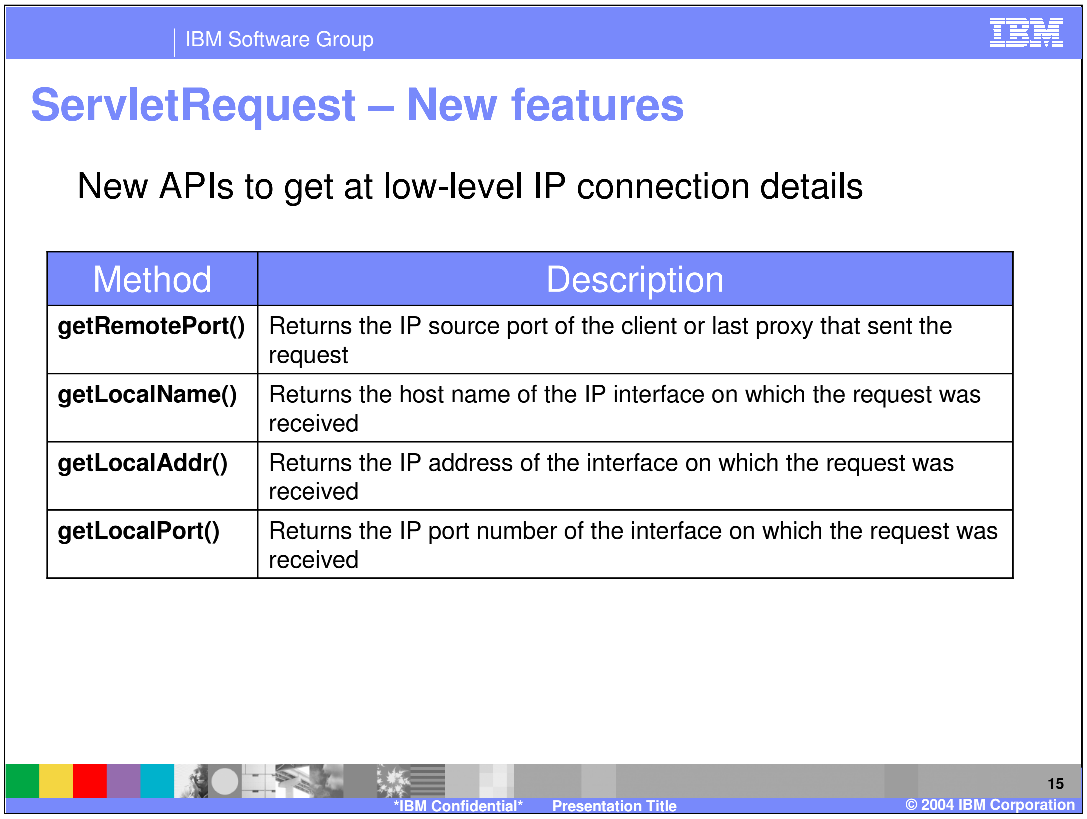
## ServletRequest Overview

- ServletRequest provides client request information
- Methods introduced in prior version have been clarified in Servlet 2.4
  - getServerName()
    - Returns the host name of the server to which the request was sent
  - getServerPort()
    - Returns the port number to which the request was sent.
- Exposes the HTTP HOST header details ("host:port")

14

*IBM Confidential* Presentation Title © 2004 IBM Corporation

The preexisting methods getServerName() and getServerPort() have been clarified to expose the HTTP HOST header details ("host:port") . getServerName( ) will return host part of the header and getServerPort( ) will return port in host header.



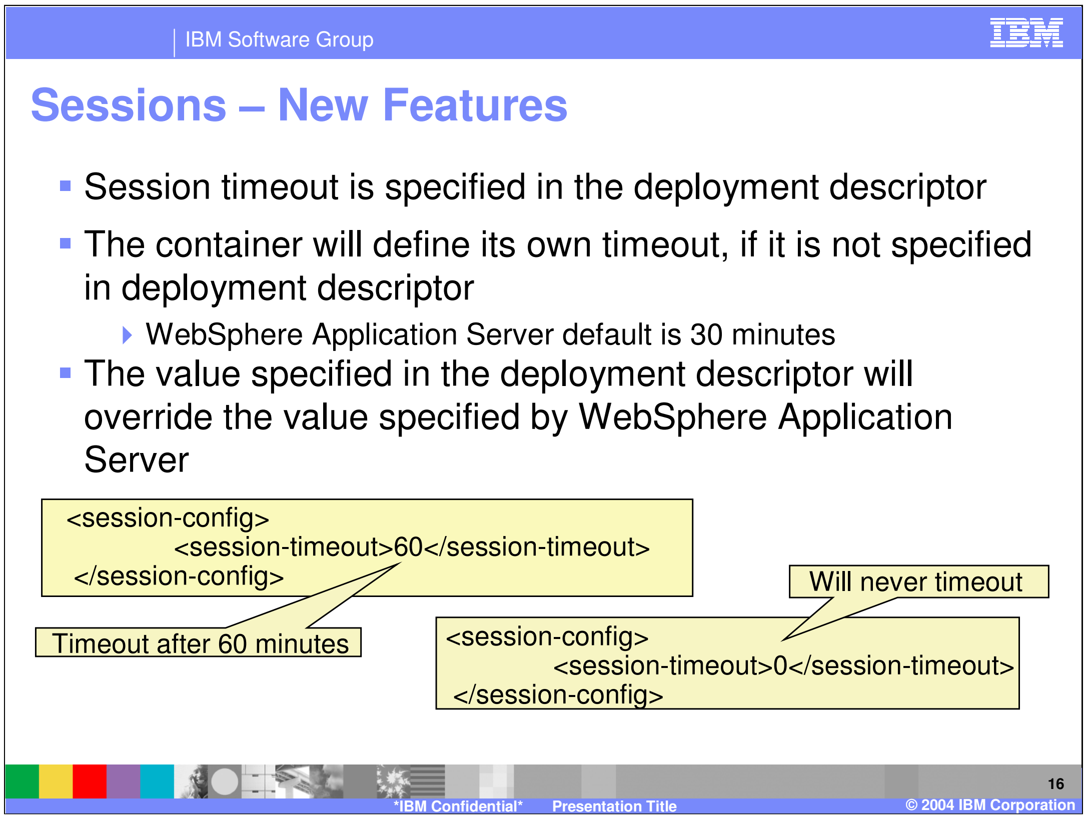
## ServletRequest – New features

New APIs to get at low-level IP connection details

| Method | Description |
|---|---|
| **getRemotePort()** | Returns the IP source port of the client or last proxy that sent the request |
| **getLocalName()** | Returns the host name of the IP interface on which the request was received |
| **getLocalAddr()** | Returns the IP address of the interface on which the request was received |
| **getLocalPort()** | Returns the IP port number of the interface on which the request was received |

With the combination of existing methods, these new methods provide a way to get at low-level IP connection details and helps determine how the connection got routed.

The getRemotePort() method, combined with the preexisting getRemoteAddr() and getRemoteHost() methods, exposes the client side of the IP connections. The new getLocalPort(), getLocalAddr(), and getLocalName methods exposes the IP's from which the request was received.



## Sessions – New Features

- Session timeout is specified in the deployment descriptor
- The container will define its own timeout, if it is not specified in deployment descriptor
  - WebSphere Application Server default is 30 minutes
- The value specified in the deployment descriptor will override the value specified by WebSphere Application Server

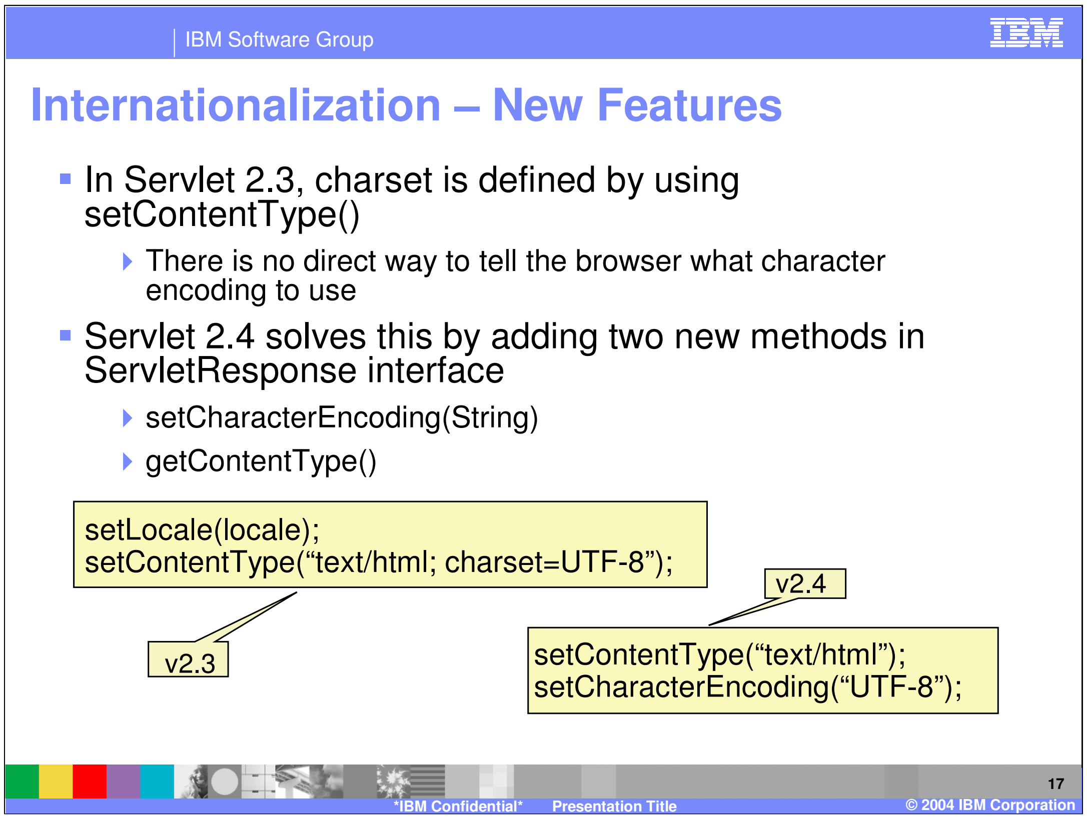
```
<session-config>
        <session-timeout>60</session-timeout>
</session-config>
```

Timeout after 60 minutes

Will never timeout

```
<session-config>
        <session-timeout>0</session-timeout>
</session-config>
```

Avoid setting a session timeout avoid setting it too low. Ensure that the user has ample time to complete the online forms. In the example, Session will be timed out after 60 minutes


IBM Software Group

## Internationalization – New Features

- In Servlet 2.3, charset is defined by using setContentType()
  - There is no direct way to tell the browser what character encoding to use
- Servlet 2.4 solves this by adding two new methods in ServletResponse interface
  - setCharacterEncoding(String)
  - getContentType()

```
setLocale(locale);
setContentType(“text/html; charset=UTF-8”);
```

v2.3

v2.4

```
setContentType(“text/html”);
setCharacterEncoding(“UTF-8”);
```

17
*IBM Confidential* Presentation Title © 2004 IBM Corporation

In prior versions, the ServletRequest had these methods but in Servlet 2.4, the ServletResponse interface adds two new methods:

•setCharacterEncoding(String)

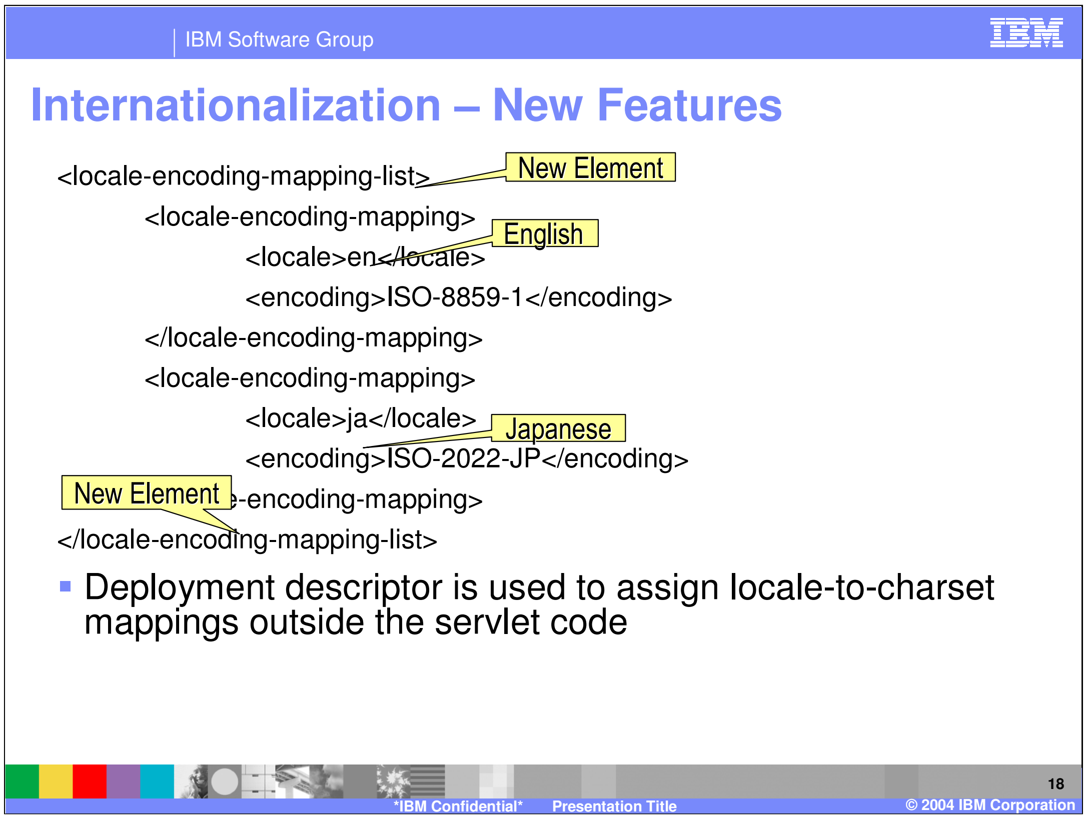
•getContentType()

**setCharacterEncoding(String c)** – To set the response's character encoding. This method provides an alternative to passing charset in setContentType(String) or passing a Locale to setLocale(Locale). One can now avoid setting of charset via setContentType( )call.

For example, application can avoid setting the charset via the awkward setContentType("text/html; *charset=UTF-8*") call. Application can do setContentType("text/html”); followed by setCharaterEncoding(“UTF-8”) or a setLocale(); This method can be called repeatedly to change the character encoding. This method has no effect if it is called after getWriter has been called or after the response has been committed.

**getContentType( ) -** Returns the response's content type. The content type proper must have been specified using setContentType() before the response is committed. This will include a charset parameter set by either setContentType(), setLocale(), or setCharacterEncoding(). If no content type has been specified, this method returns null.

The other preexisting methods are setContentType(String), getCharacterEncoding()



## Internationalization – New Features

```
<locale-encoding-mapping-list>
        <locale-encoding-mapping>
                 <locale>en</locale>
                 <encoding>ISO-8859-1</encoding>
        </locale-encoding-mapping>
        <locale-encoding-mapping>
                 <locale>ja</locale>
                 <encoding>ISO-2022-JP</encoding>
        </locale-encoding-mapping>
</locale-encoding-mapping-list>
```

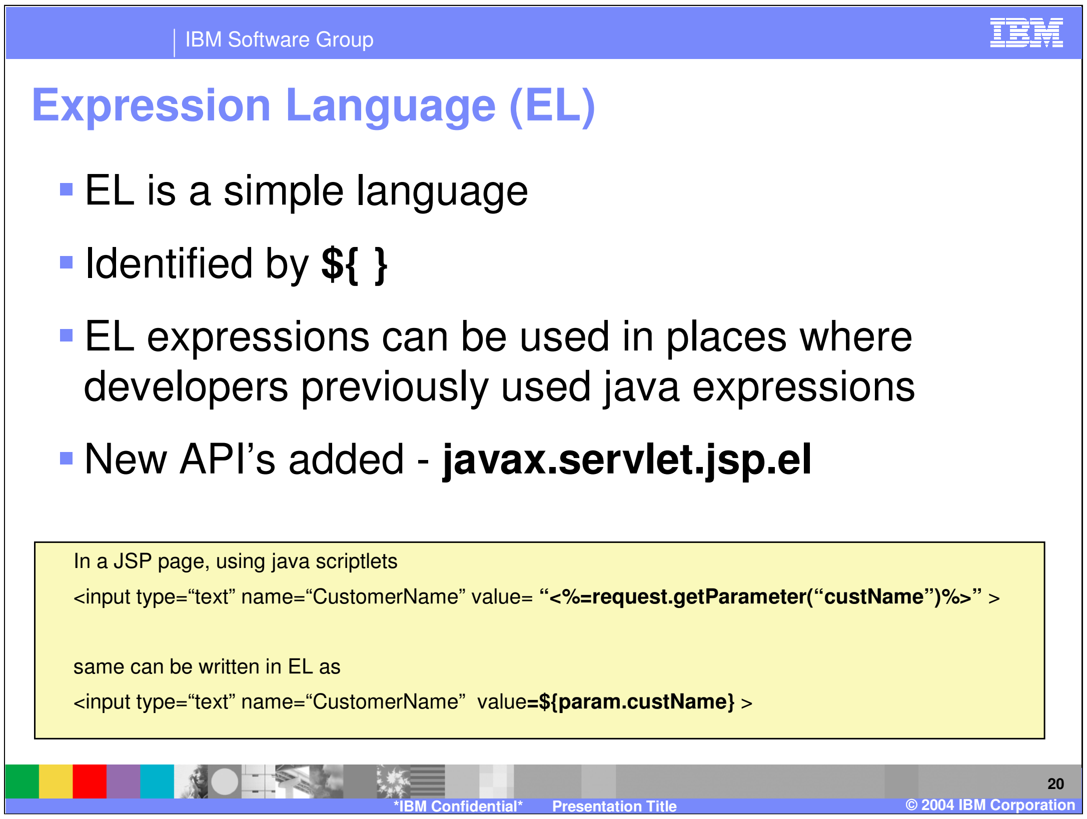
New Element
English
Japanese
New Element

- Deployment descriptor is used to assign locale-to-charset mappings outside the servlet code

A new <locale -encoding-mapping-list> element in the deployment descriptor is to let the deployer assign locale-to-charset mappings outside the servlet code.Default is English ISO-8859-1

## Section

**_JSP 2.0_**


IBM Software Group

## Expression Language (EL)

- EL is a simple language
- Identified by **${ }**
- EL expressions can be used in places where developers previously used java expressions
- New API's added - **javax.servlet.jsp.el**

In a JSP page, using java scriptlets

<input type="text" name="CustomerName" value= **"<%=request.getParameter("custName")%>"** >

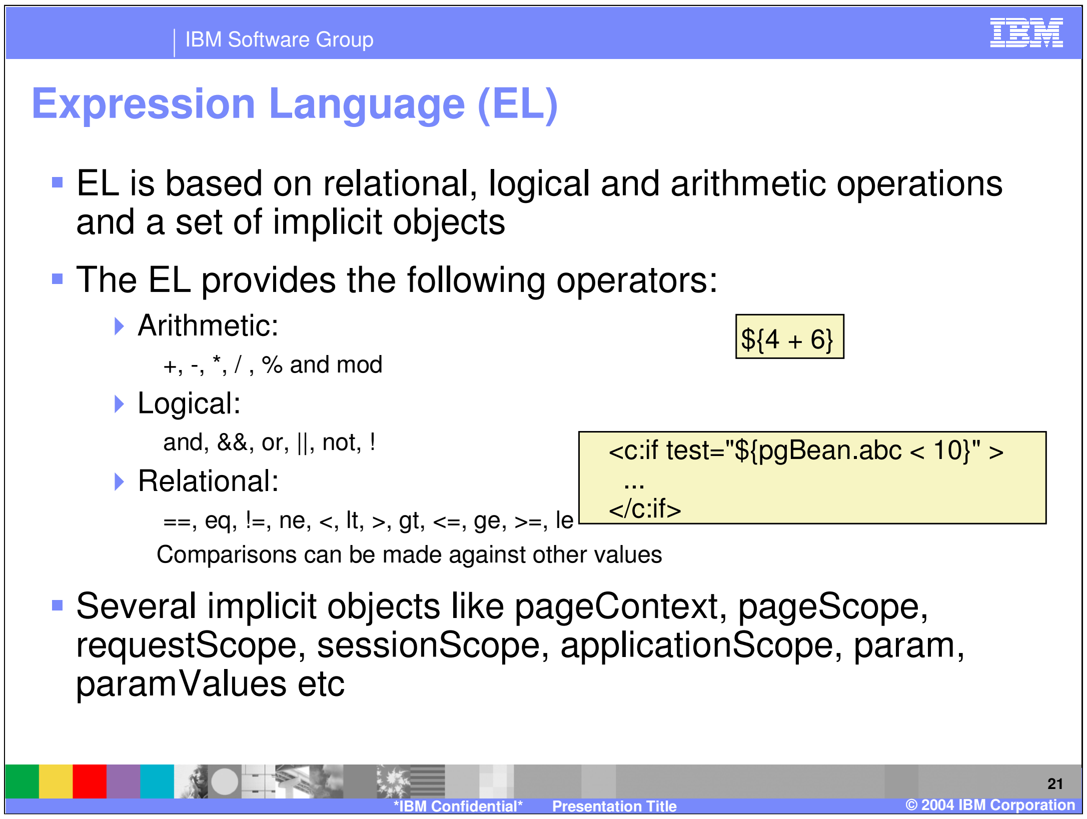
same can be written in EL as

<input type="text" name="CustomerName" value**=${param.custName}** >

20

*IBM Confidential* Presentation Title © 2004 IBM Corporation

Expression Language (EL) was initially defined by the Java Server Pages Standard Tag Library (JSTL) 1.0 specification, but now it is incorporated in the JSP 2.0 specification. Expression Language is a simple language.


IBM Software Group

## Expression Language (EL)

- EL is based on relational, logical and arithmetic operations and a set of implicit objects
- The EL provides the following operators:
  - Arithmetic:
    - +, -, *, / , % and mod
  - Logical:
    - and, &&, or, ||, not, !
  - Relational:
    - ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le
    - Comparisons can be made against other values
- Several implicit objects like pageContext, pageScope, requestScope, sessionScope, applicationScope, param, paramValues etc

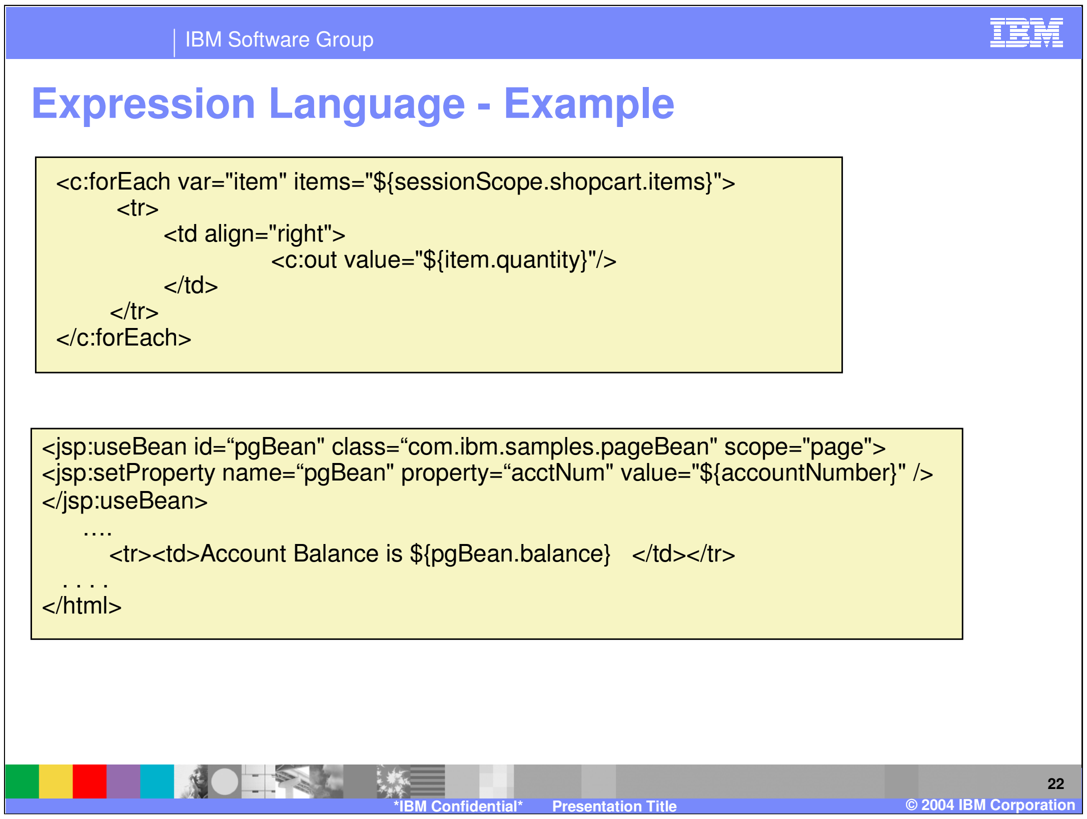
```
${4 + 6}
```

```
<c:if test="${pgBean.abc < 10}" >
  ...
</c:if>
```

21

*IBM Confidential* Presentation Title © 2004 IBM Corporation

EL can be used in
•Relational operations like `==`, `eq`, `!=`, `ne`, `<`, `lt`, `>`, `gt`, `<=`, `ge`, `>=`, `le`.
•Logical operations like `and`, `&&`, `or`, `||`, `not`, `!`
•Arithmetic operations like `+`, `-` (binary), `*`, `/` and `div`, `%` and `mod`, `-`(unary)
•Several implicit objects like pageContext, pageScope, requestScope, sessionScope, applicationScope, param, paramValues etc.


IBM Software Group

## Expression Language - Example

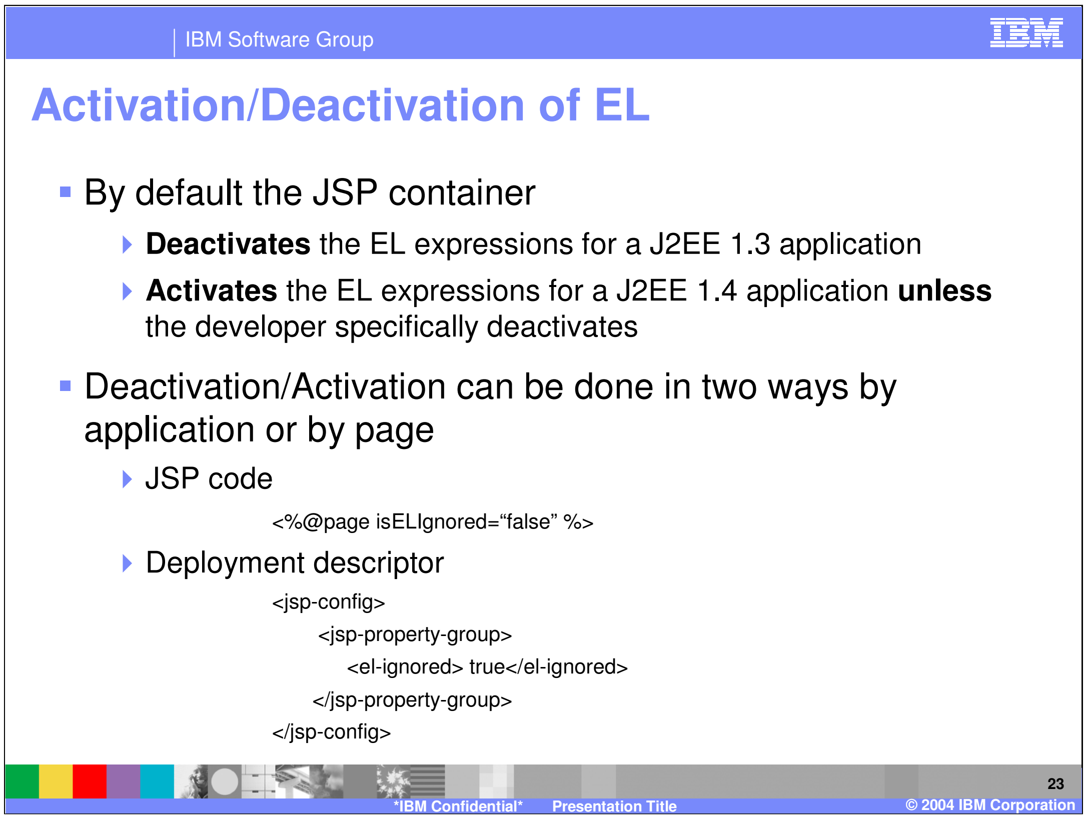
```
<c:forEach var="item" items="${sessionScope.shopcart.items}">
     <tr>
          <td align="right">
                    <c:out value="${item.quantity}"/>
          </td>
     </tr>
</c:forEach>
```

```
<jsp:useBean id=“pgBean" class=“com.ibm.samples.pageBean" scope="page">
<jsp:setProperty name=“pgBean" property=“acctNum" value="${accountNumber}" />
</jsp:useBean>
    ….
      <tr><td>Account Balance is ${pgBean.balance}   </td></tr>
  . . . .
</html>
```

22

*IBM Confidential*  Presentation Title  © 2004 IBM Corporation

An expression language makes it possible to easily access application data stored in JavaBeans components.



## Activation/Deactivation of EL

- By default the JSP container
  - **Deactivates** the EL expressions for a J2EE 1.3 application
  - **Activates** the EL expressions for a J2EE 1.4 application **unless** the developer specifically deactivates
- Deactivation/Activation can be done in two ways by application or by page
  - JSP code

```
<%@page isELIgnored=“false” %>
```

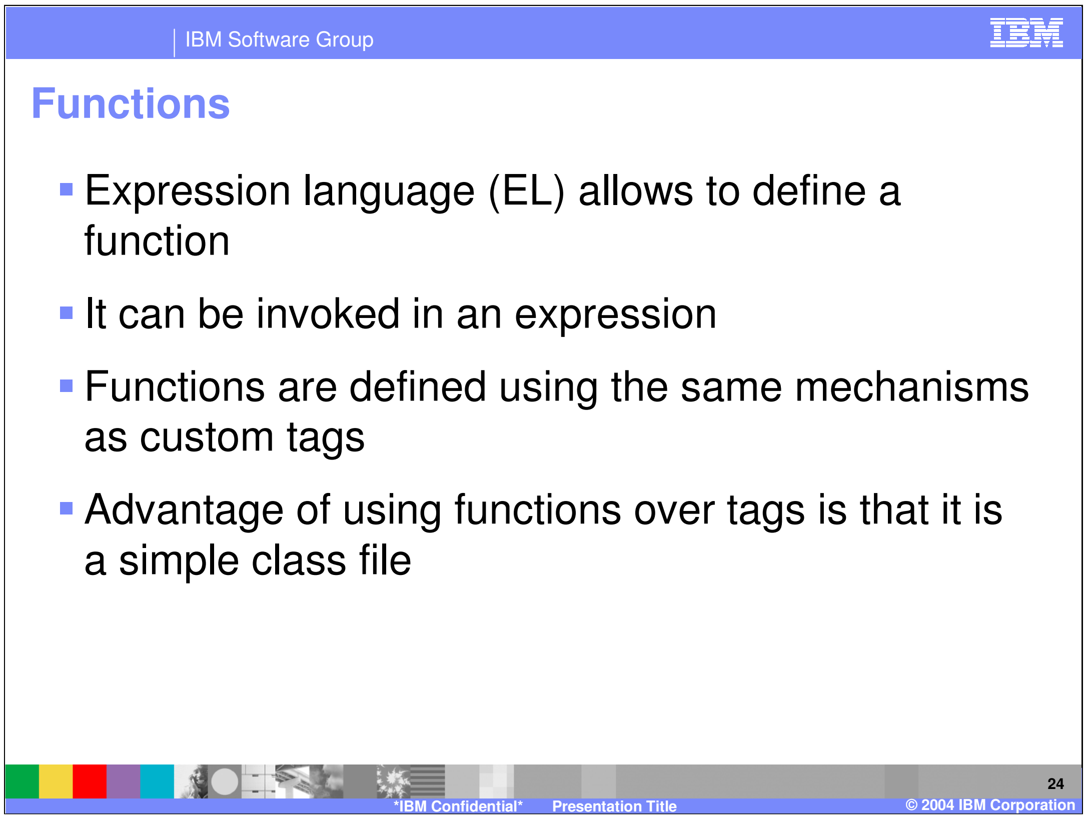
  - Deployment descriptor

```
<jsp-config>
      <jsp-property-group>
          <el-ignored> true</el-ignored>
     </jsp-property-group>
</jsp-config>
```

For a J2EE 1.3 application the JSP container treats the $ sign as a character and not as an Expression Language but if it is J2EE 1.4 application it treats  the $ sign as an Expression Language. In the next slide you will see how to manually deactivate the expression language.

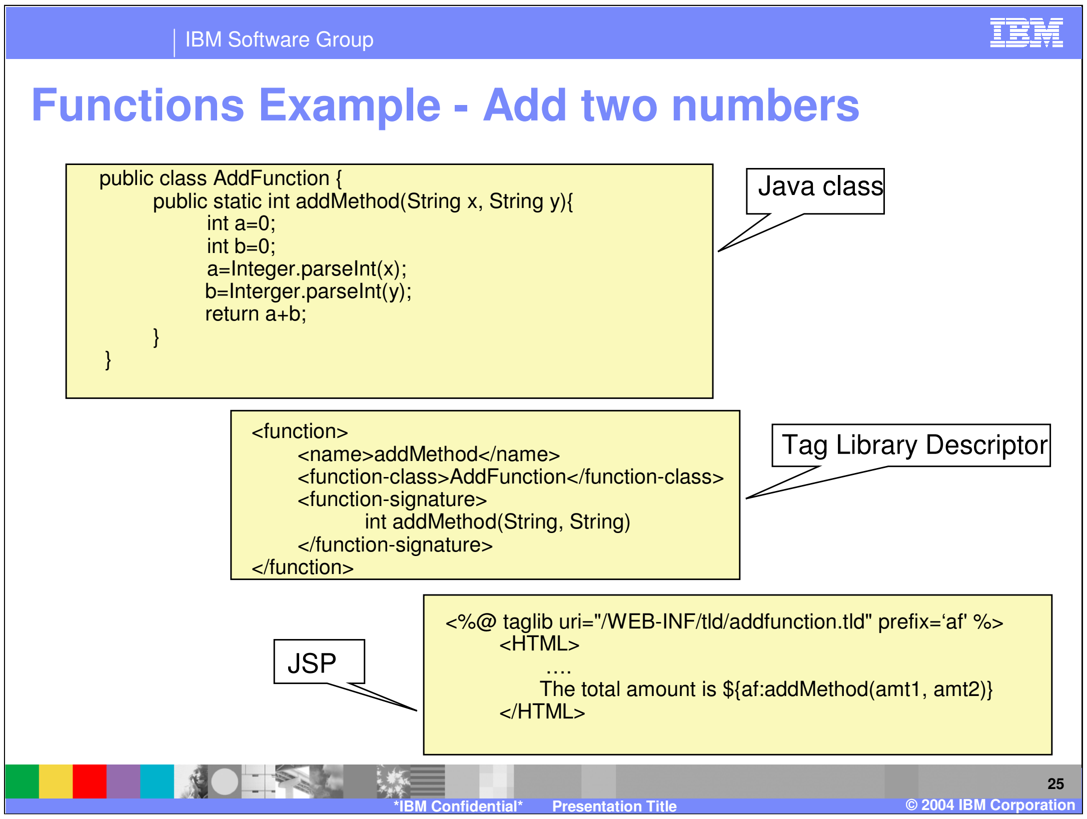
In a JSP page one can specify whether to activate or deactivate an expression language. This can be done only for a J2EE 1.4 application.

The value in the JSP overrides the value in the deployment descriptor



## Functions

- Expression language (EL) allows to define a function
- It can be invoked in an expression
- Functions are defined using the same mechanisms as custom tags
- Advantage of using functions over tags is that it is a simple class file



IBM Software Group

## Functions Example - Add two numbers

Java class

```
public class AddFunction {
        public static int addMethod(String x, String y){
                int a=0;
                int b=0;
                a=Integer.parseInt(x);
                b=Interger.parseInt(y);
                return a+b;
        }
 }
```

Tag Library Descriptor

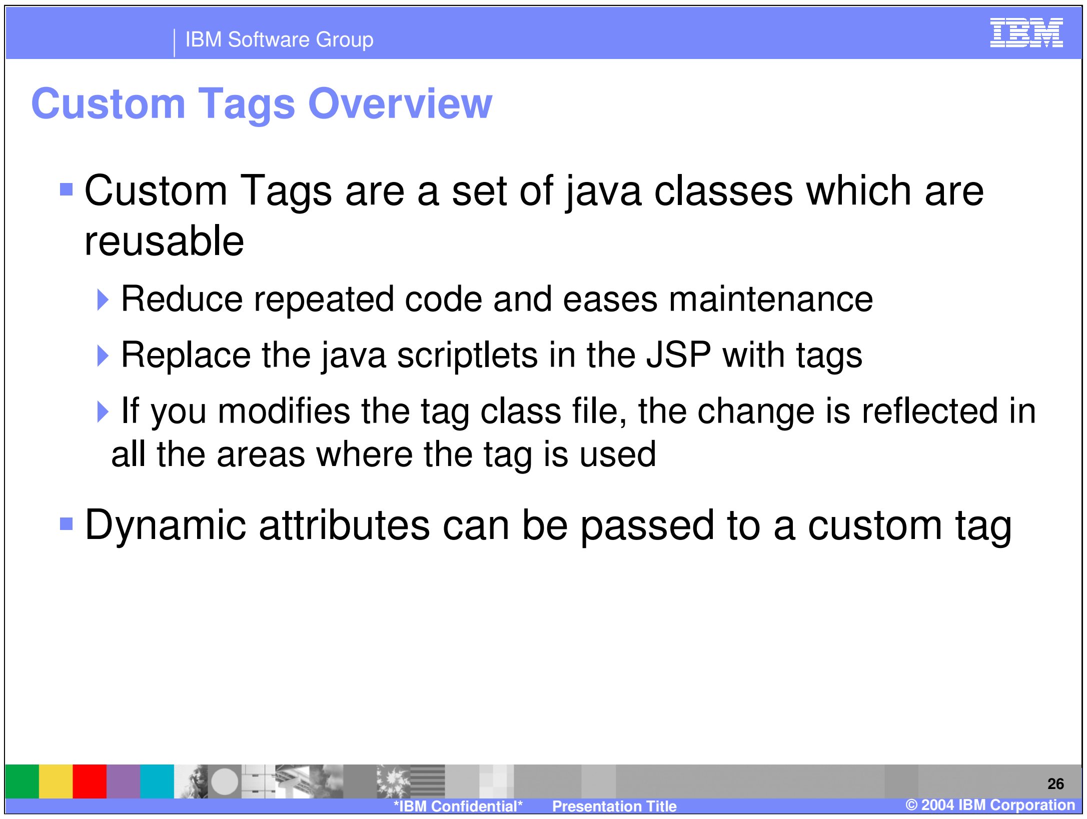
```
<function>
        <name>addMethod</name>
        <function-class>AddFunction</function-class>
        <function-signature>
                int addMethod(String, String)
        </function-signature>
</function>
```

JSP

```
<%@ taglib uri="/WEB-INF/tld/addfunction.tld" prefix=‘af' %>
        <HTML>
                ….
                The total amount is ${af:addMethod(amt1, amt2)}
        </HTML>
```

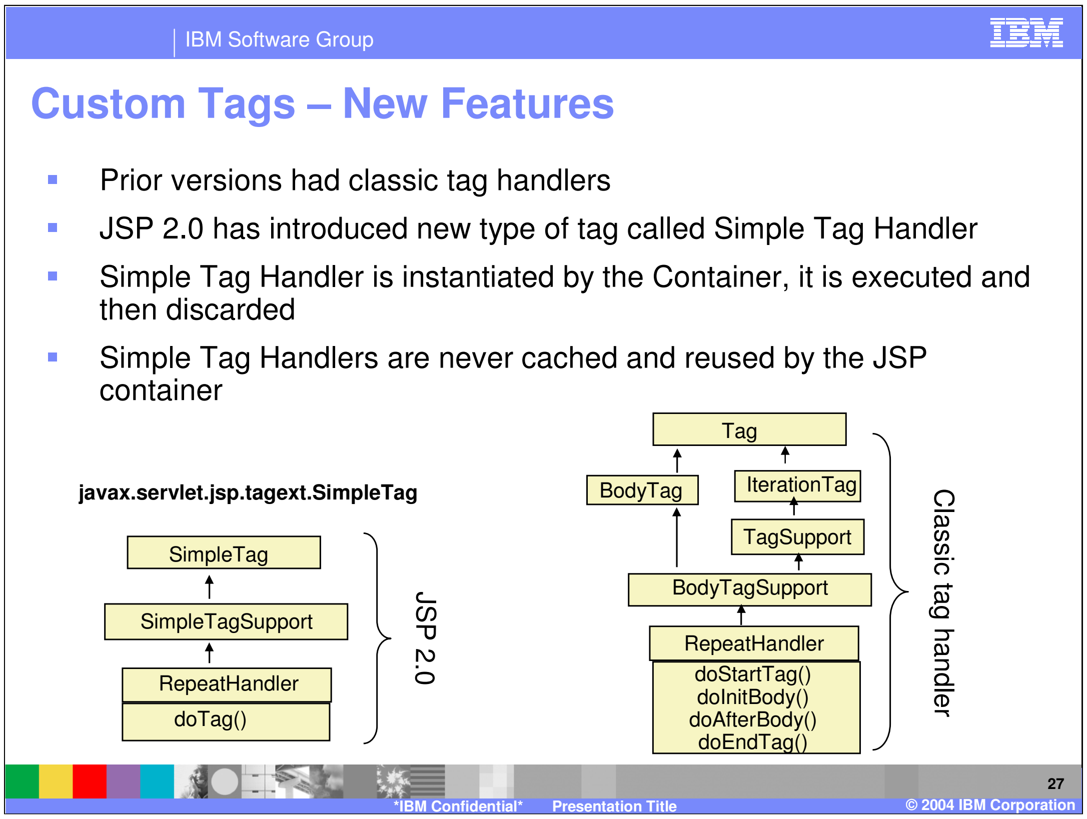
25

*IBM Confidential* Presentation Title © 2004 IBM Corporation



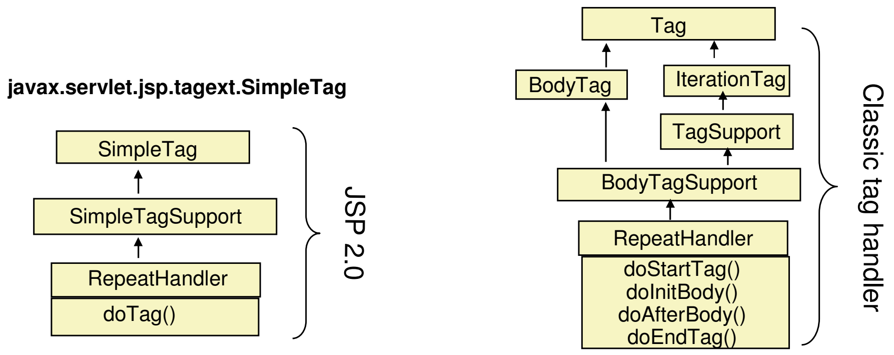
## Custom Tags Overview

- Custom Tags are a set of java classes which are reusable
  - Reduce repeated code and eases maintenance
  - Replace the java scriptlets in the JSP with tags
  - If you modifies the tag class file, the change is reflected in all the areas where the tag is used
- Dynamic attributes can be passed to a custom tag



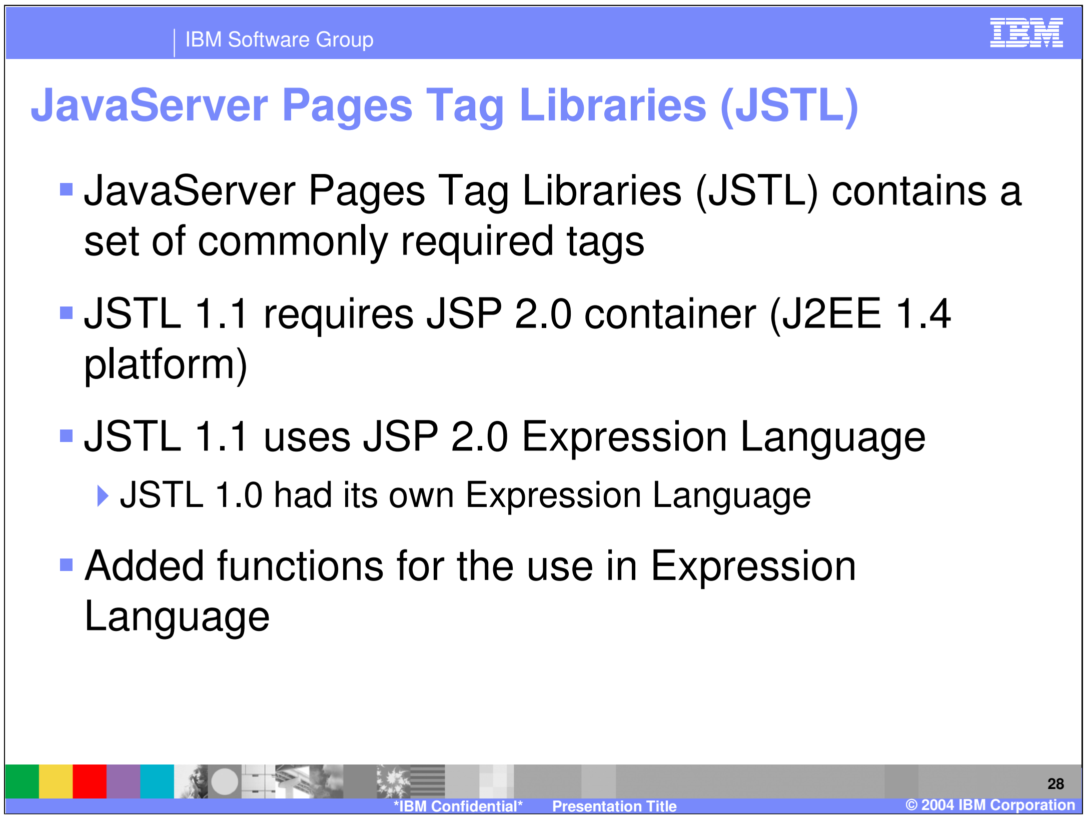
## Custom Tags – New Features

- Prior versions had classic tag handlers
- JSP 2.0 has introduced new type of tag called Simple Tag Handler
- Simple Tag Handler is instantiated by the Container, it is executed and then discarded
- Simple Tag Handlers are never cached and reused by the JSP container



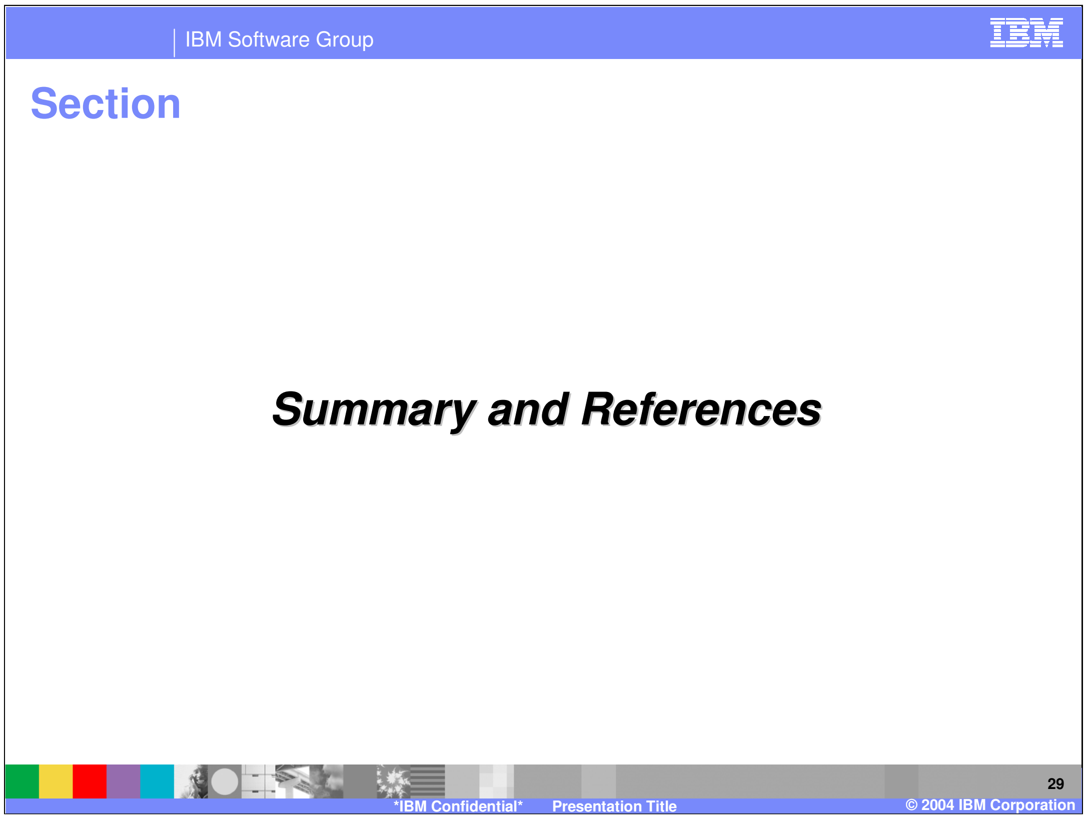
Simple Tag Handlers differ from Classic Tag Handlers in that instead of supporting doStartTag() and doEndTag(), the SimpleTag interface provides a simple doTag() method, which is called once and only once for any given tag invocation.

All tag logic, iteration, body evaluations, etc. are to be performed in this single method. Thus, simple tag handlers have the equivalent power of BodyTag, but with a much simpler lifecycle and interface. The setters for each attribute defined for this tag are called by the container.



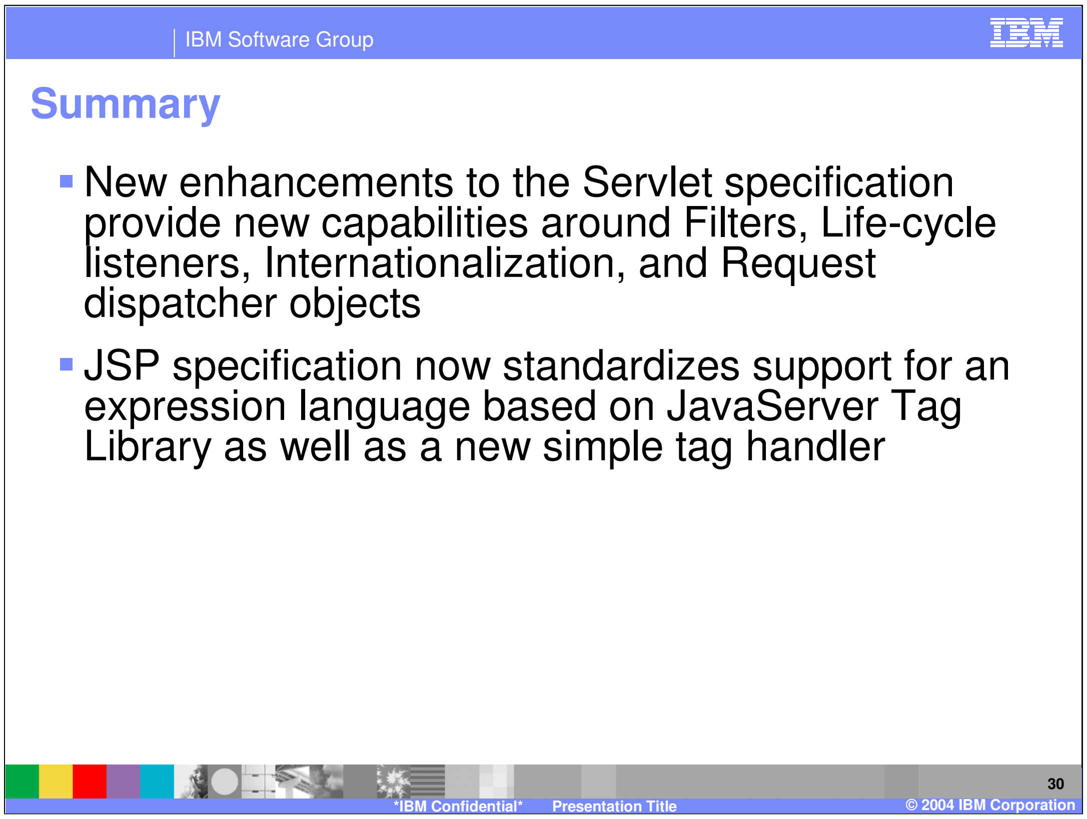
## JavaServer Pages Tag Libraries (JSTL)

- JavaServer Pages Tag Libraries (JSTL) contains a set of commonly required tags
- JSTL 1.1 requires JSP 2.0 container (J2EE 1.4 platform)
- JSTL 1.1 uses JSP 2.0 Expression Language
  - JSTL 1.0 had its own Expression Language
- Added functions for the use in Expression Language



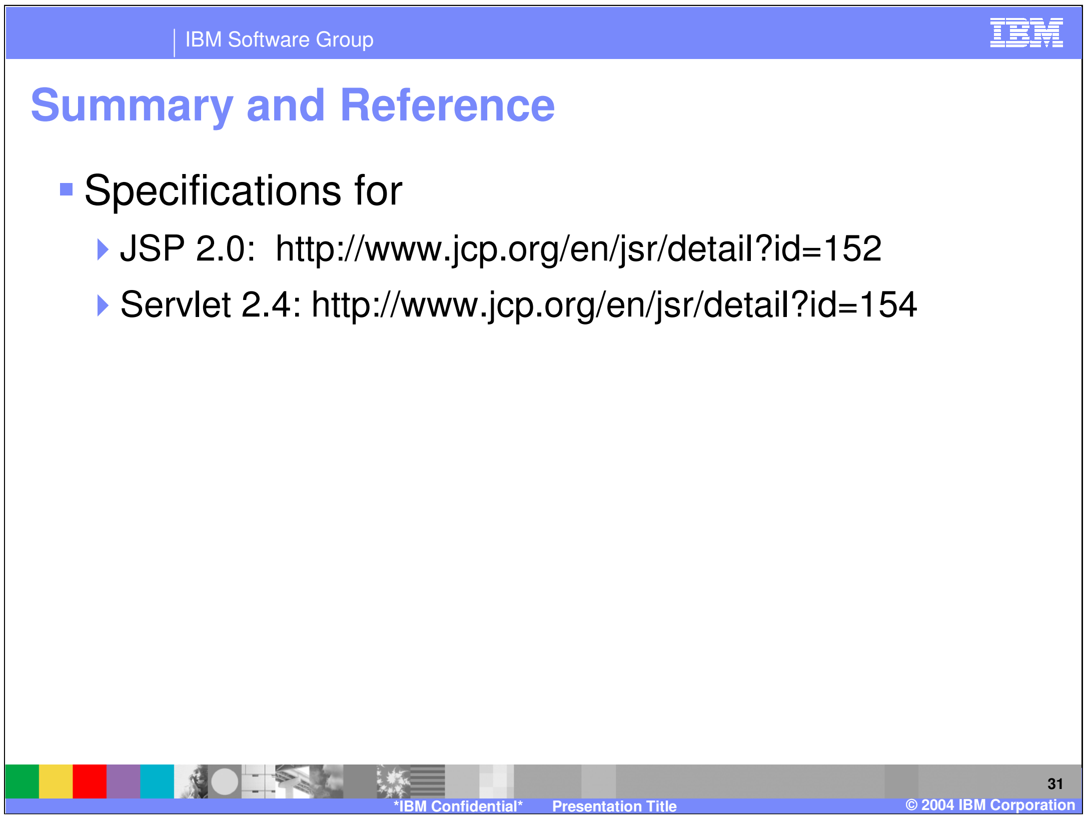
## Section

***Summary and References***



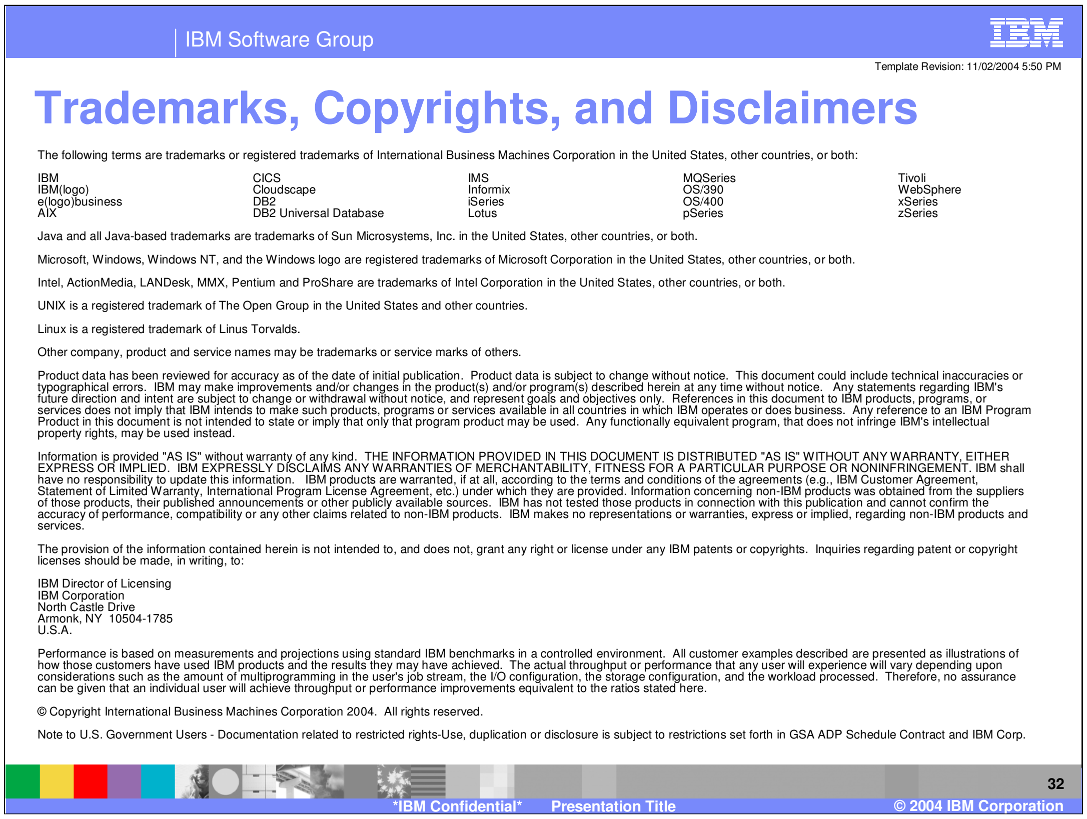
## Summary

- New enhancements to the Servlet specification provide new capabilities around Filters, Life-cycle listeners, Internationalization, and Request dispatcher objects
- JSP specification now standardizes support for an expression language based on JavaServer Tag Library as well as a new simple tag handler



## Summary and Reference

- Specifications for
  - JSP 2.0:  http://www.jcp.org/en/jsr/detail?id=152
  - Servlet 2.4: http://www.jcp.org/en/jsr/detail?id=154



Template Revision: 11/02/2004 5:50 PM

## Trademarks, Copyrights, and Disclaimers