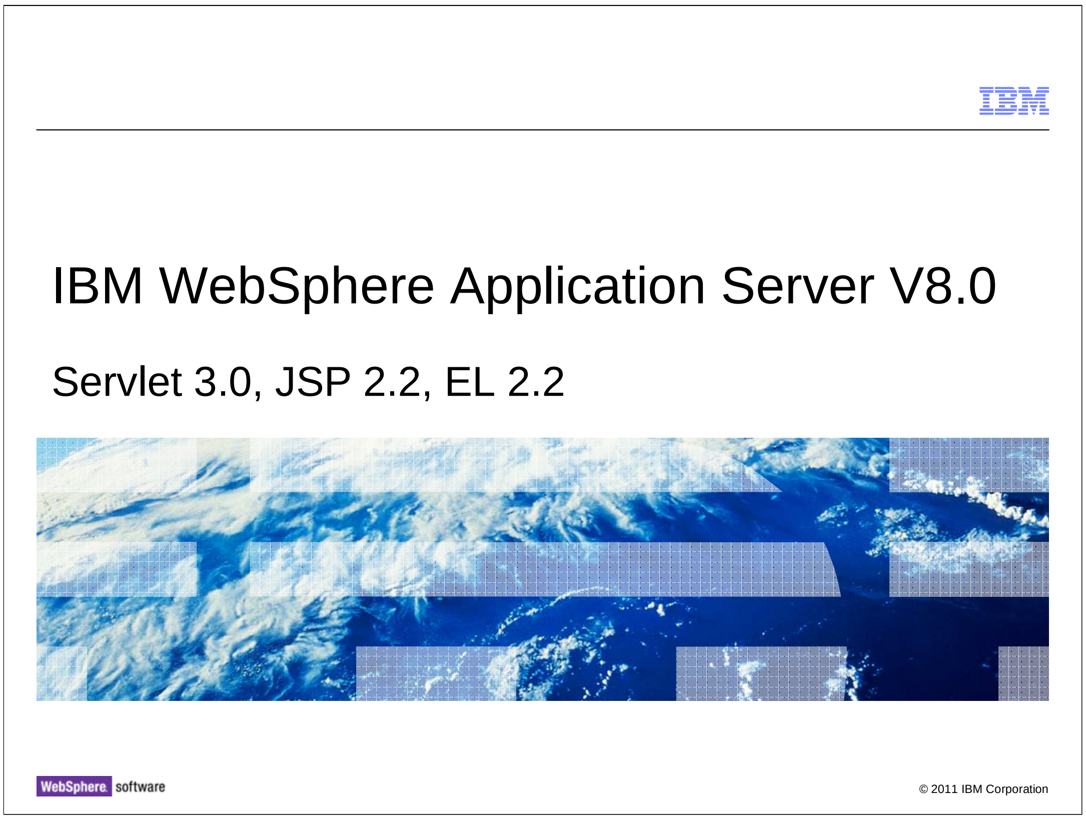


IBM WebSphere Application Server V8.0

Servlet 3.0, JSP 2.2, EL 2.2

This presentation describes support for Servlet 3.0, JavaServer Pages 2.2, and Expression Language 2.2 which are all included in IBM WebSphere Application Server V8.

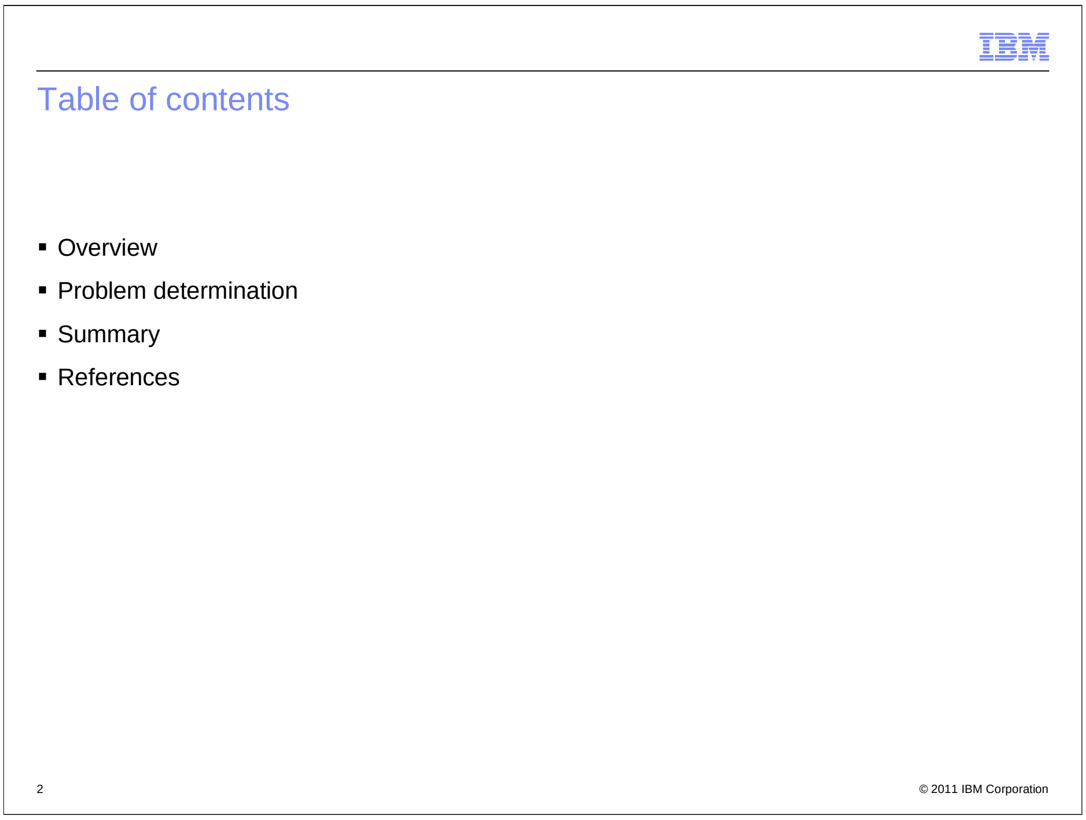


Table of contents

2

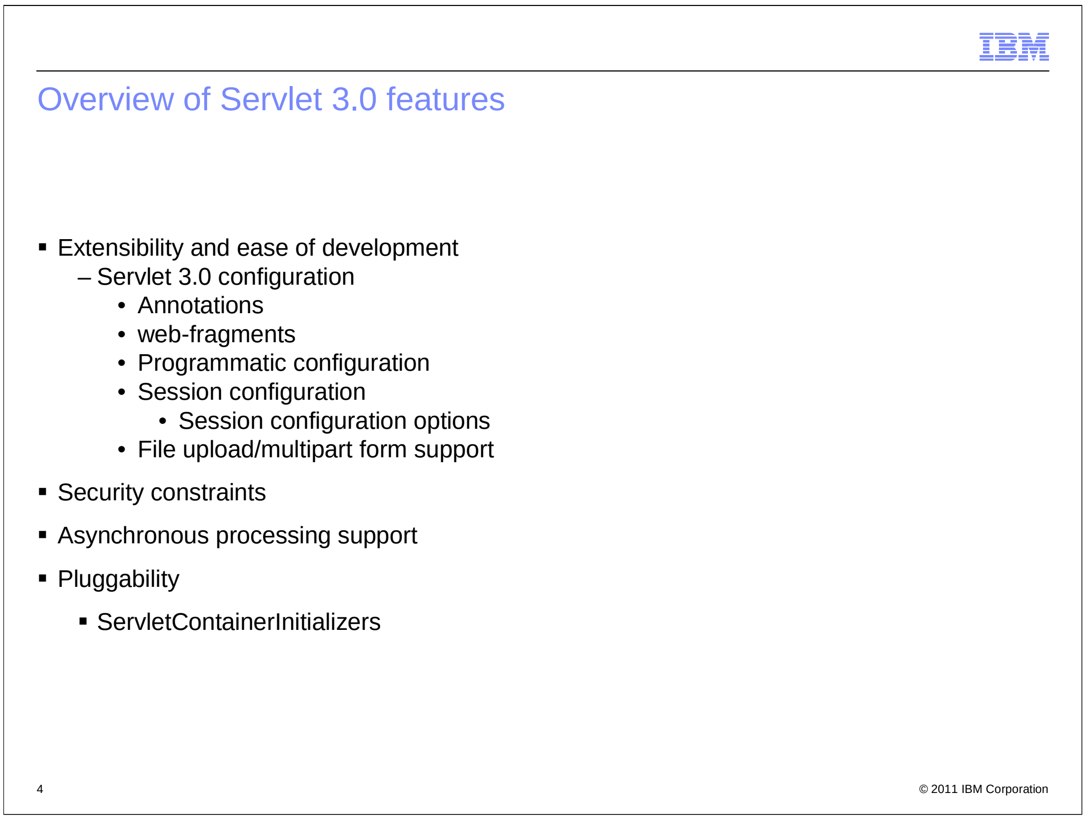
This presentation will go over these three technologies, show a few code samples, discuss problem determination, and show some references.

Section

Overview

3

What will Servlet, JSP, and the EL do for you?



Overview of Servlet 3.0 features

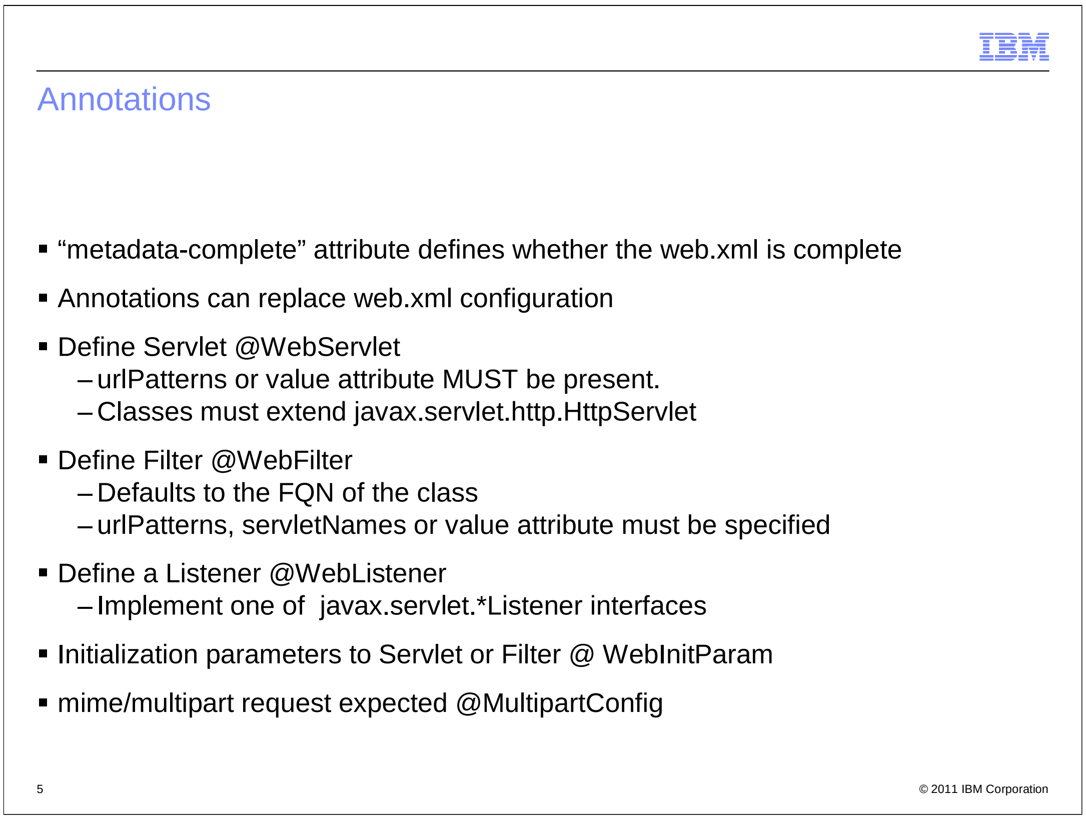
- Extensibility and ease of development
  - Servlet 3.0 configuration
    - Annotations
    - web-fragments
    - Programmatic configuration
    - Session configuration
      - Session configuration options
    - File upload/multipart form support
- Security constraints
- Asynchronous processing support
- Pluggability
  - ServletContainerInitializers

4

Servlet 3.0 addresses a few central themes. Servlet 3.0 is meant to provide extensibility and ease the development process. It accomplishes this through the use of annotations, web-fragments, and new configuration options. It also provides the ability to perform file upload and handle multipart form submission.

There are new security constraints that can be used to more easily protect certain resources and methods making the overall system more secure.

Servlet 3.0 also provides new APIs for asynchronous processing and provides pluggability through the use of ServletContainerInitializers.



Annotations

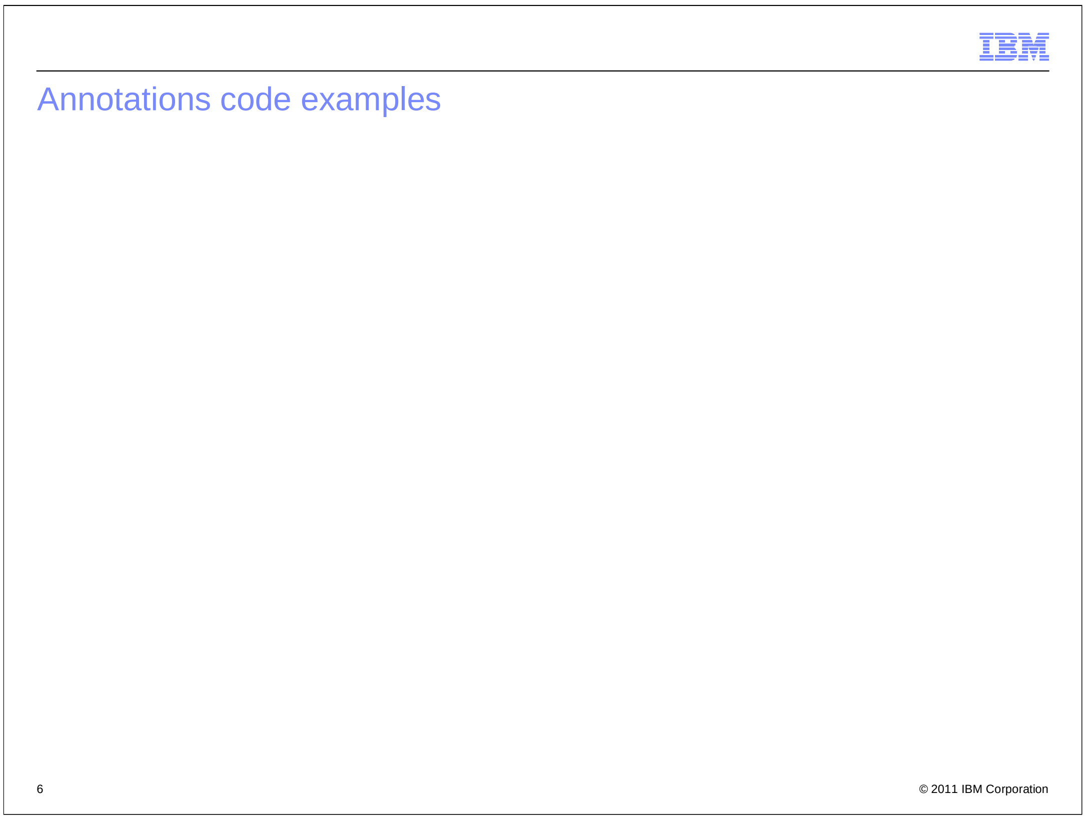
- "metadata-complete" attribute defines whether the web.xml is complete
- Annotations can replace web.xml configuration
- Define Servlet @WebServlet
  - urlPatterns or value attribute MUST be present.
  - Classes must extend javax.servlet.http.HttpServlet
- Define Filter @WebFilter
  - Defaults to the FQN of the class
  - urlPatterns, servletNames or value attribute must be specified
- Define a Listener @WebListener
  - Implement one of javax.servlet.*Listener interfaces
- Initialization parameters to Servlet or Filter @ WebInitParam
- mime/multipart request expected @MultipartConfig

5 © 2011 IBM Corporation

Annotation processing can be very useful, but it can also affect the performance due to the scanning that needs to be done. If annotation processing is not required, a developer can set the "metadata-complete" attribute to define that the web.xml is complete. If this attribute is not set, annotation processing occurs.

Annotations can be used to define servlets, filters, and listeners. Additionally, initialization parameter annotations can be used on the servlet and filter annotations.

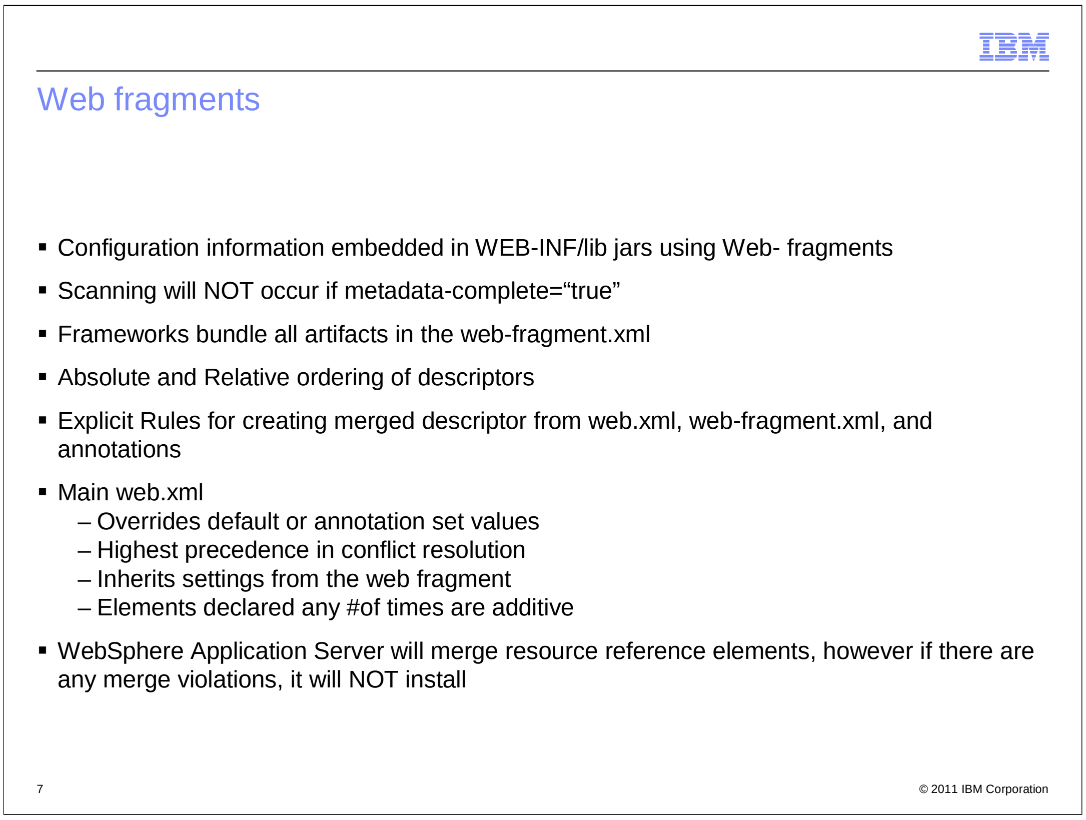
There is also a MultipartConfig annotation that can be used on a servlet expecting to receive multipart requests.



Annotations code examples

6

Here are some examples of several different annotations. Notice that some of these annotations take multiple parameters, some don't take any parameters, and some have just one parameter. You should see the javadoc for more information on each annotation.


Web fragments

- Configuration information embedded in WEB-INF/lib jars using Web- fragments
- Scanning will NOT occur if metadata-complete="true"
- Frameworks bundle all artifacts in the web-fragment.xml
- Absolute and Relative ordering of descriptors
- Explicit Rules for creating merged descriptor from web.xml, web-fragment.xml, and annotations
- Main web.xml
  - Overrides default or annotation set values
  - Highest precedence in conflict resolution
  - Inherits settings from the web fragment
  - Elements declared any #of times are additive
- WebSphere Application Server will merge resource reference elements, however if there are any merge violations, it will NOT install

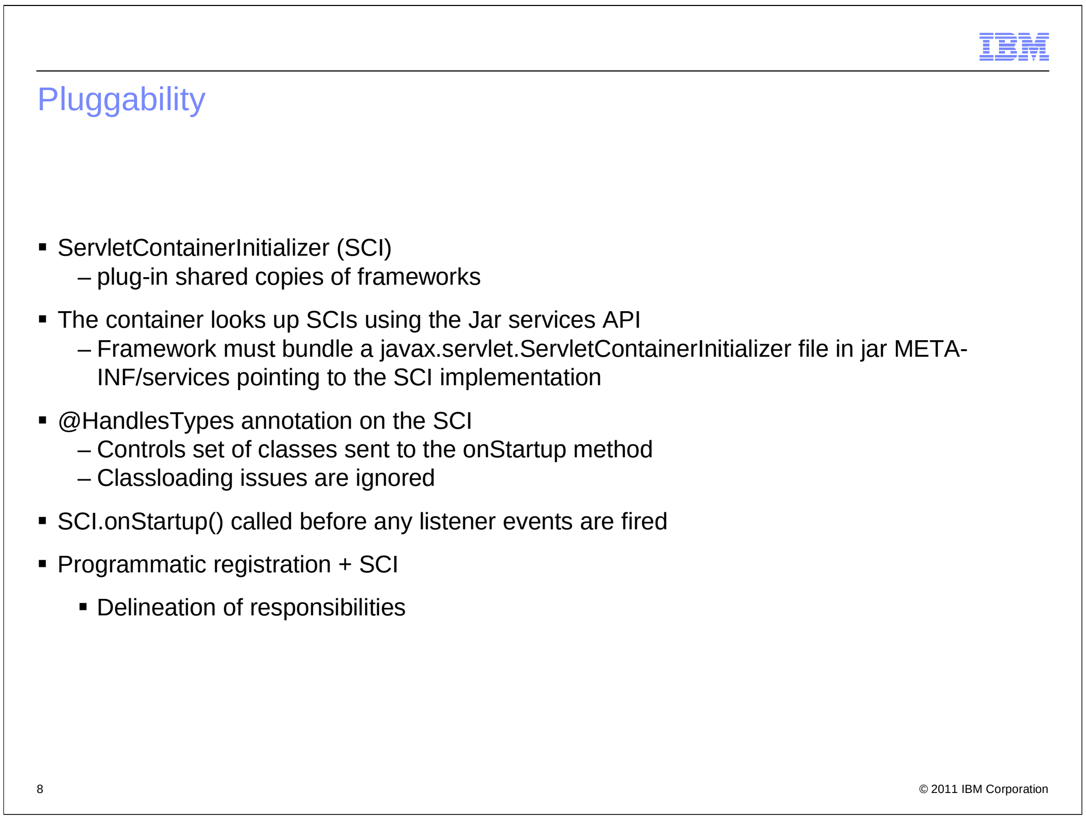
7 © 2011 IBM Corporation

Configuration information can be embedded into other jars using Web Fragments. This provides for better pluggability as some of the same configuration settings can be applied to multiple applications this jar is included with.

Scanning for annotations will not occur if the metadata-complete element is set in the web.xml.

It is suggested that frameworks bundle all artifacts in the web-fragment.xml.

The specification lists ways to order these descriptor files and has certain rules for merging and overriding configuration values. The main web.xml overrides default or annotation set values. It is used as the highest precedence when handling conflict resolution. It inherits other settings from the web fragments. Also, elements which can be declared multiple times are additive. If there are merge violations when deploying an application, the application will fail to install.

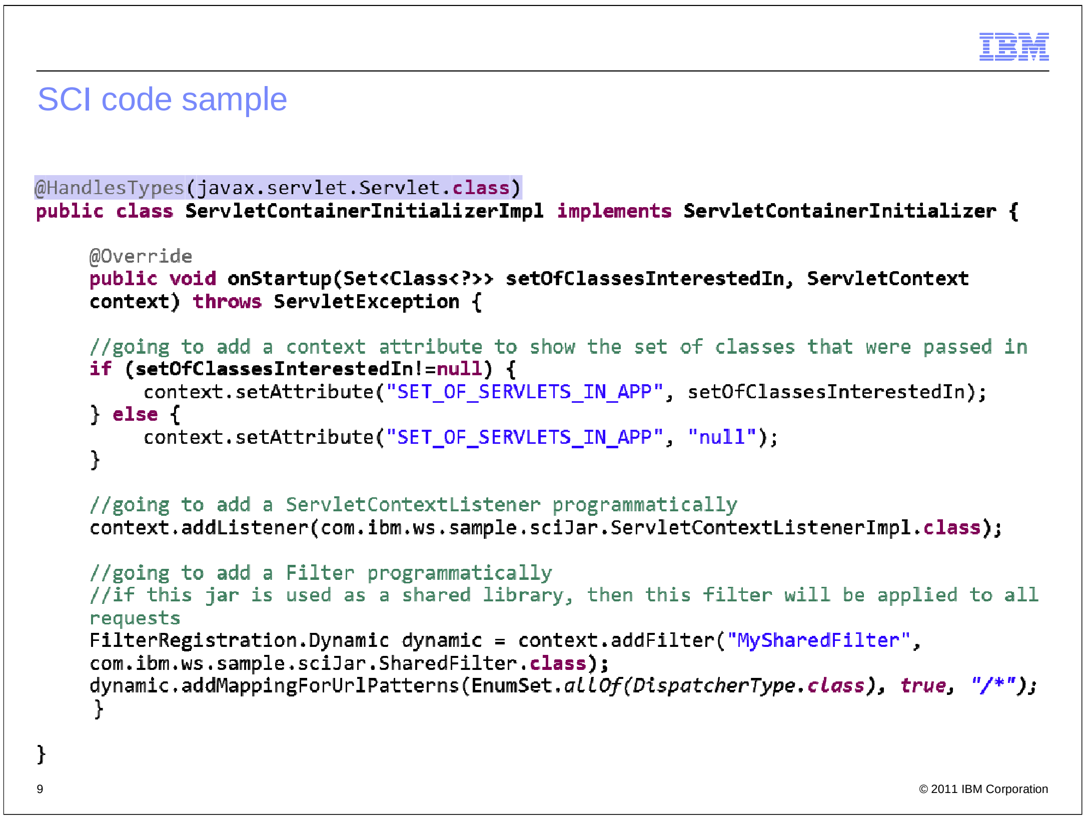


Pluggability

- ServletContainerInitializer (SCI)
  - – plug-in shared copies of frameworks
- The container looks up SCIs using the Jar services API
  - – Framework must bundle a javax.servlet.ServletContainerInitializer file in jar META-INF/services pointing to the SCI implementation
- @HandlesTypes annotation on the SCI
  - – Controls set of classes sent to the onStartup method
  - – Classloading issues are ignored
- SCI.onStartup() called before any listener events are fired
- Programmatic registration + SCI
  - Delineation of responsibilities

8 © 2011 IBM Corporation

Along with WebFragments, Servlet 3.0 also provides ServletContainerInitializer support. A ServletContainerInitializer is executed before application startup and allows for frameworks to be shared and configured. ServletContainerInitializers are found using the Jar services A.P.I. A framework must bundle a javax.servlet.ServletContainerInitializer file in the META-INF/services directory of its JAR referencing the ServletContainerInitializer implementation within this file. The onStartup method of this ServletContainerInitializer is called before any listener events are fired and a set of classes matching the HandlesTypes annotation can be passed in. A ServletContainerInitializer can programmatically register servlets, filters, and listeners thus giving a framework jar more responsibility in configuring the environment.

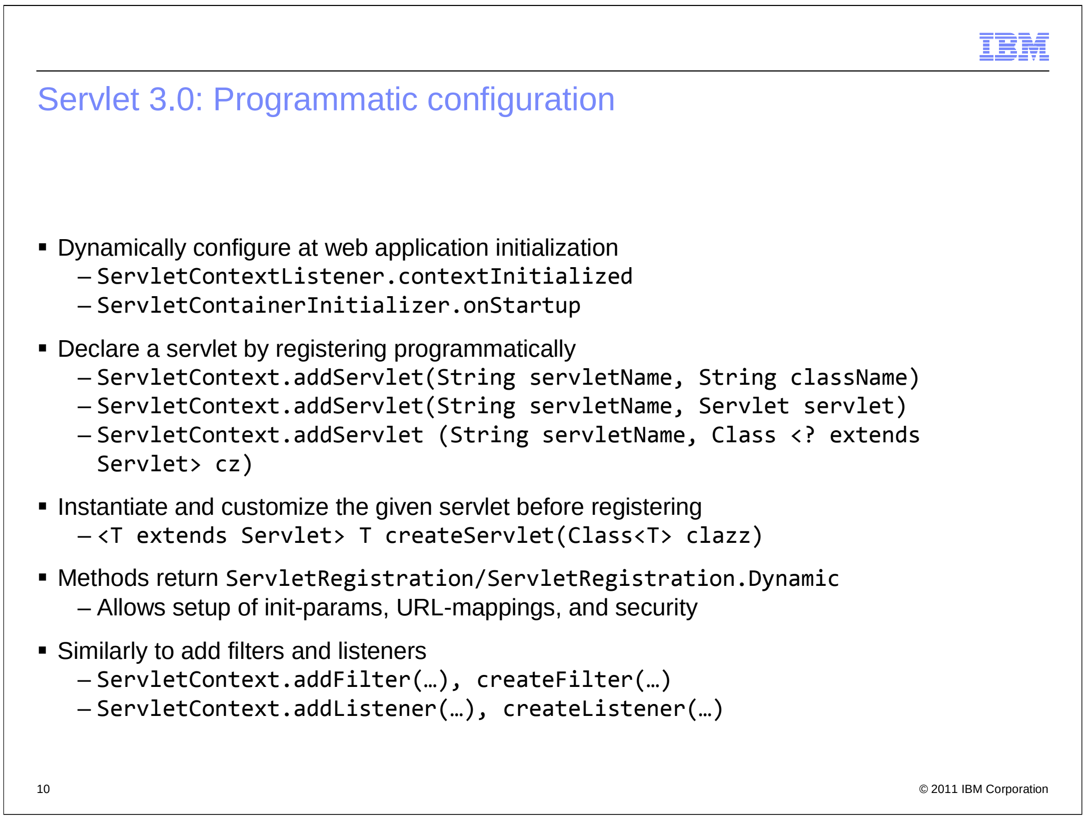


SCI code sample

```
@HandlesTypes(javax.servlet.Servlet.class)
public class ServletContainerInitializerImpl implements ServletContainerInitializer {

    @Override
    public void onStartup(Set<Class<?>> setOfClassesInterestedIn, ServletContext
    context) throws ServletException {

    //going to add a context attribute to show the set of classes that were passed in
    if (setOfClassesInterestedIn!=null) {
        context.setAttribute("SET_OF_SERVLETS_IN_APP", setOfClassesInterestedIn);
    } else {
        context.setAttribute("SET_OF_SERVLETS_IN_APP", "null");
    }

    //going to add a ServletContextListener programmatically
    context.addListener(com.ibm.ws.sample.sciJar.ServletContextListenerImpl.class);

    //going to add a Filter programmatically
    //if this jar is used as a shared library, then this filter will be applied to all
    requests
    FilterRegistration.Dynamic dynamic = context.addFilter("MySharedFilter",
    com.ibm.ws.sample.sciJar.SharedFilter.class);
    dynamic.addMappingForUrlPatterns(EnumSet.allOf(DispatcherType.class), true, "/*");
     }

}
```

9 © 2011 IBM Corporation

Here is a sample ServletContainerInitializer class file. Notice the HandlesTypes annotation which will cause the onStartup method to be called passing in all the javax.servlet.Servlet classes within the application along with the ServletContext. Also, notice that this ServletContainerInitializer programmatically adds a listener and filter.



Servlet 3.0: Programmatic configuration

- Dynamically configure at web application initialization
  - `ServletContextListener.contextInitialized`
  - `ServletContainerInitializer.onStartup`
- Declare a servlet by registering programmatically
  - `ServletContext.addServlet(String servletName, String className)`
  - `ServletContext.addServlet(String servletName, Servlet servlet)`
  - `ServletContext.addServlet (String servletName, Class <? extends Servlet> cz)`
- Instantiate and customize the given servlet before registering
  - `<T extends Servlet> T createServlet(Class<T> clazz)`
- Methods return `ServletRegistration/ServletRegistration.Dynamic`
  - Allows setup of init-params, URL-mappings, and security
- Similarly to add filters and listeners
  - `ServletContext.addFilter(…), createFilter(…)`
  - `ServletContext.addListener(…), createListener(…)`

10 © 2011 IBM Corporation

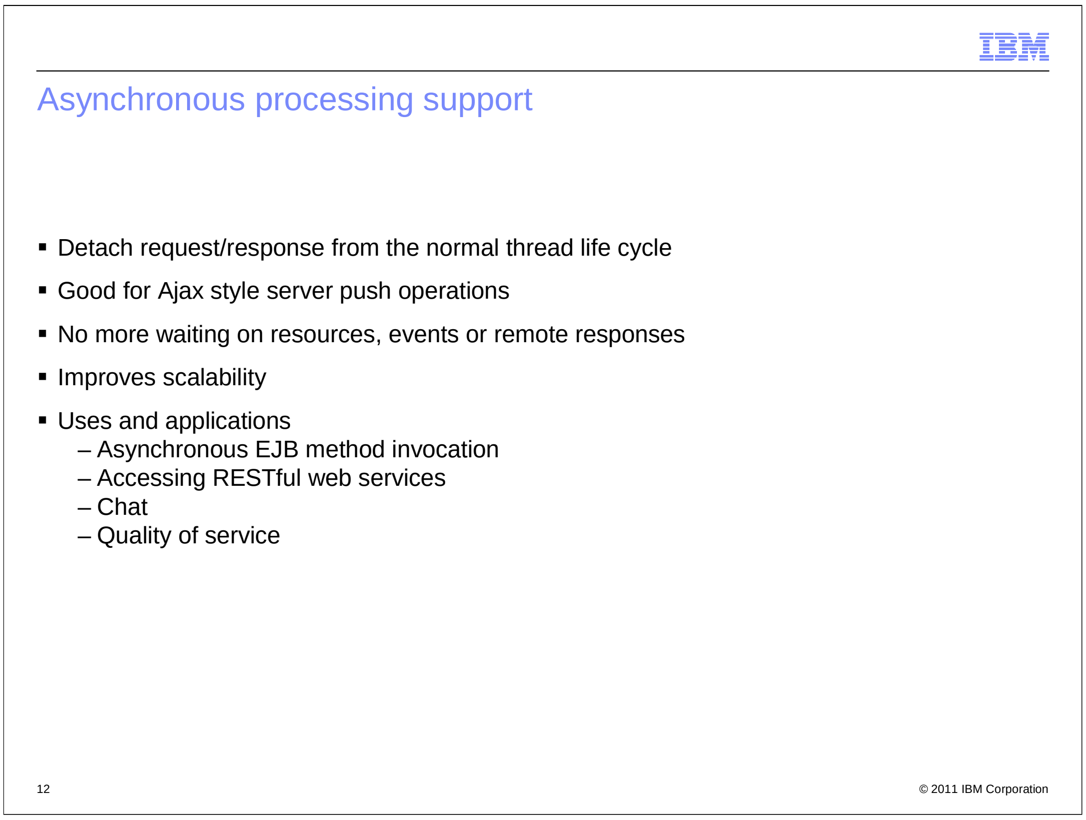
Programmatic configuration needs to occur before an application completes initialization. Therefore, programmatic configuration should be done within ServletContextListeners during their contextInitialized method or within ServletContainerInitializers during their onStartup method. You can see there are multiple methods which can be used to add Servlets. You can also create a Servlet instance using the createServlet method. If you use the addServlet method which takes the instance, the instance should have been created using the createServlet method. Otherwise, resource injection will not have occurred.

These methods return ServletRegistration.Dynamic objects which can be used to further set up init-params, URL-mappings, and security.

Filters and Listeners can be programmatically added in a similar way.

Adding servlets using APIs

Here is an example of a ServletContextListener adding three servlets using the different method parameters.



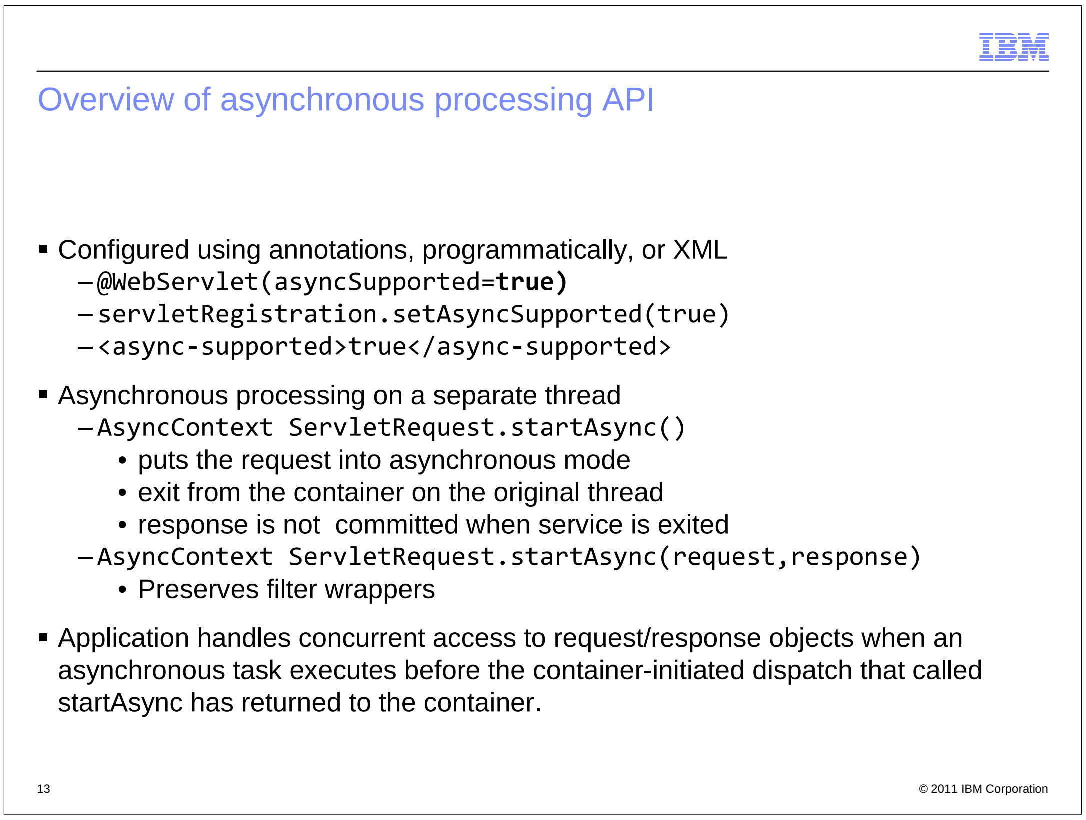
Asynchronous processing support

- Detach request/response from the normal thread life cycle
- Good for Ajax style server push operations
- No more waiting on resources, events or remote responses
- Improves scalability
- Uses and applications
  - Asynchronous EJB method invocation
  - Accessing RESTful web services
  - Chat
  - Quality of service

12 © 2011 IBM Corporation

Servlet 3.0 also supports Asynchronous Processing. An application can detach the request and response from the normal thread life cycle. This is beneficial for Ajax style server push operations. Threads can be used for active requests and there is no need to wait for resources, events, or remote responses.

Asynchronous processing improves scalability and is useful when doing Asynchronous EJB method invocation, RESTful web services, implementing chat or quality of service type applications.



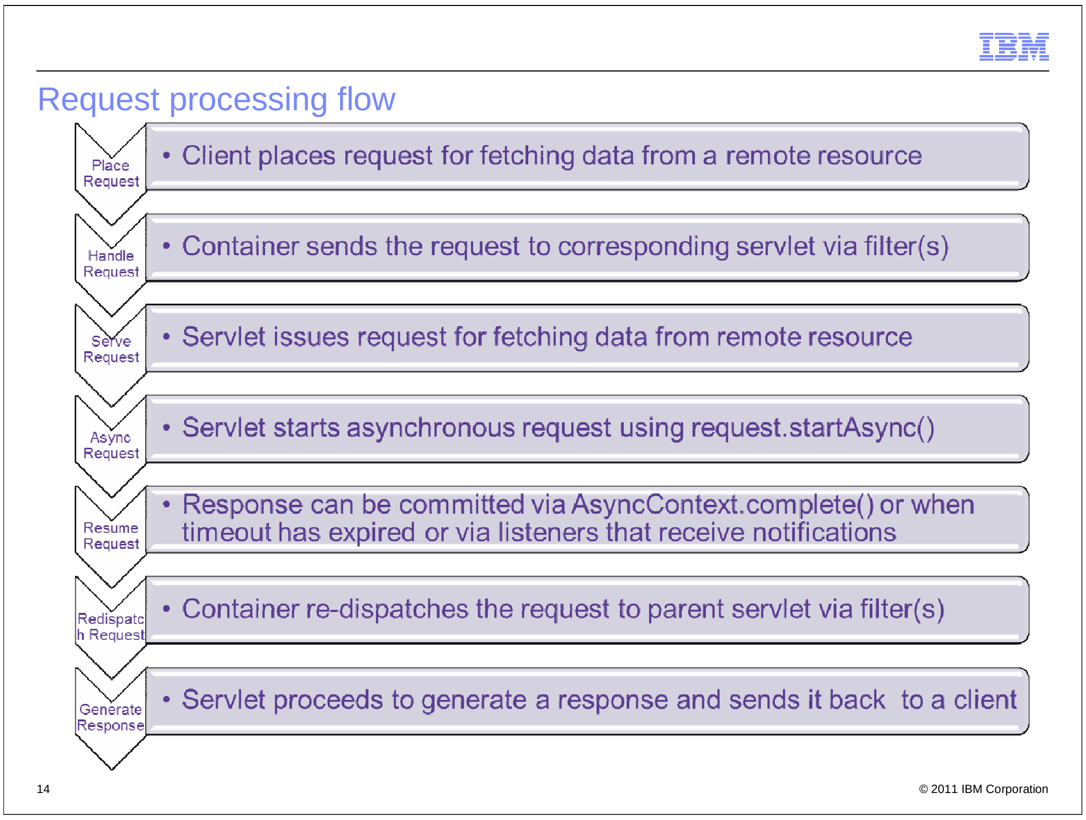
Overview of asynchronous processing API

- Configured using annotations, programmatically, or XML
  - `@WebServlet(asyncSupported=`**`true`**`)`
  - `servletRegistration.setAsyncSupported(true)`
  - `<async-supported>true</async-supported>`
- Asynchronous processing on a separate thread
  - `AsyncContext ServletRequest.startAsync()`
    - puts the request into asynchronous mode
    - exit from the container on the original thread
    - response is not committed when service is exited
  - `AsyncContext ServletRequest.startAsync(request,response)`
    - Preserves filter wrappers
- Application handles concurrent access to request/response objects when an asynchronous task executes before the container-initiated dispatch that called startAsync has returned to the container.

13

Asynchronous processing can be configured using annotations, programmatically, or XML.

In order to put a request into asynchronous mode, you must call startAsync on the ServletRequest. You can also call startAsync passing in the original request and response objects to preserve the filter wrappers. When you put a request into asynchronous mode, you can start another runnable, dispatch to another servlet, or complete the response. When the original thread exits out of the service method, the thread is returned to do more processing and the response waits until a dispatch or complete is called. A word of caution, it is up to the application developer to handle concurrent access to the request and response objects if a runnable thread is started.



Request processing flow

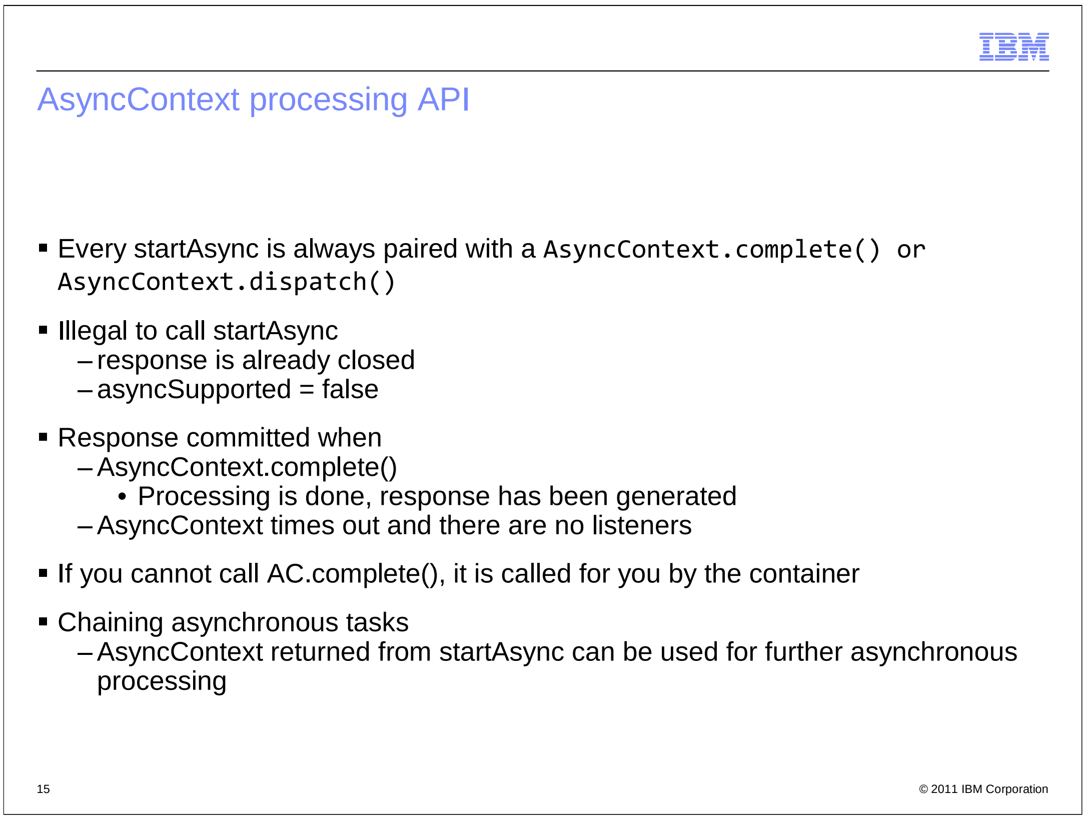
Place Request
Handle Request
Serve Request
Async Request
Resume Request
Redispatch Request
Generate Response

- Client places request for fetching data from a remote resource
- Container sends the request to corresponding servlet via filter(s)
- Servlet issues request for fetching data from remote resource
- Servlet starts asynchronous request using request.startAsync()
- Response can be committed via AsyncContext.complete() or when timeout has expired or via listeners that receive notifications
- Container re-dispatches the request to parent servlet via filter(s)
- Servlet proceeds to generate a response and sends it back to a client

14

© 2011 IBM Corporation

Here is an example of the request processing flow for a sample request that goes asynchronous. In this example, a client makes a request. The container handles the request. It starts an asynchronous thread which waits for a response from the remote resource while returning the original thread. When the response was obtained (or a timeout occurred or an AsyncListener fired) the container re-dispatches to a servlet which can generate and print the response.

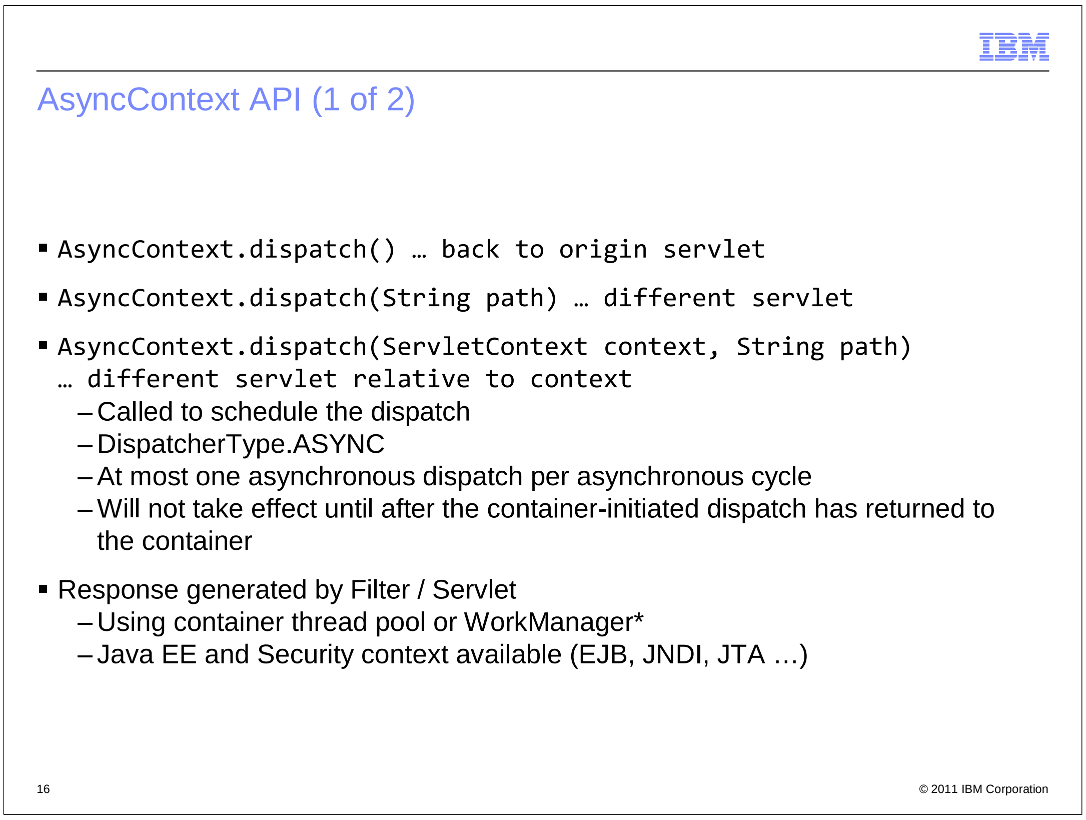


AsyncContext processing API

- Every startAsync is always paired with a `AsyncContext.complete() or AsyncContext.dispatch()`
- Illegal to call startAsync
  - response is already closed
  - asyncSupported = false
- Response committed when
  - AsyncContext.complete()
    - Processing is done, response has been generated
  - AsyncContext times out and there are no listeners
- If you cannot call AC.complete(), it is called for you by the container
- Chaining asynchronous tasks
  - AsyncContext returned from startAsync can be used for further asynchronous processing

15

Here is some more detail on the AsyncContext processing APIs. Every startAsync should always be paired with an AsyncContext.complete or AsyncContext.dispatch. It is illegal to call startAsync if the response has already been closed or if the servlet does not support async behavior. The response is committed when either an AsyncContext.complete is called or when it times out and there are no listeners to handle the behavior. If you cannot call AsyncContext.complete, the container will call it for you. You can also chain asynchronous tasks together for further processing as long as you only do one AsyncContext dispatch at a time and each one is paired with a previous AsyncContext.startAsync.



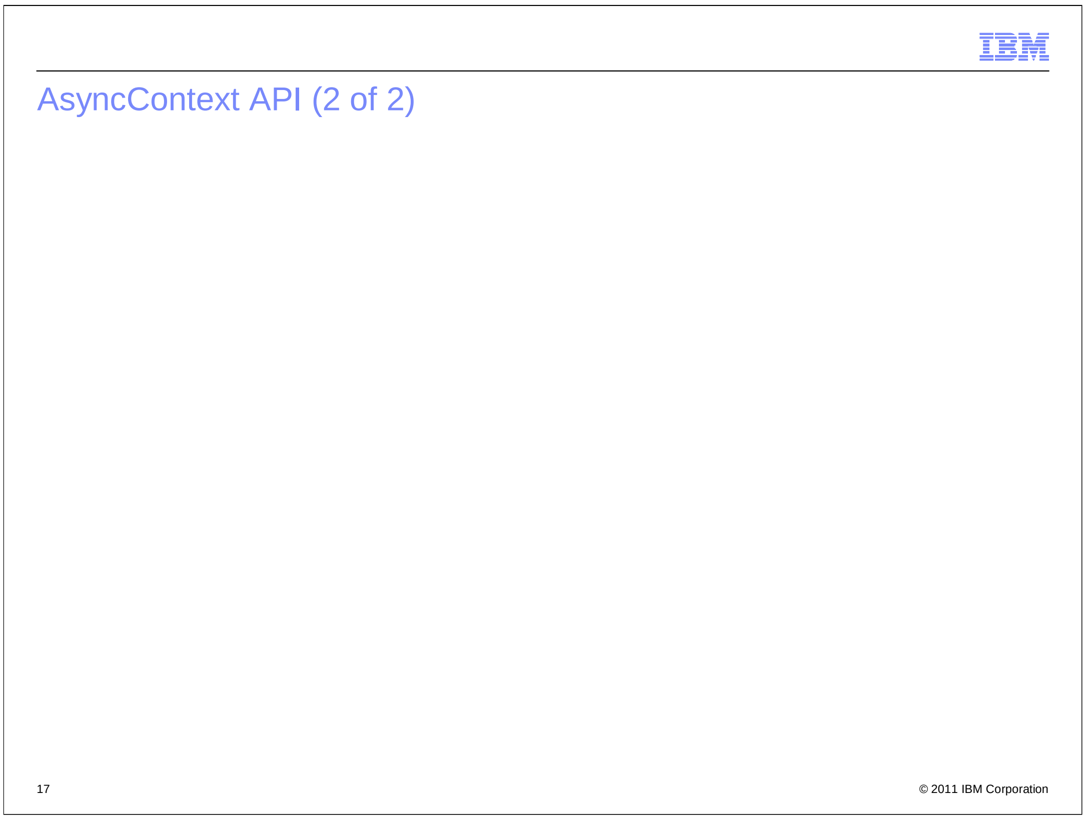
AsyncContext API (1 of 2)

- `AsyncContext.dispatch() … back to origin servlet`
- `AsyncContext.dispatch(String path) … different servlet`
- `AsyncContext.dispatch(ServletContext context, String path) … different servlet relative to context`
  - Called to schedule the dispatch
  - DispatcherType.ASYNC
  - At most one asynchronous dispatch per asynchronous cycle
  - Will not take effect until after the container-initiated dispatch has returned to the container
- Response generated by Filter / Servlet
  - Using container thread pool or WorkManager*
  - Java EE and Security context available (EJB, JNDI, JTA …)

16 © 2011 IBM Corporation

There are multiple dispatch APIs providing slightly different behavior. A dispatch can be called to continue processing the request and write out the response. You can call a dispatch from one of the asynchronous runnable threads (which makes the most sense), or you can call it from within another servlet as long as a startAsync was previously called. If it is called from a previous servlet, the dispatch will not take effect until the previous servlet has completed its method.

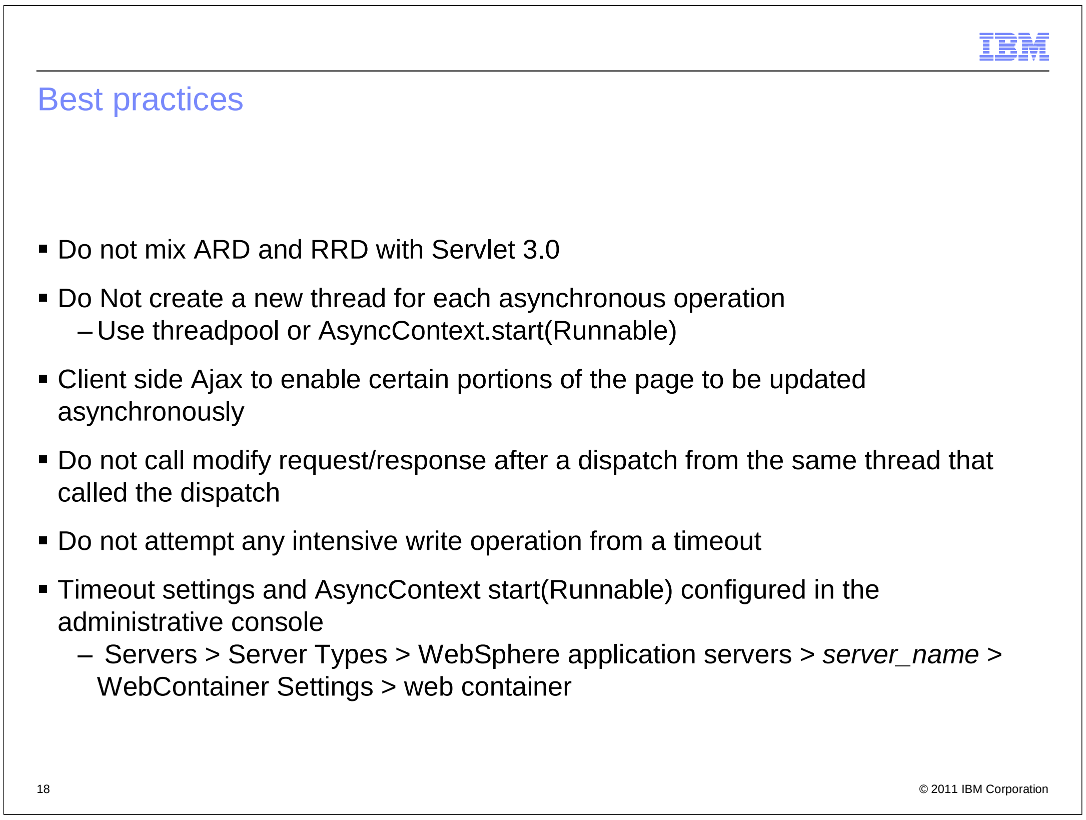
Typically, the response should be generated by the filters/servlets. The advantage to using asynchronous processing is to release a thread back to the container to continue processing. By default, the asynchronous processing occurs in the WorkManager thread pool which has Java EE and Security context available.



AsyncContext API (2 of 2)

17

Here are some more asynchronous processing APIs that allow you configure or get state from the AsyncContext object. Note that you can also create and add an AsyncListener to this context.



Best practices

- Do not mix ARD and RRD with Servlet 3.0
- Do Not create a new thread for each asynchronous operation
  - Use threadpool or AsyncContext.start(Runnable)
- Client side Ajax to enable certain portions of the page to be updated asynchronously
- Do not call modify request/response after a dispatch from the same thread that called the dispatch
- Do not attempt any intensive write operation from a timeout
- Timeout settings and AsyncContext start(Runnable) configured in the administrative console
  - Servers > Server Types > WebSphere application servers > *server_name* > WebContainer Settings > web container

18 © 2011 IBM Corporation

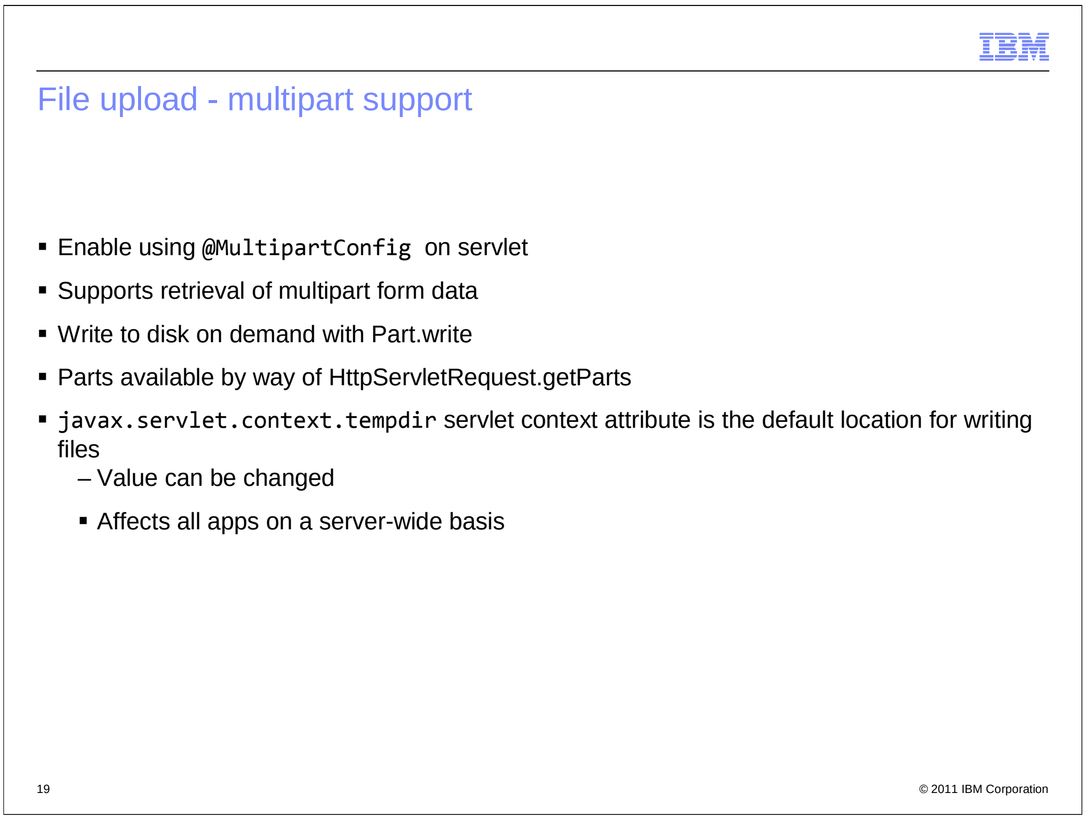
Here are some of the best practices when using Servlet 3.0. Do NOT mix ARD or RRD with Servlet 3.0 asynchronous processing. They are two separate technologies that should not be used together. You should avoid creating a new thread for each asynchronous operation. Instead use a thread pool or the AsyncContext.start(Runnable) method.

You should use client side Ajax to enable certain portions of the page to be updated asynchronously.

You should not modify the request or response after a dispatch from the same thread that called the dispatch.

You should not attempt an intensive write from a timeout.

You should use the administrative console to configure asynchronous processing settings.



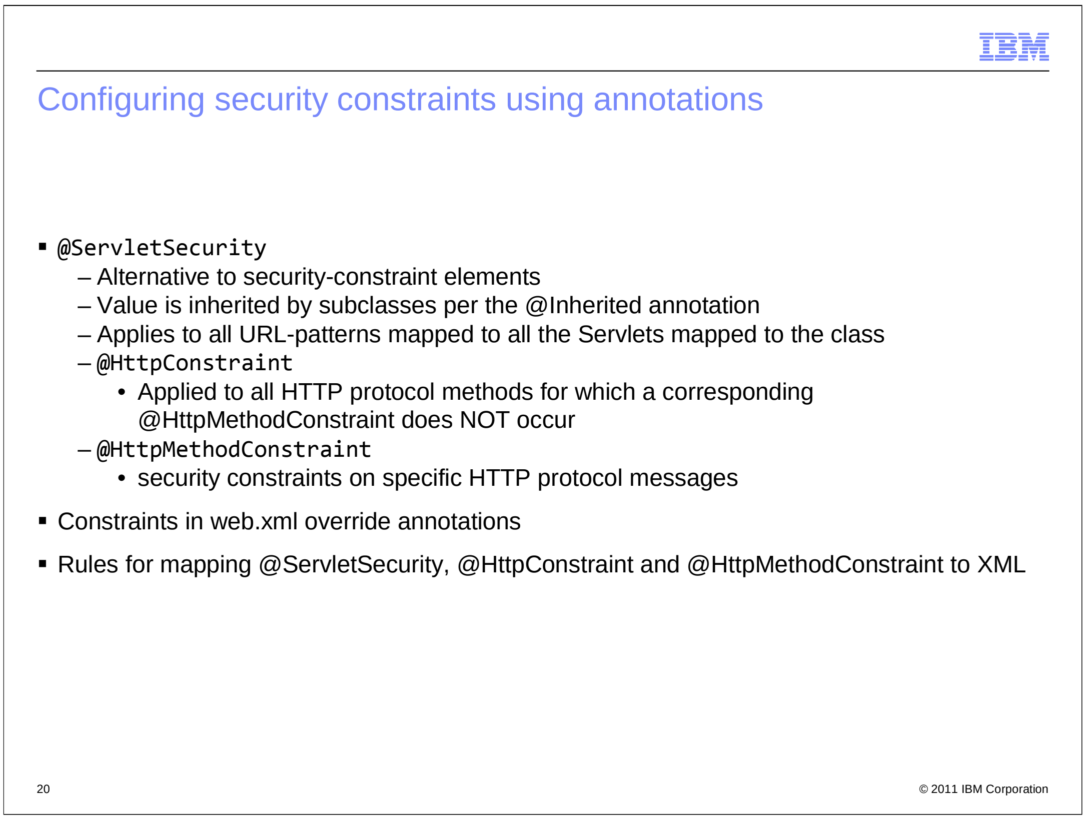
File upload - multipart support

- Enable using `@MultipartConfig` on servlet
- Supports retrieval of multipart form data
- Write to disk on demand with Part.write
- Parts available by way of HttpServletRequest.getParts
- `javax.servlet.context.tempdir` servlet context attribute is the default location for writing files
  - Value can be changed
    - Affects all apps on a server-wide basis

19 © 2011 IBM Corporation

Servlet 3.0 also supports file upload and multipart servlets. You can enable multipart using the MultipartConfig annotation. This allows the servlet to retrieve the multipart form data. The parts of a request are available using the HttpServletRequest.getParts method and you can write to disk using the Part.write method.

The default location for any file written to disk is the temp directory specified using the javax.servlet.context.tempdir attribute.



Configuring security constraints using annotations

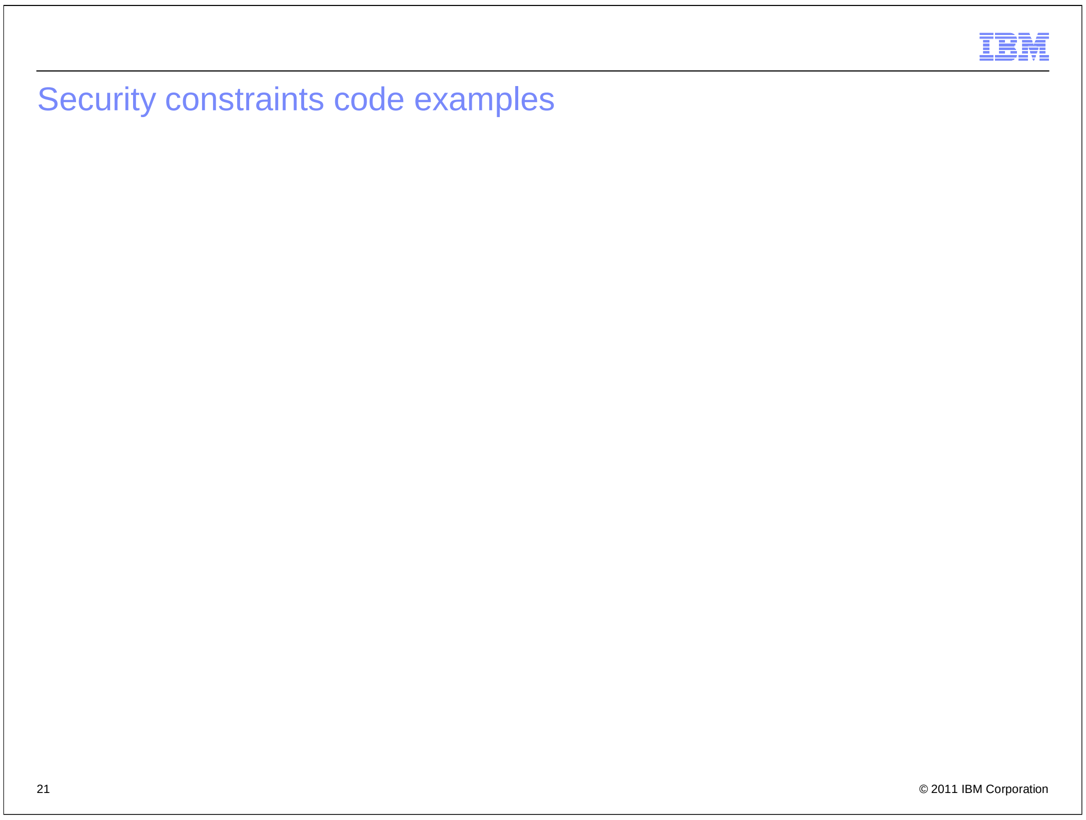
- `@ServletSecurity`
  - Alternative to security-constraint elements
  - Value is inherited by subclasses per the @Inherited annotation
  - Applies to all URL-patterns mapped to all the Servlets mapped to the class
  - `@HttpConstraint`
    - Applied to all HTTP protocol methods for which a corresponding @HttpMethodConstraint does NOT occur
  - `@HttpMethodConstraint`
    - security constraints on specific HTTP protocol messages
- Constraints in web.xml override annotations
- Rules for mapping @ServletSecurity, @HttpConstraint and @HttpMethodConstraint to XML

20

Servlet 3.0 also provides security configuration through the use of annotations. @ServletSecurity can be used as an alternative to security-constraint elements. It's value is inherited by subclasses as per the @Inherited annotation. Also, this annotation applies to all URL patterns mapped to the servlets with this annotation.

The HttpConstraint and HttpMethodConstraint annotations provides the security constraint configuration on the HTTP protocol methods.

Constraints within the web.xml do override the annotations and there are rules for mapping these annotations to the XML.

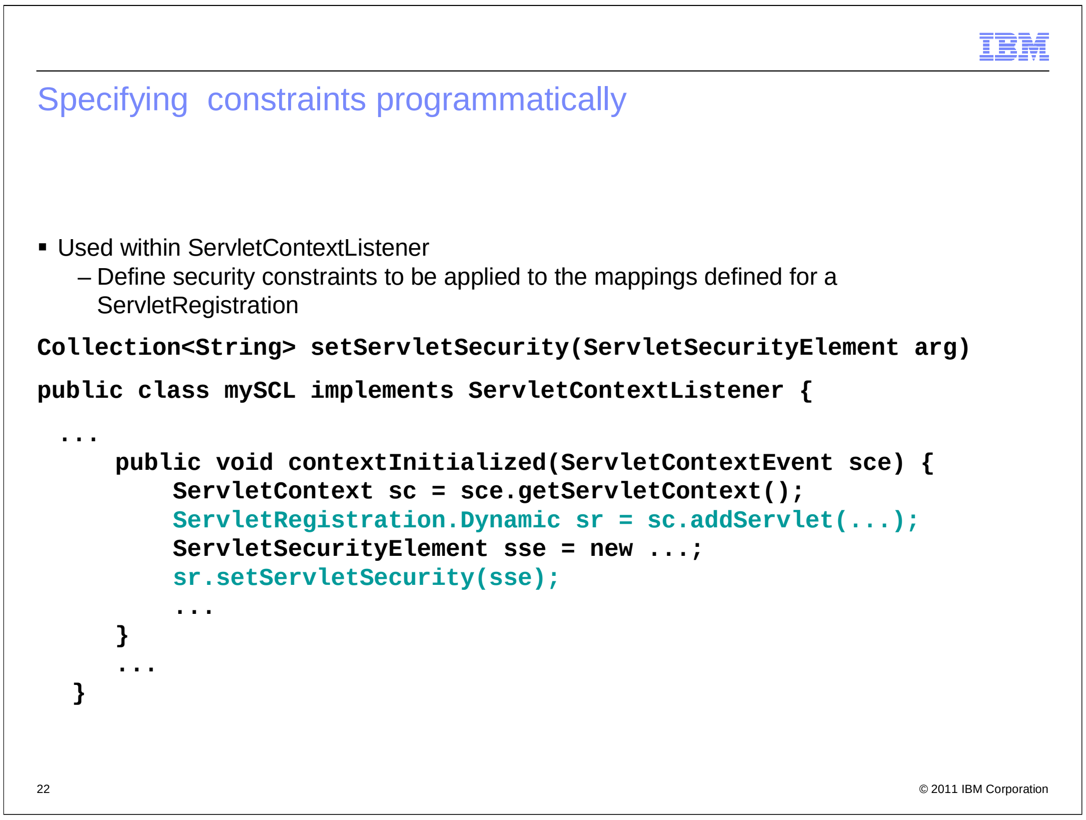


Security constraints code examples

21

© 2011 IBM Corporation

Here are two examples of using the security annotations on a class.



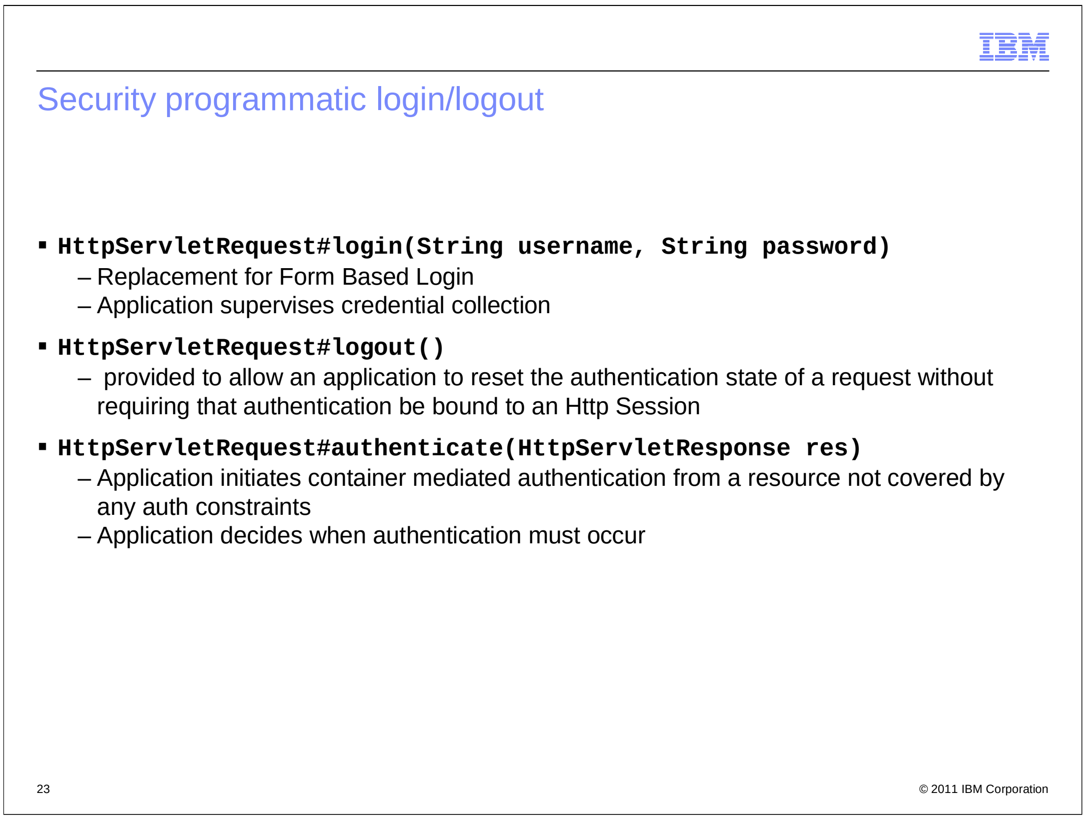
Specifying constraints programmatically

- Used within ServletContextListener
  - Define security constraints to be applied to the mappings defined for a ServletRegistration

```
Collection<String> setServletSecurity(ServletSecurityElement arg)
public class mySCL implements ServletContextListener {
  ...
      public void contextInitialized(ServletContextEvent sce) {
          ServletContext sc = sce.getServletContext();
          ServletRegistration.Dynamic sr = sc.addServlet(...);
          ServletSecurityElement sse = new ...;
          sr.setServletSecurity(sse);
          ...
      }
      ...
   }
```

22

You can also specify security constraints programmatically. When programmatically adding a servlet, you can use the ServletSecurityElement to define the constraints for the given servlet mappings.

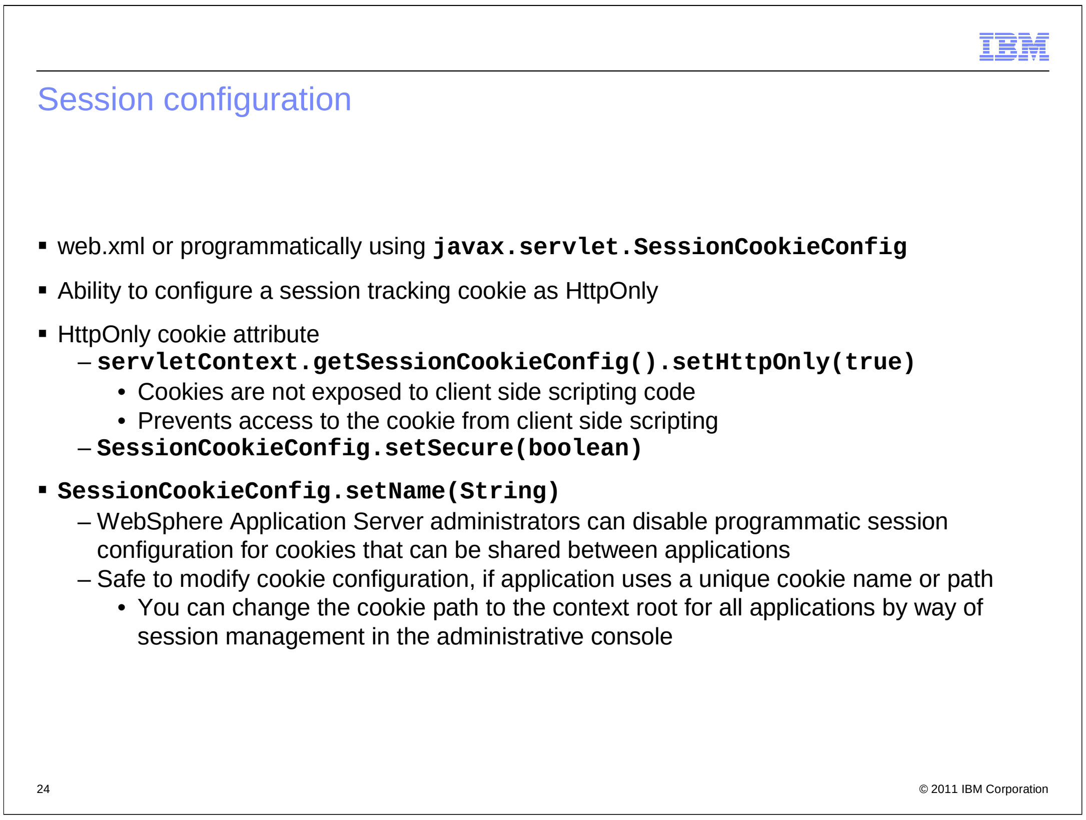


Security programmatic login/logout

- **`HttpServletRequest#login(String username, String password)`**
  - Replacement for Form Based Login
  - Application supervises credential collection
- **`HttpServletRequest#logout()`**
  - provided to allow an application to reset the authentication state of a request without requiring that authentication be bound to an Http Session
- **`HttpServletRequest#authenticate(HttpServletResponse res)`**
  - Application initiates container mediated authentication from a resource not covered by any auth constraints
  - Application decides when authentication must occur

23

The HttpServletRequest has three new methods which allow for security integration. The login method is a replacement for form based login and the application can supervise the credential collection. The logout method allows the application to reset the authentication state of a request without requiring that authentication be bound to a Http Session. The authenticate method allows an application to initiate authentication from a resource that is not covered by any authorization constraints which gives the application the control over when authentication occurs.

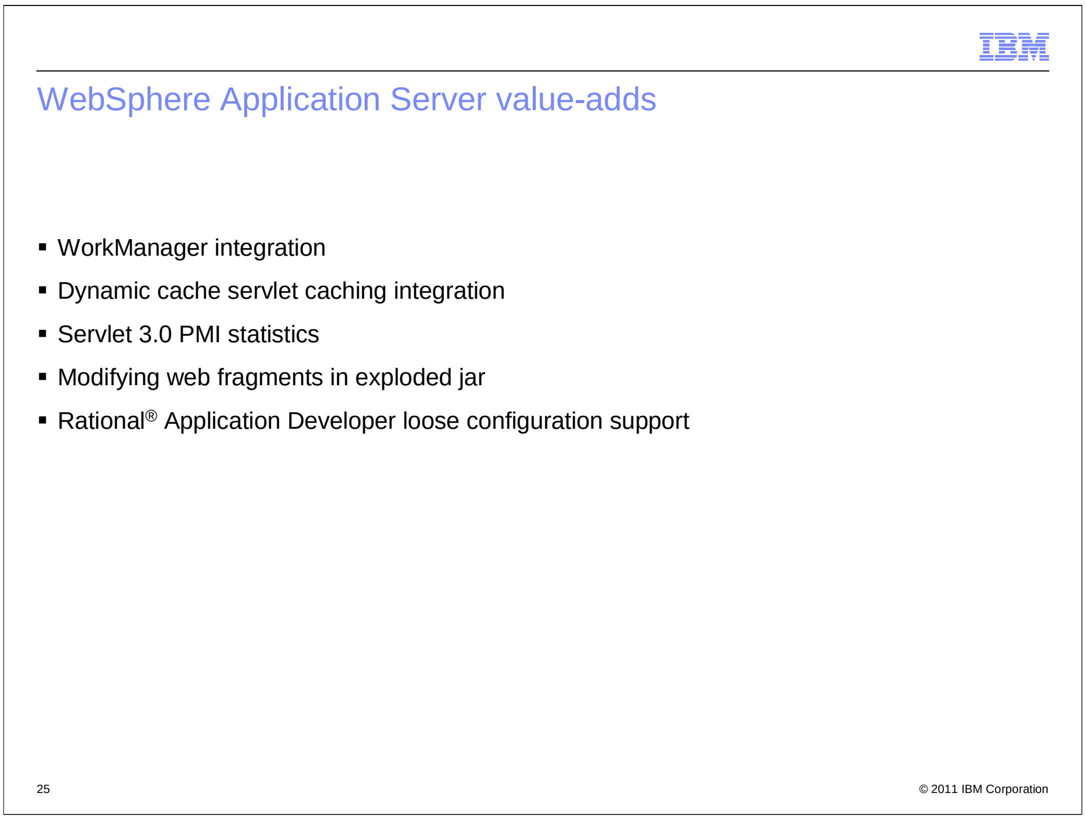


Session configuration

- web.xml or programmatically using **`javax.servlet.SessionCookieConfig`**
- Ability to configure a session tracking cookie as HttpOnly
- HttpOnly cookie attribute
  - **`servletContext.getSessionCookieConfig().setHttpOnly(true)`**
    - Cookies are not exposed to client side scripting code
    - Prevents access to the cookie from client side scripting
  - **`SessionCookieConfig.setSecure(boolean)`**
- **`SessionCookieConfig.setName(String)`**
  - WebSphere Application Server administrators can disable programmatic session configuration for cookies that can be shared between applications
  - Safe to modify cookie configuration, if application uses a unique cookie name or path
    - You can change the cookie path to the context root for all applications by way of session management in the administrative console

24

Servlet 3.0 provides additional session configuration options through the web.xml or through configuration objects. The specification now fully supports the HttpOnly cookie attribute. The session configuration can be used to change the attributes of the cookie object including the name. While this is an option, you should proceed with caution as some extended WebSphere behavior might rely on using the same cookie for all applications. You also have the option to set the path for each application to that of its context root.



WebSphere Application Server value-adds

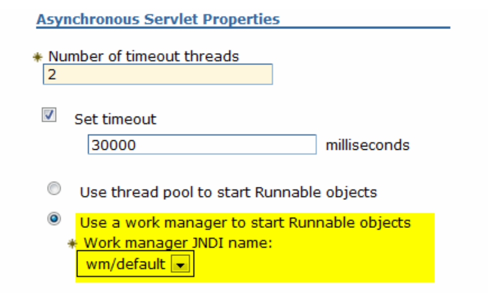
- WorkManager integration
- Dynamic cache servlet caching integration
- Servlet 3.0 PMI statistics
- Modifying web fragments in exploded jar
- Rational® Application Developer loose configuration support

25

Some of the WebSphere Application Server adds are the WorkManager integration for asynchronous processing, dynamic cache servlet caching integration, servlet 3.0 performance metric statistics, the ability to modify a web fragment in an exploded jar, and support for rational application developer’s loose configuration.

Servlet 3.0 Work manager integration

- Container Thread pool created for JEE applications
  - Uses async beans
- Bound to the JNDI namespace
- Context of the caller is inherited on the work thread
  - share asynchronous scope
- Application does NOT need to do anything
- Asynchronous task implements Runnable
- Gotcha
  - Use of asynchronous beans within a JPA extended persistence context is not supported.



Asynchronous Servlet Properties

* Number of timeout threads: 2

[✓] Set timeout: 30000 milliseconds

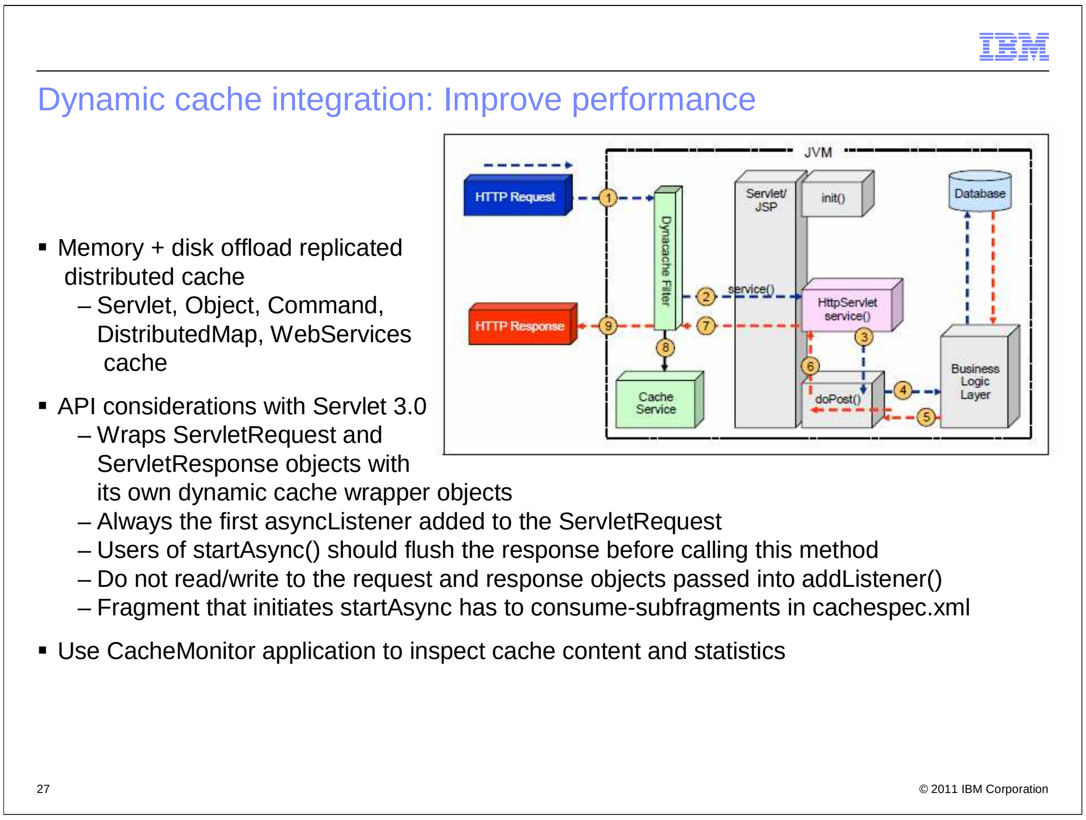
( ) Use thread pool to start Runnable objects

[✓] Use a work manager to start Runnable objects

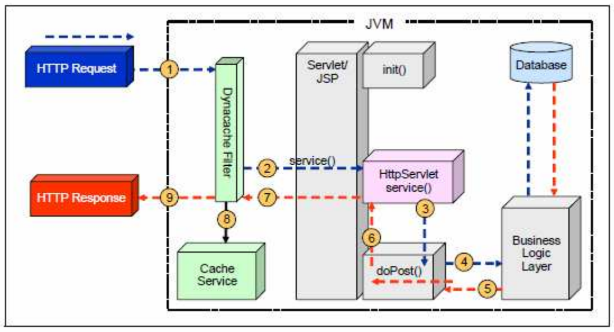
* Work manager JNDI name: wm/default

- Advantages over thread pool
  - Transactions
  - Access to Java EE component metadata
  - Connection Management
  - Security

The Work Manager is a container thread pool created for JEE applications. It can use async beans and is bound to the JNDI namespace. The context of the caller is inherited on the work manager thread thus being in the same asynchronous scope. The application does not need to do anything to take advantage of this integrated behavior.



Dynamic cache integration: Improve performance



- Memory + disk offload replicated distributed cache
  - Servlet, Object, Command, DistributedMap, WebServices cache
- API considerations with Servlet 3.0
  - Wraps ServletRequest and ServletResponse objects with its own dynamic cache wrapper objects
  - Always the first asyncListener added to the ServletRequest
  - Users of startAsync() should flush the response before calling this method
  - Do not read/write to the request and response objects passed into addListener()
  - Fragment that initiates startAsync has to consume-subfragments in cachespec.xml
- Use CacheMonitor application to inspect cache content and statistics

27 © 2011 IBM Corporation

Dynamic cache integration will improve the performance of your application and can be enabled through the web container administrative console. Objects, including servlet output, can be cached to provide quicker response times for subsequent identical requests. Dynamic cache does support caching of Servlet 3.0 asynchronous responses.

You should consider a few things when using dynamic caching with Servlet 3.0 asynchronous processing.

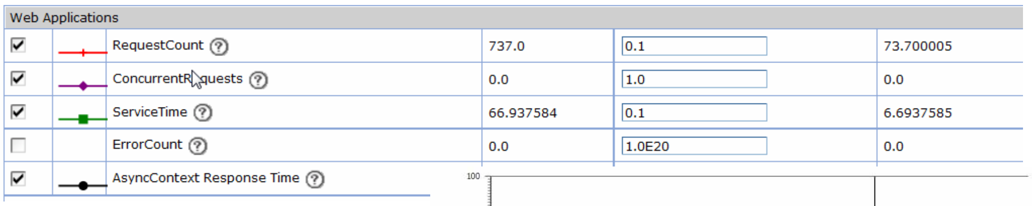
First, you should use the startAsync API that preserves filter wrappers and includes the request and response objects.

You should not read or write to the request and response objects that are passed into an AsyncListener.

You should always flush the response outputstream or writer before calling startAsync(...). dynamic caching will force the JSP or servlet that initiates the asynchronous processing, the one that calls startAsync, to consume the output of its sub fragments.

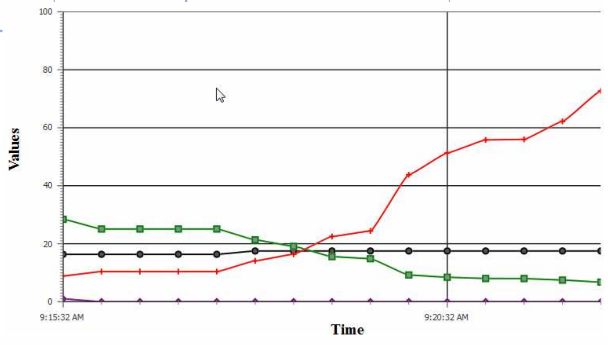
To view the cached response objects, you should install the CacheMonitor application that is provided in the installableApps directory of the server.

WebApplication Servlet 3.0 PMI statistics

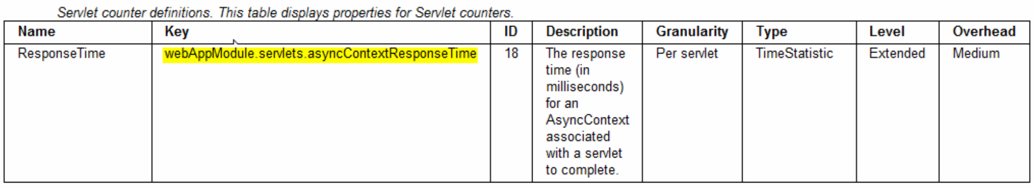


Web Applications

✓	—+—	RequestCount	737.0	0.1	73.700005
✓	—◆—	ConcurrentRequests	0.0	1.0	0.0
✓	—■—	ServiceTime	66.937584	0.1	6.6937585
☐		ErrorCount	0.0	1.0E20	0.0
✓	—●—	AsyncContext Response Time			



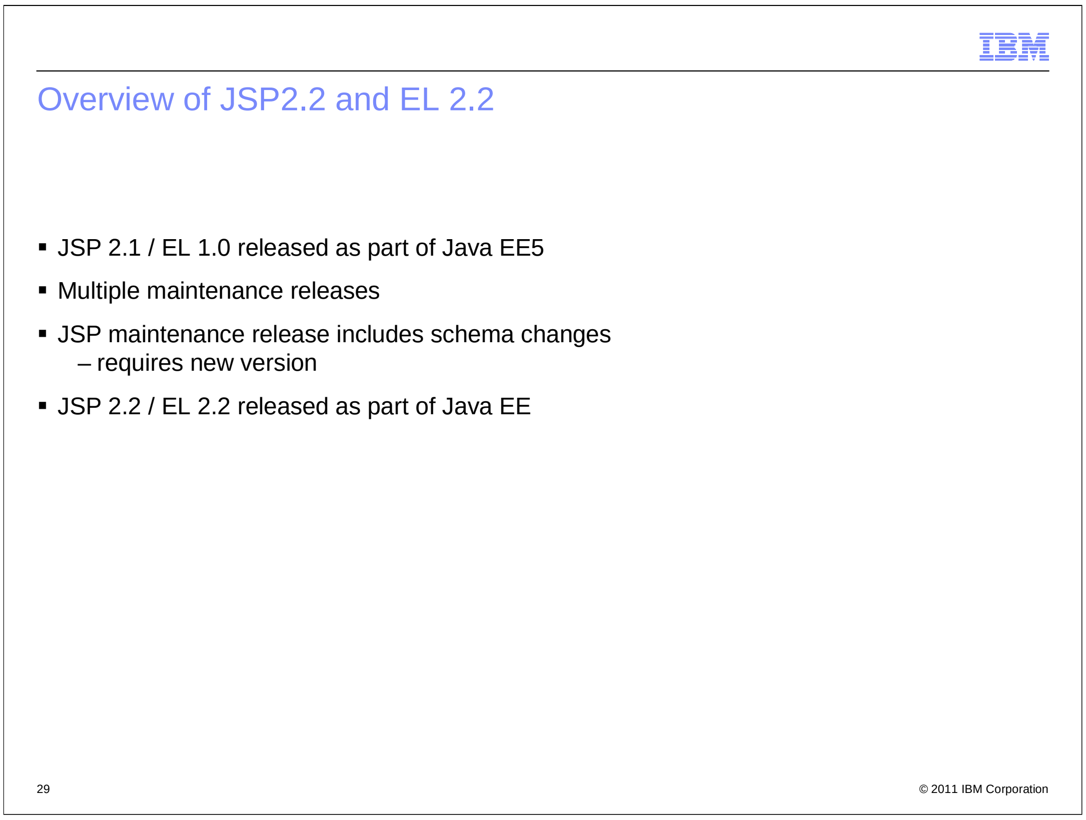
- PMI service enabled with extended statistics set
- Start monitoring the server WebApplication counters
- Enabled per servlet or in aggregate



Servlet counter definitions. This table displays properties for Servlet counters.

Name	Key	ID	Description	Granularity	Type	Level	Overhead
ResponseTime	webAppModule.servlets.asyncContextResponseTime	18	The response time (in milliseconds) for an AsyncContext associated with a servlet to complete.	Per servlet	TimeStatistic	Extended	Medium

This slide shows some of the available Servlet 3.0 PMI statistics which can be enabled using the Extended Statistics set. The statistics can be enabled either per servlet or in aggregate.



Overview of JSP2.2 and EL 2.2

- JSP 2.1 / EL 1.0 released as part of Java EE5
- Multiple maintenance releases
- JSP maintenance release includes schema changes
  – requires new version
- JSP 2.2 / EL 2.2 released as part of Java EE

29 © 2011 IBM Corporation

Since JSP 2.1 and its EL 1.0 were released as part of Java EE 5, there have been two additional Maintenance Releases (MRs) that contained some minor feature enhancements, clarifications and removal of errors for each specification.

As part of the first MR, the EL specification was no longer a part of the JSP specification and was re-versioned to 1.1. Then as part of Java EE 6, the MRs were folded back into the specifications. Since there were schema changes in the MRs, the JSR 245 expert group warranted a new point version of the specifications. JSP was then re-versioned to 2.2 and the EL was re-versioned to 2.2 also to keep them aligned.

JSP 2.2 changes

30

© 2011 IBM Corporation

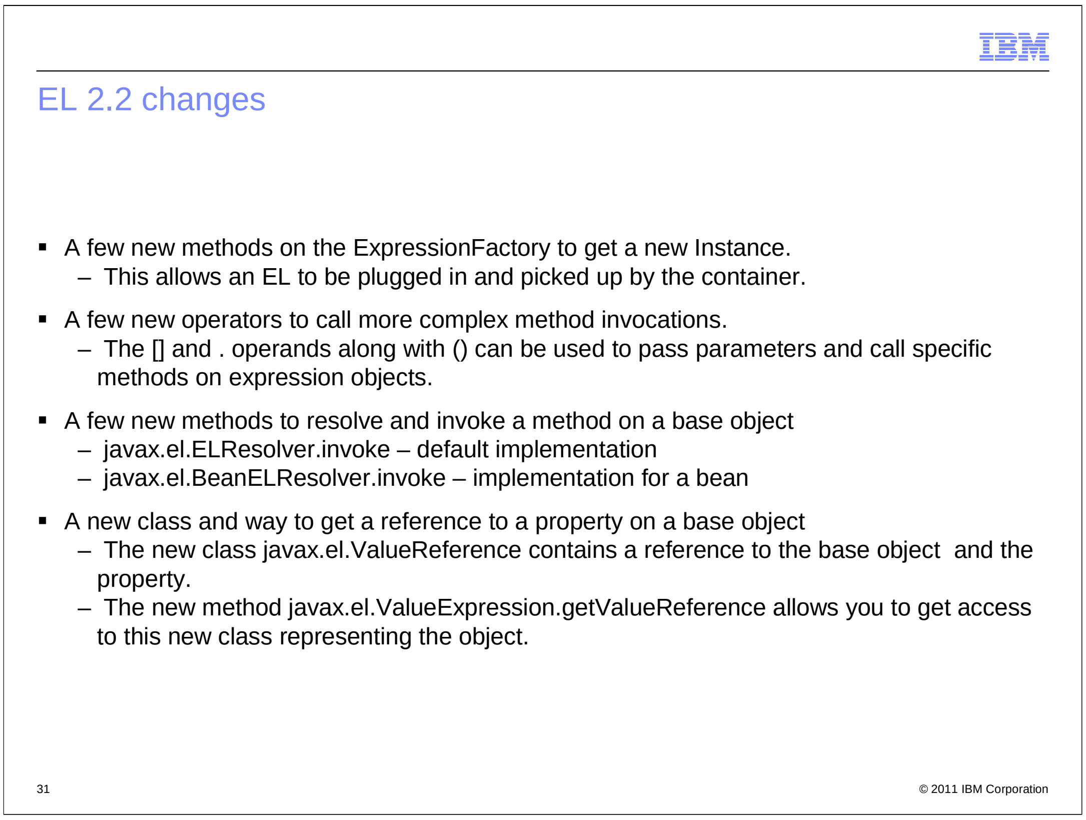
The JSP 2.2 changes include three new jsp-property-group sub-elements: buffer, default-content-type, and error-on-undeclared-namespace.

The Buffer element sets the size of the JSP writer. This can be adjusted for performance reasons.

The Default-content-type element sets the default content type for a collection of JavaServer Pages files that match the specified jsp-property-group.

The error-on-undeclared-namespace element makes it easier to debug namespace issues.

There was also the addition of the omit attribute on a <jsp attribute> within a <jsp:element> tag. This allows a JSP developer to omit certain attributes dynamically.



EL 2.2 changes

- A few new methods on the ExpressionFactory to get a new Instance.
  - This allows an EL to be plugged in and picked up by the container.
- A few new operators to call more complex method invocations.
  - The [] and . operands along with () can be used to pass parameters and call specific methods on expression objects.
- A few new methods to resolve and invoke a method on a base object
  - javax.el.ELResolver.invoke – default implementation
  - javax.el.BeanELResolver.invoke – implementation for a bean
- A new class and way to get a reference to a property on a base object
  - The new class javax.el.ValueReference contains a reference to the base object and the property.
  - The new method javax.el.ValueExpression.getValueReference allows you to get access to this new class representing the object.

31

The EL 2.2 changes include a few new methods on the ExpressionFactory to get a new instance. This allows an EL implementation to be plugged in and picked up by the container.

There are a few new operators introduced which allow you to call more complex method invocations.

There is a new invoke method in the ELResolver and BeanELResolver classes which can be used to resolve and invoke a method on a base object.

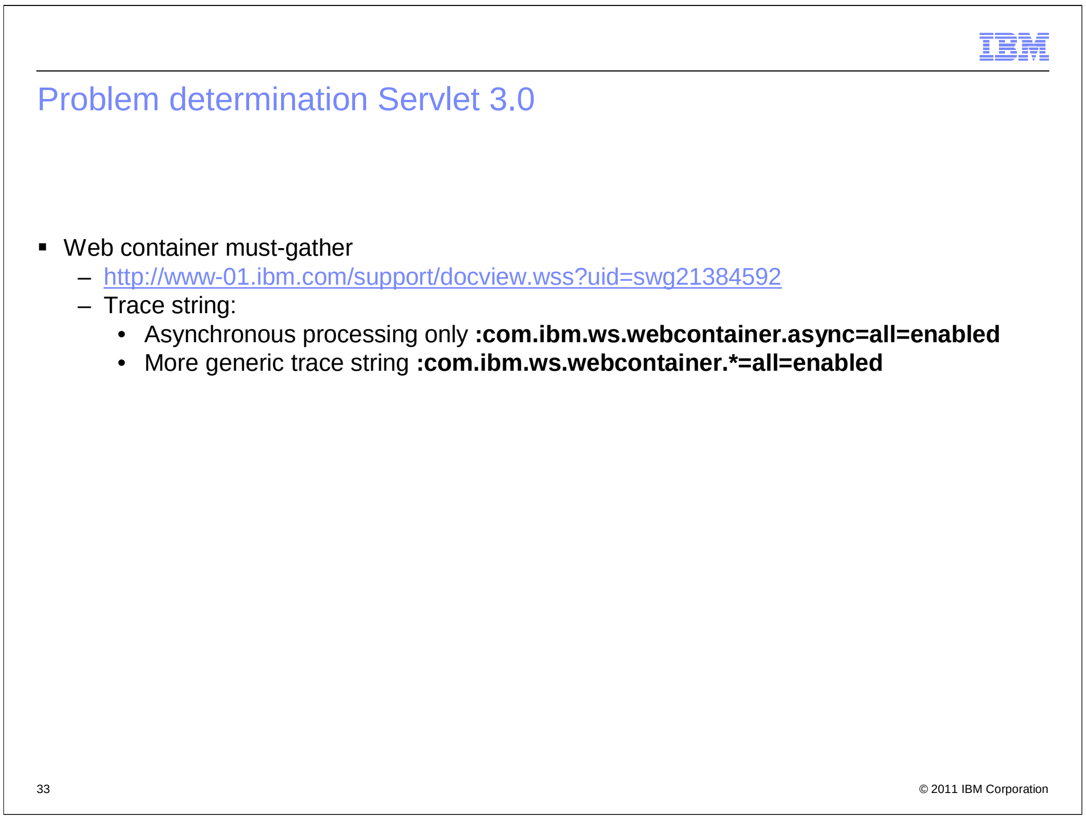
There is also a way to reference a property on a base object using the new ValueReference class.

Section

Problem determination

32

This section provides you with information that you should gather when investigating a servlet, JSP, or EL problem.

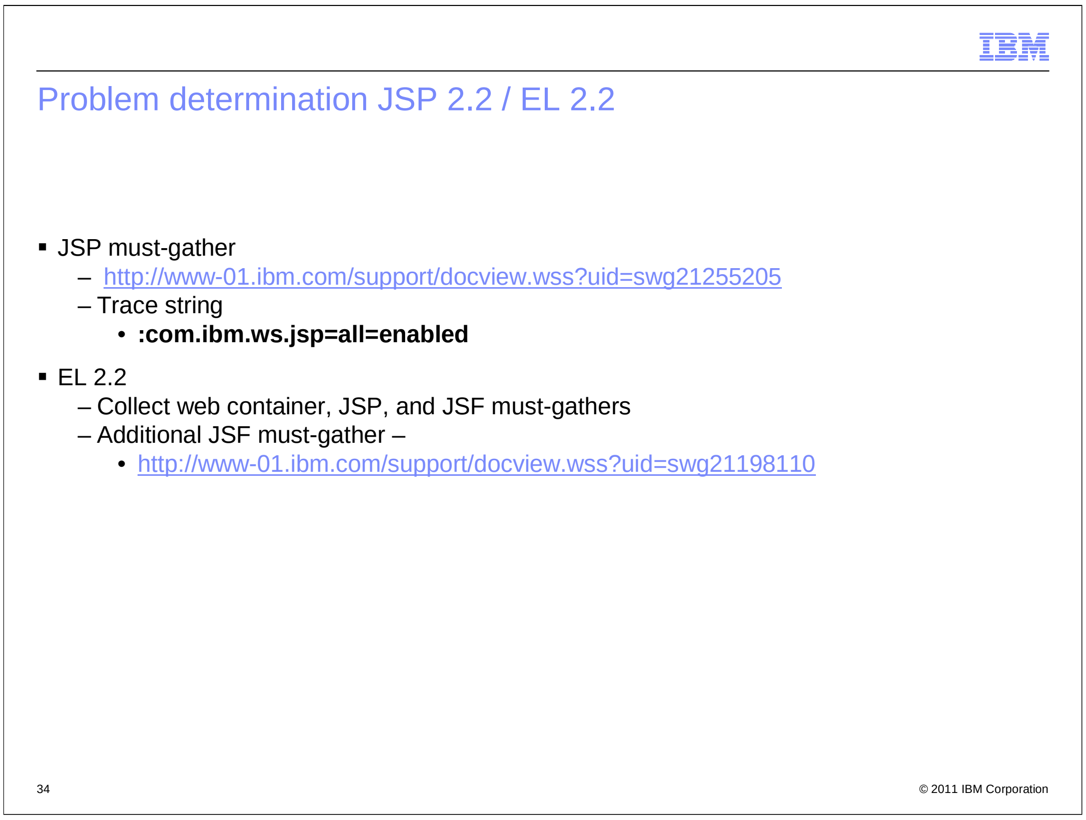


Problem determination Servlet 3.0

- Web container must-gather
  - http://www-01.ibm.com/support/docview.wss?uid=swg21384592
  - Trace string:
    - Asynchronous processing only **:com.ibm.ws.webcontainer.async=all=enabled**
    - More generic trace string **:com.ibm.ws.webcontainer.*=all=enabled**

33

This page has information that will help you in problem determination for Servlet 3.0.



Problem determination JSP 2.2 / EL 2.2

- JSP must-gather
  - http://www-01.ibm.com/support/docview.wss?uid=swg21255205
  - Trace string
    - **:com.ibm.ws.jsp=all=enabled**
- EL 2.2
  - Collect web container, JSP, and JSF must-gathers
  - Additional JSF must-gather –
    - http://www-01.ibm.com/support/docview.wss?uid=swg21198110

34

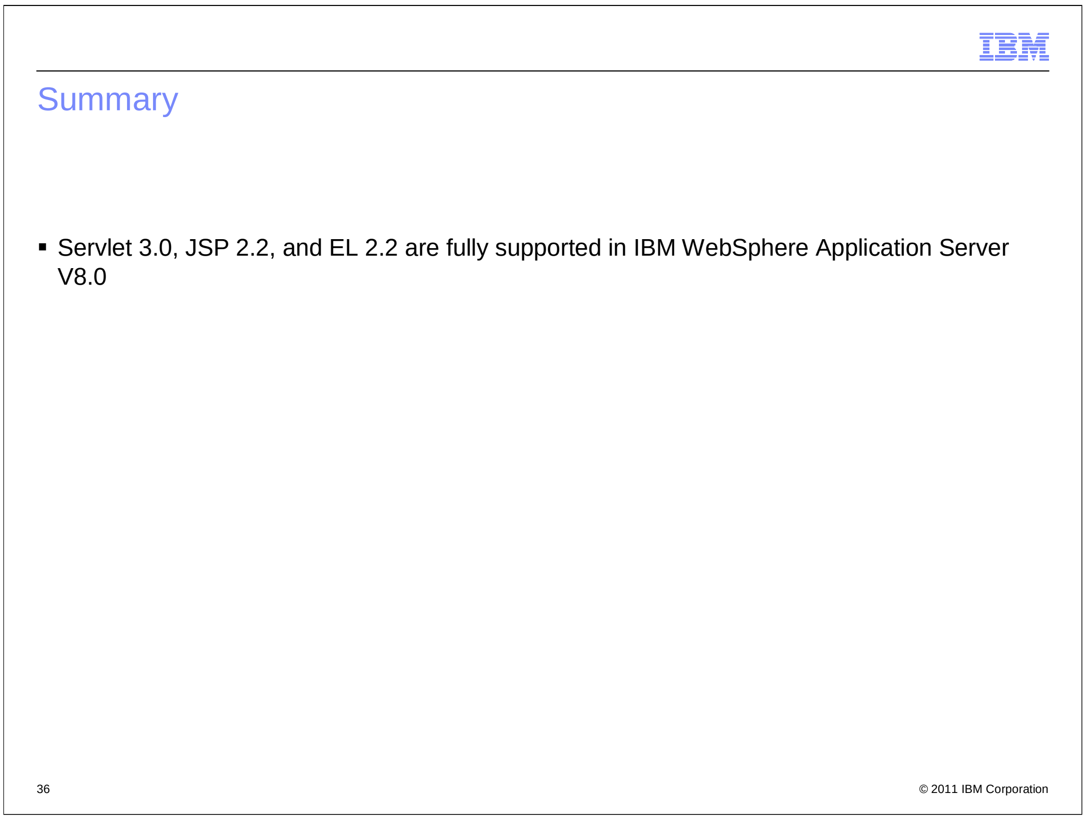
This page has information that will help you in problem determination for JSP 2.2 and EL 2.2.

Section

Summary

35

This section provides a summary of what you have learned in this presentation.

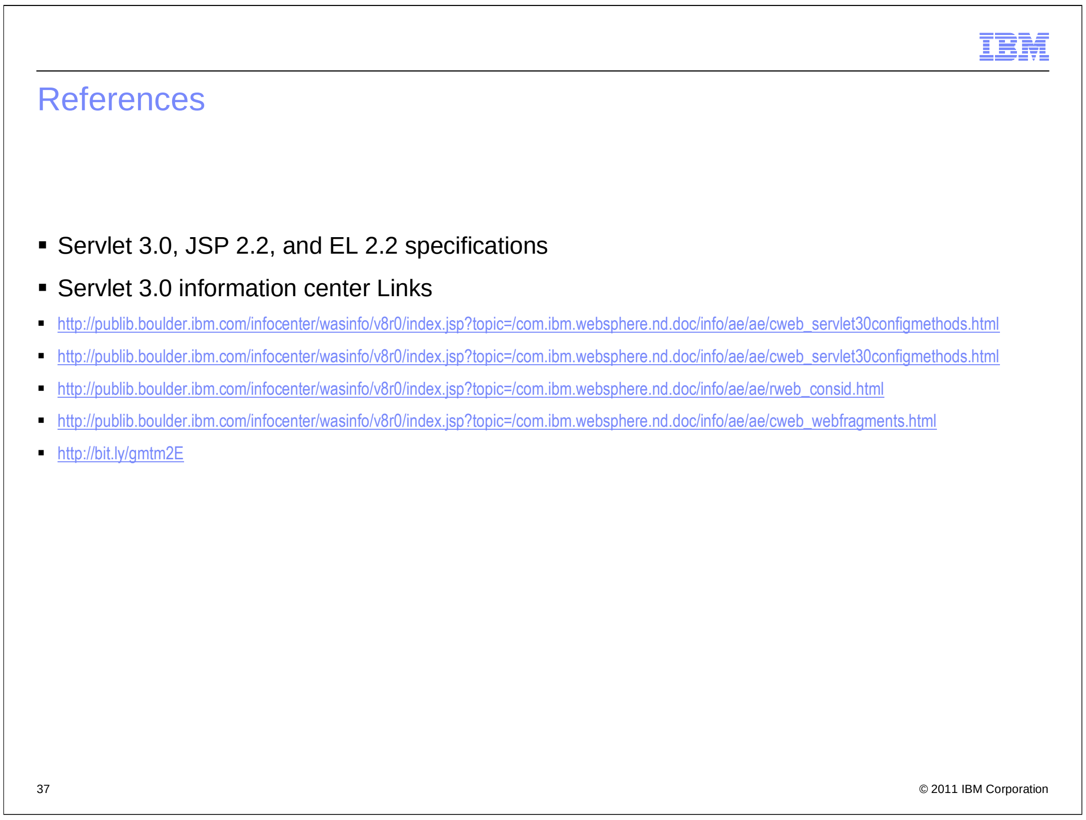


Summary

- Servlet 3.0, JSP 2.2, and EL 2.2 are fully supported in IBM WebSphere Application Server V8.0

36

In short, Servlet 3.0, JSP 2.2, and EL 2.2 are fully supported by IBM WebSphere Application Server version 8.0.

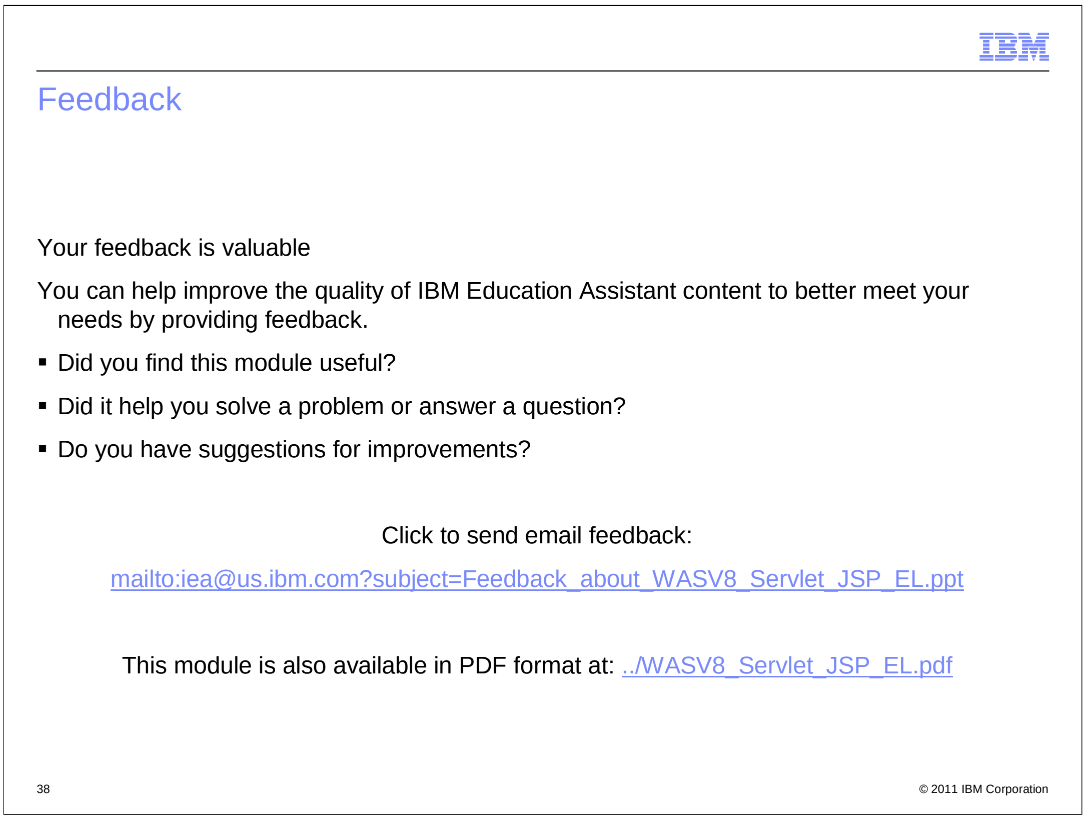


References

- Servlet 3.0, JSP 2.2, and EL 2.2 specifications
- Servlet 3.0 information center Links
- http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cweb_servlet30configmethods.html
- http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cweb_servlet30configmethods.html
- http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/rweb_consid.html
- http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cweb_webfragments.html
- http://bit.ly/gmtm2E

37 © 2011 IBM Corporation

Here are some references for Servlet 3.0.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

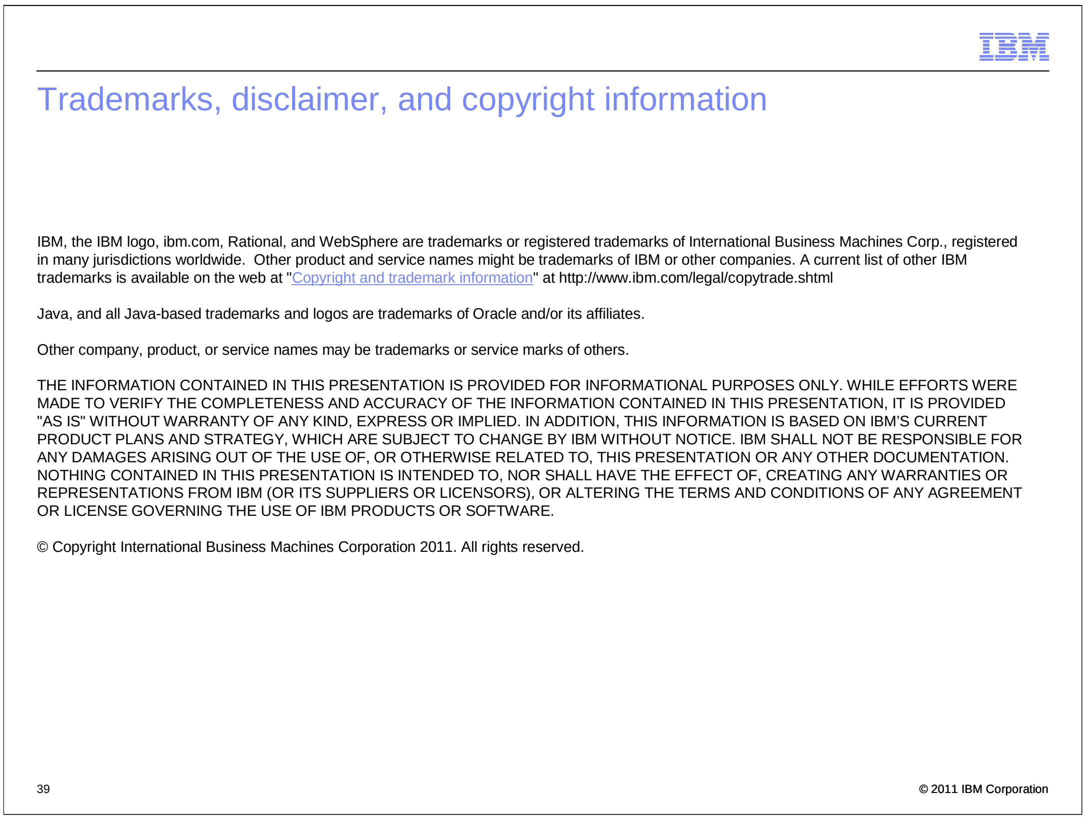
Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASV8_Servlet_JSP_EL.ppt

This module is also available in PDF format at: ../WASV8_Servlet_JSP_EL.pdf

38

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, disclaimer, and copyright information