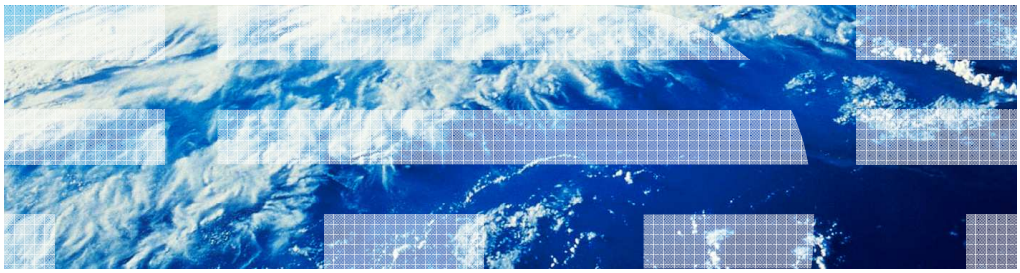
WebSphere Application Server V8

Servlet 3.0 security



WebSphere software

This presentation describes Servlet 3.0 security features in IBM WebSphere Application Server V8.

Table of contents

This presentation will cover Servlet 3.0 security in these basic sections. In the Overview, the features are presented at a high level. After that, the new features will be presented at a more detailed level with examples.

Section

Overview

This section describes the new features in Servlet 3.0 security at a high level.

Overview (1 of 2)

- Standard programmatic authentication methods
  - login()
  - logout()
  - authenticate()
- setHttpOnly() method
  - Provide a standard way to set the “httpOnly” cookie
- Display the realm name for basic HTTP logins
- http-method-omission
  - Exclude HTTP methods that should not use a set of security constraints

Several new security features are defined by the Servlet 3.0 specification. New login, authentication, and logout methods provide a standard programmatic mechanism for processing logins across implementations. The setHttpOnly method provides standard means of setting the “httpOnly” cookie to disallow cookie access by non-HTTP means. Servlet 3.0 also provides a way for containers to display the realm name to users in a login prompt when using basic HTTP authentication. The “HTTP method omission” element is a new and simple way to exclude HTTP methods from security constraints.

Overview (2 of 2)

- **Security annotations**
  - Provide flexibility for a developer to configure the security constraints directly in the servlet code.
- **Dynamic security annotations**
  - Provide a way to inject security constraints using standard APIs.

Servlet 3.0 provides the ability to use security-related annotations to specify security metadata in Servlet code rather than in the web deployment descriptor, and dynamic support for annotations allows for dynamic injection of security constraints and role mappings.

Section

Standard authentication methods

This section covers the standard authentication methods defined by Servlet 3.0.

HTTPServletRequest.login method

- This method allows an application to perform user name and password collection (as an alternative to form-based login) and to instigate authentication of the request caller by the container from within an unconstrained request context.
- login(String userName, String password)
  - If SSO is enabled, create cookie and add to the response
  - If HttpOnly is enabled, set HttpOnly cookie
  - If callerSubject is null, set callerSubject
  - Set invocationSubject
  - Push a client subject onto the thread.
- Information Center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/rsec_secgetru.html

The login method authenticates a user to the WebSphere Application Server with a user ID and password. If authentication is successful, it creates a user subject on the thread and Lightweight Third Party Authentication (LTPA) cookies (if single sign-on (SSO) is enabled). The “isUserInRole”, “getRemoteUser”, and “getAuthType” methods can be used to retrieve the user information after calling the login method. More detailed information on these methods can be found in the Information Center using the link shown here.

HTTPServletRequest.login example

- WebSphere Application Server V8:

```
req.login( loginUser, loginPassword);
invokeProtectedEJB();
```

- Before WebSphere Application Server V8:

```
LoginContext lc = new
 javax.security.auth.login.LoginContext("WSLogin",  new
 com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl(
 "loginUser", "securityrealm", "loginPassword"));
Subject sub = lc.getSubject();
WSSubject.doAs(sub, …) ; // invoke a protected EJB
```


This example shows how the new, standard login method in Servlet 3.0 can be used in WebSphere Application Server V8, as compared to the code for handling a login in previous versions.

HTTPServletRequest.logout method

- This method logs out the user and removes the subject
- logout()
  - Remove the user from authentication cache
  - Remove the user client subject from the thread
  - Set cookie maxAge to 0
  - If no cookie in HttpServletRequest, then remove cookie from the HttpServletResponse
  - Invalidate the session

The logout method logs the user out of the application and invalidates the HTTP session. WebSphere Application Server removes the user from the authentication cache and removes the user subject from the thread. It then clears the LTPA cookie if single sign-on is enabled and invalidates the HTTP session. Finally, the caller and invocation subjects are cleared, and the authentication type is set to null.

HTTPServletRequest.authenticate method

- This method allows an application to authenticate a user to WebSphere Application Server by using the web container login mechanism configured for the ServletContext.
- HTTPServletRequest Authenticate(HttpServletResponse res)
  - If SSO is enabled, create cookie and add to the response
  - If HttpOnly is enabled, set HttpOnly cookie
  - If callerSubject is null, set callerSubject
  - Set invocationSubject
  - Push a client subject onto the thread

The “authenticate” method authenticates a user by using the WebSphere Application Server container login mechanism configured for the Servlet context. During this process WebSphere Application Server creates the necessary cookies, sets the caller and invocation subjects, and pushes the client subject onto the appropriate thread.

HTTPServletRequest.authenticate example

- WebSphere Application Server V8:

```
authCheck = req.authenticate(response);
  invokeProtectedEJB();
```

- Before WebSphere Application Server V8:

```
LoginContext lc = new
  javax.security.auth.login.LoginContext("WSLogin", new
  com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl
  ());
lc.login();
Subject sub = lc.getSubject();
WSSubject.doAs(sub ,  …) // invoke a protected EJB
```


Similar to the previous example for the “login” method, this example shows how the new, standard “authenticate” method in Servlet 3.0 can be used in WebSphere Application Server V8, as compared to the authentication procedure in previous versions.

Section

HTTPOnly cookie support

This section covers support for the HTTPOnly cookie.

HttpOnly cookie support

- Use the Cookie.setHttpOnly() method to set the HttpOnly cookie.
- Global Security -> Single Sign-on (SSO)
  - Add a check box “HttpOnly cookie”
  - Set security custom property, com.ibm.ws.security.addHttpOnlyAttributeToCookies, to true if “HttpOnly cookie” box is checked. Default value is true

Servlet 3.0 security provides a standard method Cookie.setHttpOnly() that can be used to set the HttpOnly cookie programmatically. For more information on the HttpOnly cookie, see the presentation titled “WebSphere Application Server V8 security hardening”.

Section

Realm name support for HTTP login prompts

This section covers realm name support for HTTP login prompts.

Display the realm name in HTTP login prompts

- If the realm name is specified in the web.xml, this feature has no affect
  - Realm name in web.xml is displayed
- This feature is disabled by default, enabled using custom property
  - com.ibm.websphere.security.displayRealm = true
- If active authentication mechanism is Kerberos, the Kerberos realm name is displayed
- If active authentication mechanism is LTPA or SWAM, the WebSphere Application Server realm is displayed

Servlet 3.0 defines a simple way to display information about the user registry as the realm name within a basic HTTP login prompt, providing users with more information about the site to which they are logging in, if no realm name is specified in web.xml. If this property is set to true, the user registry's realm name is displayed to the user. For example, if stand-alone LDAP configuration is used, the LDAP server host name and port are displayed. For LocalOS-based security, the hostname is displayed. If a realm name is defined in web.xml, that realm name is displayed instead, regardless of whether this feature is enabled.

This section covers support for omitting certain HTTP methods from security constraints.

http-method-omission element in the web.xml file

- Support the <http-method-omission> element in the security-constraint

```
<security-constraint>
     <web-resource-collection>
          <web-resource-name>precluded methods</web-resource-name>
          <url-pattern>/acme/wholesale/*</url-pattern>
          <url-pattern>/acme/retail/*</url-pattern>
          <http-method-omission>GET</http-method-omission>
          <http-method-omission>POST</http-method-omission>
     </web-resource-collection>
      </security-constraint>
```

- For the above, everything except the GET and POST methods are excluded. GET and POST may have other security-constraint

The “http-method-omission” element within a security constraint provides a simple way to exclude one or more HTTP methods from the security constraint being defined. The example shown here excludes the GET and POST methods from the defined security constraint.

Support http-method-omission element in the web.xml file

- Method GET, DELETE, PUT, POST and HEAD have security constraint

```
<security-constraint>
   <web-resource-collection>
    <url-pattern>/acme/retail/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>DELETE</http-method>
    <http-method>PUT</http-method>
    <http-method>POST</http-method>
    <http-method>HEAD</http-method>
   </web-resource-collection>
   <auth-constraint/>
</security-constraint>
```

- The same security constraint above can be defined using http-method-omission as follows:

```
<http-method-omission>TRACE</http-method-omission>
<http-method-omission>OPTIONS</http-method-omission>
```


This slide shows an example of how to use the http-method-omission to exclude the TRACE and OPTIONS methods from the security-constraint, as opposed to using the http-method element to include all other elements. http-method-omission and http-method elements are never mixed in the same security-constraint collection.

This section covers support for security-related annotations in Servlet 3.0.

Security annotations @Inherited and @ServletSecurity

- At most one instance of the @ServletSecurity annotation may occur on a servlet implementation class
- @ServletSecurity annotation MUST NOT be specified on (that is, targeted to) a Java method
- @HttpConstraint and @HttpMethodConstraint are using only within the @ServletSecurity annotation
- @HttpConstraint followed by a list of zero or more @HttpMethodConstraint.

There are several rules that apply to the @Inherited and @ServletSecurity annotations. At most one instance of the @ServletSecurity annotation may occur on a servlet implementation class. The @ServletSecurity annotation must not be specified on (that is, targeted to) a Java method. The @HttpConstraint and @HttpMethodConstraint annotations are used only within the @ServletSecurity annotation. The @HttpConstraint annotation is followed by a list of zero or more @HttpMethodConstraint annotations.

Security annotation @ServletSecurity interface

```
public @interface ServletSecurity {
   HttpConstraint value();
   HttpMethodConstraint[] httpMethodConstraints();
 }
public @interface HttpConstraint {
   ServletSecurity.EmptyRoleSemantic value();
   java.lang.String[] rolesAllowed();
   ServletSecurity.TransportGuarantee transportGuarantee();
}
public @interface HttpMethodConstraint {
   ServletSecurity.EmptyRoleSemantic value();
   java.lang.String[] rolesAllowed();
   ServletSecurity.TransportGuarantee transportGuarantee();
 }
```


This slide describes the ServletSecurity, HttpConstraint, and HttpMethodConstraints interfaces.

@HttpConstraint and @HttpMethodConstraint value

- **emptyRoleSemantic**
  - The authorization semantic, either PERMIT or DENY, that applies when no roles are named in rolesAllowed. The default value for this element is PERMIT. DENY is not supported in combination with a non-empty rolesAllowed list.
- **rolesAllowed**
  - A list containing the names of the authorized roles. When this list is empty, its
  - meaning depends on the value of the emptyRoleSemantic. The role name "*"
  - has no special meaning when included in the list of allowed roles. The default
  - value for this element in an empty list
- **transportGuarantee**
  - The data protection requirements, either NONE or CONFIDENTIAL. This element is equivalent in meaning to a user-data-constraint containing a transport-guarantee with the corresponding value. The default value for this element is NONE.

This slide describes the mapping of the @HttpConstraint and @HttpMethodConstraint annotation values (defined for use within @ServletSecurity) to their corresponding auth-constraint and user-dataconstraint representations in web.xml file.

Web Servlet annotation @WebServlet

- Used to declare a Servlet.
- If there is a @WebServlet in a Servlet code, the urlPatterns is required.
- If there is no @WebServlet in a Servlet code and servlet-mapping defined in the web.xml, @ServletSecurity annotation is processed using the mapping information form web.xml file.
- If there is no @WebServlet in a Servlet code and no servlet-mapping defined in the web.xml; @ServletSecurity will be ignored

@WebServlet is used to define a Servlet component in a web application. The urlPatterns or the value attribute on the annotation must be present. All other attributes are optional with default settings. Refer to the javadoc for more details.

Servlet security annotations examples (1 of 4)

Example 1: for all HTTP methods, no constraints

```
@WebServlet (name=" Example ", urlPatterns={"/ Example "})
@ServletSecurity
public class Example extends HttpServlet {
}
```

Example 2: for all HTTP methods, no auth-constraint, confidential transport required

```
@WebServlet (name=" Example ", urlPatterns={"/ Example "})
@ServletSecurity(@HttpConstraint(transportGuarantee =
  TransportGuarantee.CONFIDENTIAL))
public class Example extends HttpServlet {
}
```


This slide provides two examples of Servlet 3.0 security annotations. The first example handles all HTTP methods with no constraints. The second example is for all HTTP methods, with no auth-constraint and confidential transport required.

Servlet security annotations examples (2 of 4)

Example 3: for all HTTP methods, all access denied

```
@WebServlet (name=" Example ", urlPatterns={"/ Example "})
@ServletSecurity(@HttpConstraint( EmptyRoleSemantic.DENY))
public class Example extends HttpServlet {
}
```

Example 4: for All HTTP methods except GET and POST, no constraints; for methods GET and POST, auth-constraint requiring membership in Role R1; for POST, confidential transport required

```
@WebServlet (name=" Example ", urlPatterns={"/ Example "})
@ServletSecurity((httpConstraints = {
   @HttpMethodConstraint(value = "GET", rolesAllowed = "R1"),
   @HttpMethodConstraint(value = "POST", rolesAllowed = "R1",
   transportGuarantee = TransportGuarantee.CONFIDENTIAL)
})
public class Example extends HttpServlet {
}
```


This slide provides two more examples of Servlet 3.0 security annotations.

The first example denies access for all HTTP methods.

In the second example all HTTP methods except GET and POST are handled with no constraints. For GET and POST requests, auth-constraint requiring membership in Role R1 and for POST, confidential transport is required.

Servlet security annotations examples (3 of 4)

Example 5: for all HTTP methods, auth-constraint requiring membership in Role R1

```
@WebServlet (name=" Example ", urlPatterns={"/ Example "})
@ServletSecurity(@HttpConstraint( rolesAllowed = "R1"))
public class Example extends HttpServlet {
}
```

Example 6: for All HTTP methods except GET and POST, no constraints; for methods GET , auth-constraint requiring membership in Role R1; for POST, all access denied

```
@WebServlet (name=" Example ", urlPatterns={"/ Example "})
@ServletSecurity((httpMethodConstraints = {
   @HttpMethodConstraint(value = "GET", rolesAllowed = "R1"),
    @HttpMethodConstraint(value=" POST“, emptyRoleSemantic =
   EmptyRoleSemantic.DENY))
})
public class Example extends HttpServlet {
}
```


This slide also provides two examples of Servlet 3.0 security annotations.

The first example has an auth-constraint requiring membership in Role R1 for all HTTP methods.

The second example handles all HTTP methods except GET and POST with no constraints. GET requests require membership in role R1, and all POST requests are denied.

Servlet security annotations examples (4 of 4)

Example 7: for all HTTP methods except GET auth-constraint requiring membership in Role R1; for GET, no constraints

```
@WebServlet (name=" Example ", urlPatterns={"/ Example "})
@ServletSecurity(value = @HttpConstraint(rolesAllowed = "R1"),
   httpMethodConstraints = @HttpMethodConstraint("GET"))
public class Example extends HttpServlet {
}
```

Example 8: for all HTTP methods except TRACE, auth-constraint requiring membership in Role R1; for TRACE, all access denied

```
@WebServlet (name=" Example ", urlPatterns={"/ Example "})
@ServletSecurity(value = @HttpConstraint(rolesAllowed = "R1"),
   httpMethodConstraints = @HttpMethodConstraint(value="TRACE",
   emptyRoleSemantic = EmptyRoleSemantic.DENY))
public class Example extends HttpServlet {
}
```


This slide shows more two examples of Servlet 3.0 security annotations.

The first example handles GET requests with no constraints. All other requests require membership in role R1.

The second example denies access to TRACE requests, and all other requests require membership in role R1.

Mapping @ServletSecurity to security constraint

- When metadata-complete attribute is set in DD, the @ServletSecurity annotation has no affect on security constraint.
- When a security-constraint in the portable deployment descriptor includes a urlPattern that is an exact match for a pattern mapped to a class annotated with @ServletSecurity, there is no effect on the constraint.
- With the exceptions listed above, when a Servlet class is annotated with @ServletSecurity, the annotation defines the security constraints that apply to all the urlPatterns mapped to all the Servlets mapped to the class.

This slide describes rules for mapping the @ServletSecurity annotation to a security constraint. Note that if the “metadata complete” attribute has been set in the deployment descriptor, the @ServletSecurity annotation will be ignored.

Mapping @ServletSecurity to security constraint

Example 1:
emptyRoleSemantic=PERMIT, rolesAllowed={}, transportGuarantee=NONE

web.xml:
no constraints

Example 2:
emptyRoleSemantic=PERMIT, rolesAllowed={},
transportGuarantee=CONFIDENTIAL

web.xml:

```
<user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```


This slide shows two examples of the @ServletSecurity annotation and their equivalents in the web.xml file.

The first example has no security constraints and data protection is NONE.

The second example has no security constraints and data protection is CONFIDENTIAL.

Mapping @ServletSecurity to security constraint (1 of 2)

Example 3:
emptyRoleSemantic=PERMIT, rolesAllowed={Role1}, transportGuarantee=NONE

web.xml:

```
<auth-constraint>
    <security-role-name>Role1</security-role-name>
</auth-constraint>
```

Example 4:
emptyRoleSemantic=PERMIT, rolesAllowed={Role1},
transportGuarantee=CONFIDENTIAL

web.xml

```
<auth-constraint>
    <security-role-name>Role1</security-role-name>
</auth-constraint>
<user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```


This slide shows two more examples of the @ServletSecurity annotation and their deployment descriptor equivalents.

The first example allows access for users in the role “Role1” and data protection is NONE.

The second example allows access for users in the role “Role1” and data protection is CONFIDENTIAL.

Mapping @ServletSecurity to security constraint (2 of 2)

Example 5:
emptyRoleSemantic=DENY, rolesAllowed={},
transportGuarantee=CONFIDENTIAL

web.xml

```
<auth-constraint/>
<user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```


This example will deny access for all roles, and CONFIDENTIAL data protection is required.

@Inherited

- The @ServletSecurity annotation may be specified on (that is, targeted to) a Servlet implementation class, and its value is inherited by sub classes according to the rules defined for the @Inherited meta-annotation.
- If you have the @ServletSecurity annotation on a super class, then the security constraint will apply to the sub classes that did not have any @ServletSecurity.
- If you want to override the security constraint in the sub class, then you can specify the @ServletSecurity

These are rules that apply to the @Inherited annotation. The @Inherited annotation can be used to specify that metadata specified by an @ServletSecurity annotation should be inherited by a Servlet class' subclasses.

Section

Dynamic security annotations

This section covers support for dynamic security annotations in Servlet 3.0.

Dynamic Servlet security annotations

- ServletRegistration.Dynamic
  - setServletSecurity(ServletSecurityElement constraint)
  - setRunAsRole(String roleName)
- ServletContext.declareRoles(String roleNames)
- These methods can be used within a ServletContextListener to define the security constraints to be applied to the mappings defined for a ServletRegistration
- These methods apply to all mappings added to this ServletRegistration up until the point that the ServletContext from which it WebSphere Application Server obtained has been initialized

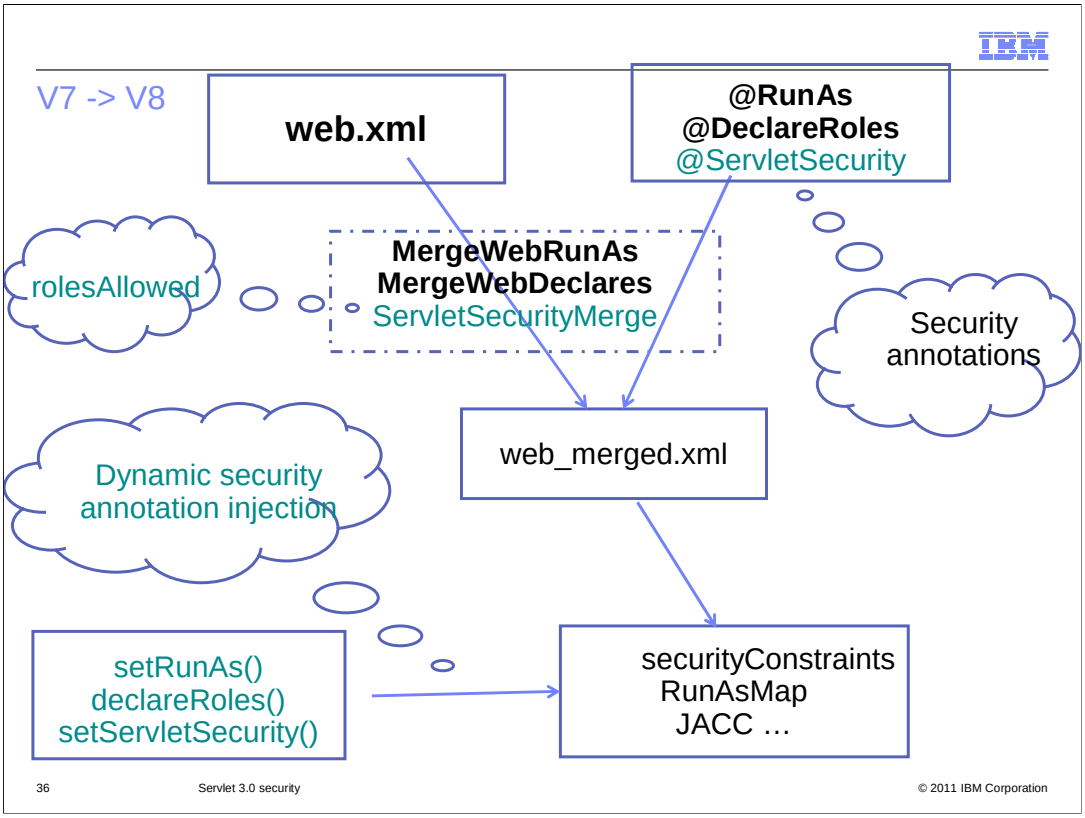
Servlet 3.0 security provides dynamic security annotations. You can use the methods setServletSecurity(), setRunAsRole and declareRoles to control access control in the dynamic servlet.

If the dynamic security annotations declareRoles, setRunAsRole and rolesAllowed, are used, the role name must be pre-defined, either through the deployment descriptor or through the @declareRoles and or @RunAs annotations in the servlet class. At deployment time, you can use the administrative console to map a user or group to this role.

Servlet security annotation merge action

- If a URL pattern of this ServletRegistration is an exact target of a security-constraint that WebSphere Application Server established in the portable deployment descriptor, then this method does not change the security-constraint for that pattern, and the pattern will be included in the return value.
- The setServletSecurity method returns the (possibly empty) Set of URL patterns that are already the exact target of a security-constraint element in the DD (and thus were unaffected by the call).
- When a security-constraint in the DD includes a URL pattern that is an exact match for a pattern mapped by a ServletRegistration, calls to setServletSecurity on the ServletRegistration must have no effect on the constraints enforced by the Servlet container on the pattern.

This slide describes how security-related annotations are merged with security-related metadata that is contained in the web deployment descriptor.



WebSphere Application Server V7 supported the @RunAs and @DeclareRoles security annotations; WebSphere Application Server V8 added support for the @ServletSecurity annotation. The ServletSecurityMerge function merges only the rolesAllowed; other security constraints are merged by the web container and security components.

@ServletSecurity example

```
HttpConstraintElement constraint = new HttpConstraintElement()
List<HttpMethodConstraintElement> methodConstraints =
          new ArrayList<HttpMethodConstraintElement>();

//Method GET permits role “Manager”
     methodConstraints.add(new HttpMethodConstraintElement("GET",
               new HttpConstraintElement(TransportGuarantee.NONE,
          new String[]{“Manager"})));

ServletSecurityElement servletSecurityElement =
                  new ServletSecurityElement(constraint,
    methodConstraints);

//Set the Servlet Security Constraints on servletRegistration.Dynamic
     servletRegDynamic.setServletSecurity(servletSecurityElement);
```


This slide provides an example of dynamic security annotation for the GET method that requires the role of Manager for a dynamic Servlet.

Summary

- Servlet 3.0 security updates in WebSphere Application Server V8
  - Display the realm name for basic HTTP login prompts
  - Login, authenticate, and logout methods
  - HTTP method omission element in web.xml
  - Security annotations
  - Dynamic security annotations

The Servlet 3.0 specification introduces several new security features, and WebSphere Application Server V8 implements support for these new features. When prompting users to login with HTTP, the realm name will now be displayed in the user's web browser. There is now also support for programmatically logging in, authenticating, and logging out users using standard methods. The "http-method-omission" element provides a simple way to exclude certain HTTP methods from security constraints using the web deployment descriptor. Support for security annotations allows developers to define security-related metadata directly in Servlet code, rather than using the web deployment descriptor. Dynamic support for annotations provides a means for dynamic injection of security constraints and role mappings from Java code.

Resources and references

- Servlet 3.0 specification

http://jcp.org/en/jsr/detail?id=315

- ServletRegistration dynamic Java document

http://java.sun.com/javaee/6/docs/api/javax/servlet/ServletRegistration.Dynamic.html

- Security annotation Java document

http://java.sun.com/javaee/6/docs/api/javax/servlet/annotation/ServletSecurity.html

- Servlet security methods section of the information center

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.base.doc/info/aes/ae/rsec_secgetru.html

See these resources and references for additional information on Servlet 3.0 security support in WebSphere Application Server V8.

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_Servlet30.ppt

This module is also available in PDF format at: ../WASV8_Servlet30.pdf

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

Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.