Web Application Programmer's Guide

JOnAS Team (Florent BENOIT)

- March 2009 -

Table of Contents

Chapter 1. Web Application Programmer's Guide

1.1. Target Audience and Content

The target audience for this guide is the Enterprise Bean provider, i.e. the person in charge of developing the software components on the server side and, more specifically, the web components.

1.2. Developing Web Components

1.2.1. Introduction

A Web Component is a generic term which denotes both JSP pages and Servlets. Web components are packaged in a `.war` file and can be deployed in a JOnAS server via the web container service. Web components can be integrated in a J2EE application by packing the `.war` file in an `.ear` file (refer to the J2EE Application Programmer's Guide [1]).

The JOnAS distribution includes a Web application example: `The EarSample example.`

The directory structure of this application is the following:

etc/xml	contains the web.xml file describing the web application
etc/resources/web	contains html pages and images; JSP pages can also be placed here.
src/java/org/ow2/jonas/examples/ear/web	servlet sources
src/java/org/ow2/jonas/examples/ear/(!web)	beans sources (Session, Message Driven, JPA Entity)

The bean directory is not needed if beans coming from another application will be used.

1.2.2. The JSP pages

Java Server Pages (JSP) is a technology that allows regular, static HTML, to be mixed with dynamically-generated HTML written in Java programming language for encapsulating the logic that generates the content for the page. Refer to the Java Server Pages™[2] and the Quickstart guide [3] for more details.

1.2.2.1. Example:

The following example shows a sample JSP page that lists the content of a cart.

```
<!-- Get the session -->
<%@ page session="true" %>

<!-- The import to use -->
<%@ page import="java.util.Enumeration" %>
<%@ page import="java.util.Vector"      %>
```

[1] j2eeprogrammerguide#j2ee.appAss
[2] http://java.sun.com/products/jsp/
[3] http://java.sun.com/products/jsp/docs.html

```
<html>
<body bgcolor="white">
  <h1>Content of your cart</h1><br>
  <table>
    <!-- The header of the table -->
    <tr bgcolor="black">
      <td><font color="lightgreen">Product Reference</font></td>
      <td><font color="lightgreen">Product Name</font></td>
      <td><font color="lightgreen">Product Price</font></td>
    </tr>

    <!-- Each iteration of the loop display a line of the table -->
    <%
      Cart cart = (Cart) session.getAttribute("cart");
      Vector products = cart.getProducts();
      Enumeration enum = products.elements();
      // loop through the enumeration
      while (enum.hasMoreElements()) {
          Product prod = (Product) enum.nextElement();
    %>
    <tr>
      <td><%=prod.getReference()%></td>
      <td><%=prod.getName()%></td>
      <td><%=prod.getPrice()%></td>
    </tr>
    <%
    } // end loop
    %>
  </table>
</body>
</html>
```

It is a good idea to hide all the mechanisms for accessing EJBs from JSP pages by using a proxy java bean, referenced in the JSP page by the `usebean` special tag. This technique is shown in the `alarm example`, where the .jsp files communicate with the EJB via a proxy java bean `ViewProxy.java`.

1.2.3. The Servlets

Servlets are modules of Java code that run in an application server for answering client requests. Servlets are not tied to a specific client-server protocol. However, they are most commonly used with HTTP, and the word "Servlet" is often used as referring to an "HTTP Servlet."

Servlets make use of the Java standard extension classes in the packages `javax.servlet` (the basic Servlet framework) and `javax.servlet.http` (extensions of the Servlet framework for Servlets that answer HTTP requests).

Typical uses for HTTP Servlets include:

- processing and/or storing data submitted by an HTML form,
- providing dynamic content generated by processing a database query,
- managing information of the HTTP request.

For more details refer to the Java™ Servlet Technology [4] and the Servlets tutorial [5].

1.2.3.1. Example:

The following example is a sample of a Servlet that lists the content of a cart. This example is the servlet version of the previous JSP page example.

```
import java.util.Enumeration;
import java.util.Vector;
import java.io.PrintWriter;
```

[4] http://java.sun.com/products/servlet/
[5] http://java.sun.com/docs/books/tutorial/servlets/TOC.html

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class GetCartServlet extends HttpServlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<html><head><title>Your cart</title></head>");
        out.println("<body>");
        out.println("<h1>Content of your cart</h1><br>");
        out.println("<table>");

        // The header of the table
        out.println("<tr>");
        out.println("<td><font color="lightgreen">Product Reference</font></td>");
        out.println("<td><font color="lightgreen">Product Name</font></td>");
        out.println("<td><font color="lightgreen">Product Price</font></td>");
        out.println("</tr>");

        // Each iteration of the loop display a line of the table
        HttpSession session = req.getSession(true);
        Cart cart = (Cart) session.getAttribute("cart");
        Vector products = cart.getProducts();
        Enumeration enum = products.elements();
        while (enum.hasMoreElements()) {
            Product prod = (Product) enum.nextElement();
            int prodId = prod.getReference();
            String prodName = prod.getName();
            float prodPrice = prod.getPrice();
            out.println("<tr>");
            out.println("<td>" + prodId + </td>);
            out.println("<td>" + prodName + </td>);
            out.println("<td>" + prodPrice + </td>);
            out.println("</tr>");
        }

        out.println("</table>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

1.2.4. Accessing an EJB from a Servlet or JSP page

Through the JOnAS web container service, it is possible to access an enterprise java bean and its environment in a J2EE-compliant way.

The following sections describe:

1. How to access the Remote Home interface of a bean.
2. How to access the Local Home interface of a bean.
3. How to access the environment of a bean.
4. How to start transactions in servlets.

Note that all the following code examples are taken from the `The EarSample example` provided in the JOnAS distribution.

1.2.4.1. Accessing the Remote Home interface of a bean:

In this example the servlet gets the Remote Home interface OpHome registered in JNDI using an EJB reference, then creates a new instance of the session bean:

```
import javax.naming.Context;
import javax.naming.InitialContext;

//remote interface
import org.objectweb.earsample.beans.secusb.Op;
import org.objectweb.earsample.beans.secusb.OpHome;

        Context initialContext = null;
        try {
            initialContext = new InitialContext();
        } catch (Exception e) {
            out.print("<li>Cannot get initial context for JNDI: ");
            out.println(e + "</li>");
            return;
        }
      // Connecting to OpHome thru JNDI
        OpHome opHome = null;
        try {
            opHome = (OpHome) PortableRemoteObject.narrow(initialContext.lookup
                     ("java:comp/env/ejb/Op"),OpHome.class);
        } catch (Exception e) {
            out.println("<li>Cannot lookup java:comp/env/ejb/Op: " + e + "</li>");
            return;
        }
        // OpBean creation
        Op op = null;
        try {
            op = opHome.create("User1");
        } catch (Exception e) {
            out.println("<li>Cannot create OpBean: " + e + "</li>");
            return;
        }
```

Note that the following elements must be set in the `web.xml` file tied to this web application:

```
  <ejb-ref>
    <ejb-ref-name>ejb/Op</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>org.objectweb.earsample.beans.secusb.OpHome</home>
    <remote>org.objectweb.earsample.beans.secusb.Op</remote>
    <ejb-link>secusb.jar#Op</ejb-link>
  </ejb-ref>
```

1.2.4.2. Accessing the Local Home of a bean:

The following example shows how to obtain a local home interface OpLocalHome using an EJB local reference:

```
//local interfaces
import org.objectweb.earsample.beans.secusb.OpLocal;
import org.objectweb.earsample.beans.secusb.OpLocalHome;


      // Connecting to OpLocalHome thru JNDI
        OpLocalHome opLocalHome = null;
        try {
            opLocalHome = (OpLocalHome)
                initialContext.lookup("java:comp/env/ejb/OpLocal");
        } catch (Exception e) {
            out.println("<li>Cannot lookup java:comp/env/ejb/OpLocal: " + e + "</li>");
            return;
        }
```

This is found in the `web.xml` file:

```
  <ejb-local-ref>
    <ejb-ref-name>ejb/OpLocal</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>org.objectweb.earsample.beans.secusb.OpLocalHome</local-home>
    <local>org.objectweb.earsample.beans.secusb.OpLocal</local>
    <ejb-link>secusb.jar#Op</ejb-link>
  </ejb-local-ref>
```

1.2.4.3. Accessing the environment of the component:

In this example, the servlet seeks to access the component's environment:

```
      String envEntry = null;
       try {
           envEntry = (String) initialContext.lookup("java:comp/env/envEntryString");
       } catch (Exception e) {
           out.println("<li>Cannot get env-entry on JNDI " + e + "</li>");
           return;
       }
```

This is the corresponding part of the `web.xml` file:

```
  <env-entry>
    <env-entry-name>envEntryString</env-entry-name>
    <env-entry-value>This is a string from the env-entry</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
```

1.2.4.4. Starting transactions in servlets:

The servlet wants to start transactions via the UserTransaction :

```
import javax.transaction.UserTransaction;

       // We want to start transactions from client: get UserTransaction
        UserTransaction utx = null;
       try {
           utx = (UserTransaction) initialContext.lookup("java:comp/UserTransaction");
       } catch (Exception e) {
           out.println("<li>Cannot lookup java:comp/UserTransaction: " + e + "</li>");
           return;
       }

       try {
           utx.begin();
           opLocal.buy(10);
           opLocal.buy(20);
           utx.commit();

       } catch (Exception e) {
           out.println("<li>exception during 1st Tx: " + e + "</li>");
           return;
       }
```

1.3. Defining the Web Deployment Descriptor

1.3.1. Principles

The Web component programmer is responsible for providing the deployment descriptor associated with the developed web components. The Web component provider's responsibilities and the application assembler's responsibilities are to provide an XML deployment descriptor that conforms to the deployment descriptor's XML schema as defined in the Java™ Servlet Specification Version 2.4. (Refer to `$JONAS_ROOT/xml/web-app_2_4.xsd` or http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd [6]).

To customize the Web components, information not defined in the standard XML deployment descriptor may be needed. For example, the information may include the mapping of the name of referenced resources to its JNDI name. This information can be specified during the deployment phase, within another XML deployment descriptor that is specific to JOnAS. The JOnAS-specific deployment descriptor's XML schema is located in `$JONAS_ROOT/xml/jonas-web-app_X_Y.xsd` .

[6] http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd

The file name of the JOnAS-specific XML deployment descriptor must be the file name of the standard XML deployment descriptor prefixed by `' jonas- '`.

The parser gets the specified schema via the classpath (schemas are packaged in the $JONAS_ROOT/lib/common/ow_jonas.jar file).

The standard deployment descriptor (web.xml) should contain structural information that includes the following:

- The Servlet's description (including Servlet's name, Servlet's class or jsp-file, Servlet's initialization parameters),
- Environment entries,
- EJB references,
- EJB local references,
- Resource references,
- Resource env references.

The JOnAS-specific deployment descriptor (jonas-web.xml) may contain information that includes:

- The JNDI name of the external resources referenced by a Web component,
- The JNDI name of the external resources environment referenced by a Web component,
- The JNDI name of the referenced bean's by a Web component,
- The name of the virtual host on which to deploy the servlets,
- The name of the context root on which to deploy the servlets,
- The compliance of the web application classloader to the java 2 delegation model or not.

<host> element: If the configuration file of the web container contains virtual hosts, the host on which the WAR file is deployed can be set.

<context-root> element: The name of the context on which the application will be deployed should be specified. If it is not specified, the context-root used can be one of the following:

- If the war is packaged into an EAR file, the context-root used is the context specified in the application.xml file.
- If the war is standalone, the context-root is the name of the war file (i.e, the context-root is jonasAdmin for jonasAdmin.war).

If the context-root is / or empty, the web application is deployed as ROOT context (i.e., http://localhost:9000/).

<java2-delegation-model> element: Set the compliance to the java 2 delegation model.

- If true: the web application context uses a classloader, using the Java 2 delegation model (ask parent classloader first).
- If false: the class loader searches inside the web application first, before asking parent class loaders.

1.3.2. Examples of Web Deployment Descriptors

- Example of a standard Web Deployment Descriptor (web.xml):

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/
j2ee/web-app_2_4.xsd"
         version="2.4">

  <servlet>
    <servlet-name>Op</servlet-name>
      <servlet-class>org.objectweb.earsample.servlets.ServletOp</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Op</servlet-name>
    <url-pattern>/secured/Op</url-pattern>
  </servlet-mapping>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
        <!-- Define the context-relative URL(s) to be protected -->
        <url-pattern>/secured/*</url-pattern>
        <!-- If you list http methods, only those methods are protected -->
        <http-method>DELETE</http-method>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
        <http-method>PUT</http-method>
    </web-resource-collection>
    <auth-constraint>
      <!-- Anyone with one of the listed roles may access this area -->
      <role-name>tomcat</role-name>
      <role-name>role1</role-name>
    </auth-constraint>
  </security-constraint>

  <!-- Default login configuration uses BASIC authentication -->
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Example Basic Authentication Area</realm-name>
  </login-config>

  <env-entry>
    <env-entry-name>envEntryString</env-entry-name>
    <env-entry-value>This is a string from the env-entry</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>

  <!-- reference on a remote bean without ejb-link-->
  <ejb-ref>
    <ejb-ref-name>ejb/Op</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>org.objectweb.earsample.beans.secusb.OpHome</home>
    <remote>org.objectweb.earsample.beans.secusb.Op</remote>
  </ejb-ref>

  <!-- reference on a remote bean using ejb-link-->
  <ejb-ref>
    <ejb-ref-name>ejb/EjbLinkOp</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>org.objectweb.earsample.beans.secusb.OpHome</home>
    <remote>org.objectweb.earsample.beans.secusb.Op</remote>
    <ejb-link>secusb.jar#Op</ejb-link>
  </ejb-ref>

  <!-- reference on a local bean -->
  <ejb-local-ref>
    <ejb-ref-name>ejb/OpLocal</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>org.objectweb.earsample.beans.secusb.OpLocalHome</local-home>
    <local>org.objectweb.earsample.beans.secusb.OpLocal</local>
    <ejb-link>secusb.jar#Op</ejb-link>
  </ejb-local-ref>
</web-app>
```

- Example of a specific Web Deployment Descriptor (jonas-web.xml):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<jonas-web-app xmlns="http://www.objectweb.org/jonas/ns"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.objectweb.org/jonas/ns
        http://www.objectweb.org/jonas/ns/jonas-web-app_4_0.xsd" >

  <!-- Mapping between the referenced bean and its JNDI name, override the ejb-link if
       there is one in the associated ejb-ref in the standard Web Deployment Descriptor
 -->
  <jonas-ejb-ref>
    <ejb-ref-name>ejb/Op</ejb-ref-name>
    <jndi-name>OpHome</jndi-name>
  </jonas-ejb-ref>

  <!-- the virtual host on which deploy the web application -->
  <host>localhost</host>

  <!-- the context root on which deploy the web application -->
  <context-root>web-application</context-root>
</jonas-web-app>
```

For advices about xml file writing, refer to Section A.1, “xml Tips”.

1.4. WAR Packaging

Web components are packaged for deployment in a standard Java programming language Archive file called a war file (Web ARchive), which is a jar similar to the package used for Java class libraries. A war has a specific hierarchical directory structure. The top-level directory of a war is the document root of the application.

The document root is where JSP pages, client-side classes and archives, and static web resources are stored. The document root contains a subdirectory called WEB-INF , which contains the following files and directories:

- web.xml : The standard xml deployment descriptor in the format defined in the Java Servlet 2.4 Specification. Refer to $JONAS_ROOT/xml/web-app_2_4.xsd .
- jonas-web.xml : The optional JOnAS-specific XML deployment descriptor in the format defined in $JONAS_ROOT/xml/jonas-web_X_Y.xsd .
- classes : a directory that contains the servlet classes and utility classes.
- lib : a directory that contains jar archives of libraries (tag libraries and any utility libraries called by server-side classes). If the Web application uses Enterprise Beans, it can also contain ejb-jars . This is necessary to give to the Web components the visibility of the EJB classes. However, if the war is intended to be packed in a ear , the ejb-jars must not be placed here. In this case, they are directly included in the ear . Due to the use of the class loader hierarchy, Web components have the visibility of the EJB classes. Details about the class loader hierarchy are described in JOnAS class loader hierarchy [7] .

1.4.1. Example

Before building a war file, the java source files must be compiled to obtain the class files (located in the WEB-INF/classes directory) and the two XML deployment descriptors must be written.

Then, the war file (<web-application>.war) is built using the jar command:

```
cd <your_webapp_directory>
jar cvf <web-application>.war *
```

During the development process, an 'unpacked version' of the war file can be used. Refer to Configuring Web Container Service [8] for information about how to use directories for the web application.

[8] configuration_guide.html#N10791

1.5. web service configuration

This service provides containers for the web components used by the Java EE applications.

JOnAS provides two implementations of this service: one for Jetty 6.x, one for Tomcat 6.x. It is necessary to run this service in order to use the JonasAdmin tool. A web container is created from a `war` file.

In development mode, as all other Java EE archives war archives can be deployed automatically as soon as they are copied under $JONAS_BASE/deploy and undeployed as soon as they has been removed from this location.

Here is the part of `jonas.properties` concerning the **web** service:

```
#
###################### JOnAS Web container service configuration
#
#  Set the name of the implementation class of the web container service.
jonas.service.web.class    org.ow2.jonas.web.tomcat6.Tomcat6Service
#jonas.service.web.class    org.ow2.jonas.web.jetty6.Jetty6Service

#  Set the XML deployment descriptors parsing mode for the WEB container
#  service (with or without validation).
jonas.service.web.parsingwithvalidation    true                          1

#  If true, the onDemand feature is enabled. A proxy is listening on the http port and will
 make actions like starting or deploying applications.
#  The web container instance is started on another port number (that can be specified) but
 all access are proxified.
#  It means that the web container will be started only when a connection is done on the
 http port.
#  The .war file is also loaded upon request.
#  This feature cannot be enabled in production mode.
jonas.service.web.ondemand.enabled    true                               2

#  The redirect port number is used to specify the port number of the http web container.
#  The proxy will listen on the http web container port and redirect all requests on this
 redirect port
#  0 means that a random port is used.
jonas.service.web.ondemand.redirectPort    0                             3
```

For customizing the **web** service, it is possible to:

1. Set or not the XML validation at the deployment descriptor parsing time.
2. Enable or not the onDemand feature. In addition of activating this global feature, each web application that has to be loaded on demand must declare the *on-demand* element in the JOnAS deployment descriptor (`WEB-INF/jonas-web.xml`) as below:

   ```
   <?xml version="1.0" encoding="ISO-8859-1"?>
   <jonas-web-app xmlns="http://www.objectweb.org/jonas/ns"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.objectweb.org/jonas/ns
     http://www.objectweb.org/jonas/ns/jonas-web-app_5_1.xsd">

     ...

     <!-- Load this application on demand (if enabled in the webcontainer service) -->
     <on-demand>true</on-demand>
   </jonas-web-app>
   ```

3. This property is specific to the onDemand feature. Useful to set the port number of the http web container in case of the port number defined in the web server configuration is used by the proxy.

Appendix A. Appendix

A.1. xml Tips

Although some characters, such as ">", are legal, it is good practice to replace them with XML entity references.

The following is a list of the predefined entity references for XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark