31

Guidelines for Oracle XML DB Applications in Java

Design guidelines are presented for writing Oracle XML DB applications in Java. This includes guidelines for writing and configuring Java servlets for Oracle XML DB.

> **Note:**
>
> The Oracle XML DB Repository is deprecated with Oracle Database 23ai.

- Overview of Oracle XML DB Java Applications
  You can use Java code either in a client or an application server, using the OCI driver for JDBC, or in the Java Virtual Machine (JVM).
- HTTP(S): Access Java Servlets or Directly Access XMLType Resources
  If a downstream client needs to work with XML in its textual representation then using HTTP(S) to either access Java servlets or directly access `XMLType` resources performs the best, especially if the XML node tree is not being manipulated much by the Java program.
- Use JDBC XMLType Support to Access Many XMLType Object Elements
  If a downstream client is an application that programmatically accesses many or most of the elements of an `XMLType` instance using Java, then use JDBC `XMLType` support for best performance. It is often easier to debug Java programs outside of the database server, as well.
- Use Servlets to Manipulate and Write Out Data Quickly as XML
  Oracle XML DB servlets are best used for applications that want to get into the database, manipulate the data, and write it out quickly as XML, not to format HTML pages for end-users.
- Oracle XML DB Java Servlet Support Restrictions
  The Oracle XML DB protocol server supports FTP, HTTP 1.1, WebDAV, and Java Servlets. It supports Java Servlet version 2.2, with a few exceptions.
- Configuration of Oracle XML DB Servlets
  Oracle XML DB servlets are configured using file `xdbconfig.xml` in Oracle XML DB Repository. Many of the XML elements in this file are the same as those defined by the Java Servlet 2.2 specification portion of Java 2 Enterprise Edition (J2EE), and they have the same semantics.
- HTTP Request Processing for Oracle XML DB Servlets
  Oracle XML DB processing of an HTTP request is described.
- Session Pool and Oracle XML DB Servlets
  Oracle Database uses one Java virtual machine (VM) for each database session. A session that is reused from the session pool retains any state that is left over in the Java VM (Java static variables) from the last time that session was used.
- Native XML Stream Support
  Java node class `DOM` has Oracle-specific method `write()`, which provides native XML stream support.

- Oracle XML DB Servlet APIs
  The APIs supported by Oracle XML DB servlets are described. They are defined by the Java Servlet 2.2 specification.
- Oracle XML DB Servlet Example
  Examples show the definition of a simple Oracle XML DB servlet that prints the content of a file resource, and how to register and map that servlet.

Overview of Oracle XML DB Java Applications

You can use Java code either in a client or an application server, using the OCI driver for JDBC, or in the Java Virtual Machine (JVM).

Because Java runs in the database in the context of the database server process, the ways you can deploy and run Java code are restricted to the following:

- You can run Java code as a stored procedure invoked from SQL or PL/SQL.
- You can run a Java servlet.

Stored procedures are easier to integrate with SQL and PL/SQL code. They require Oracle Net Services as the protocol to access Oracle Database.

Servlets work better as the top-level entry point into Oracle Database, and require using HTTP(S) as the protocol to access Oracle Database.

All Oracle XML DB application program interfaces (APIs) for Java are available to applications running both in the server and outside the database.

These APIs include:

- JDBC support for `XMLType`
- `XMLType` class
- Java DOM implementation

HTTP(S): Access Java Servlets or Directly Access XMLType Resources

If a downstream client needs to work with XML in its textual representation then using HTTP(S) to either access Java servlets or directly access `XMLType` resources performs the best, especially if the XML node tree is not being manipulated much by the Java program.

The Java implementation in the server can natively move data from the database to the network without converting character data through UCS-2 Unicode (which is required by Java strings). In many cases data is copied directly from the database buffer cache to the HTTP(S) connection. There is no need to convert data from the buffer cache into the SQL serialization format used by Oracle Net Services, then move it to the JDBC client, and then convert to XML. Loading on demand and the LRU cache for `XMLType` are most effective inside the database server.

Use JDBC XMLType Support to Access Many XMLType Object Elements

If a downstream client is an application that programmatically accesses many or most of the elements of an `XMLType` instance using Java, then use JDBC `XMLType` support for best performance. It is often easier to debug Java programs outside of the database server, as well.

Use Servlets to Manipulate and Write Out Data Quickly as XML

Oracle XML DB servlets are best used for applications that want to get into the database, manipulate the data, and write it out quickly as XML, not to format HTML pages for end-users.

Servlets are intended for writing HTTP stored procedures in Java that can be accessed using HTTP(S). If you need to develop an entire Internet application then deploy your application servlet in Oracle Fusion Middleware and have the servlet access data in the database using either JDBC or APIs such as `java.net.*`.

Oracle XML DB Java Servlet Support Restrictions

The Oracle XML DB protocol server supports FTP, HTTP 1.1, WebDAV, and Java Servlets. It supports Java Servlet version 2.2, with a few exceptions.

Support for Java Servlet version 2.2. has these restrictions:

- The servlet WAR file (`web.xml`) is not supported in its entirety. Some `web.xml` configuration parameters must be handled manually. For example, creating roles must be done using the `SQL CREATE ROLE` command.
- `RequestDispatcher` and associated methods are *not* supported.
- Method `HTTPServletRequest.getCookies()` is *not* supported.
- Only one `ServletContext` (and one `web-app`) is currently supported.
- Stateful servlets (and thus the `HttpSession` class methods) are *not* supported. Servlets must maintain state in the database itself.

Configuration of Oracle XML DB Servlets

Oracle XML DB servlets are configured using file `xdbconfig.xml` in Oracle XML DB Repository. Many of the XML elements in this file are the same as those defined by the Java Servlet 2.2 specification portion of Java 2 Enterprise Edition (J2EE), and they have the same semantics.

Table 31-1 lists the XML elements defined for the servlet deployment descriptor by the Java Servlet specification, along with extension elements supported by Oracle XML DB.

Table 31-1 XML Elements Defined for Servlet Deployment Descriptors

XML Element Name	Defined By	Supported?	Description	Comment
`auth-method`	Java	no	Specifies an HTTP authentication method required for access	N/A

Table 31-1 (Cont.) XML Elements Defined for Servlet Deployment Descriptors

XML Element Name	Defined By	Supported?	Description	Comment
`charset`	Oracle	yes	Specifies an IANA character set name	For example: ISO8859, UTF-8
`charset-mapping`	Oracle	yes	Specifies a mapping between a filename extension and a charset	N/A
`context-param`	Java	no	Specifies a parameter for a Web application	Not yet supported
`description`	Java	yes	A string for describing a servlet or Web application	Supported for servlets
`display-name`	Java	yes	A string to display with a servlet or Web application	Supported for servlets
`distributable`	Java	no	Indicates whether or not this servlet can function if all instances are not running in the same Java virtual machine	All servlets running in Oracle Database *must* be distributable.
`errnum`	Oracle	yes	Oracle error number	See *Oracle Database Error Messages Reference*
`error-code`	Java	yes	HTTP(S) error code	Defined by RFC 2616
`error-page`	Java	yes	Defines a URL to redirect to if an error is encountered.	Can be specified through an HTTP(S) error, an uncaught Java exception, or through an uncaught Oracle error message
`exception-type`	Java	yes	Classname of a Java exception mapped to an error page	N/A
`extension`	Java	yes	A filename extension used to associate with MIME types, character sets, and so on.	N/A
`facility`	Oracle	yes	Oracle facility code for mapping error pages	For example: ORA, PLS, and so on.
`form-error-page`	Java	no	Error page for form login attempts	Not yet supported
`form-login-config`	Java	no	Config spec for form-based login	Not yet supported
`form-login-page`	Java	no	URL for the form-based login page	Not yet supported
`icon`	Java	Yes	URL of icon to associate with a servlet	Supported for servlets
`init-param`	Java	Yes	Initialization parameter for a servlet	N/A
`jsp-file`	Java	No	Java Server Page file to use for a servlet	Not supported
`lang`	Oracle	Yes	IANA language name	For example: en-US
`lang-mapping`	Oracle	Yes	Specifies a mapping between a filename extension and language content	N/A
`large-icon`	Java	Yes	Large sized icon for icon display	N/A
`load-on-startup`	Java	Yes	Specifies if a servlet is to be loaded on startup	N/A

Table 31-1 (Cont.) XML Elements Defined for Servlet Deployment Descriptors

XML Element Name	Defined By	Supported?	Description	Comment
`location`	Java	Yes	Specifies the URL for an error page	Can be a local path name or HTTP(S) URL
`login-config`	Java	No	Specifies a method for authentication	Not supported
`mime-mapping`	Java	Yes	Specifies a mapping between filename extension and the MIME type of the content	N/A
`mime-type`	Java	Yes	MIME type name for resource content	For example: text/xml or application/octet-stream
`OracleError`	Oracle	Yes	Specifies an Oracle error to associate with an error page	N/A
`param-name`	Java	Yes	Name of a parameter for a Servlet or `ServletContext`	Supported for servlets
`param-value`	Java	Yes	Value of a parameter	N/A
`realm-name`	Java	No	HTTP(S) realm used for authentication	Not supported
`role-link`	Java	Yes	Specifies a role a particular user must have for accessing a servlet	Refers to a database role name. Make sure to capitalize by default!
`role-name`	Java	Yes	A servlet name for a role	Just another name to call the database role. Used by the Servlet APIs
`security-role`	Java	No	Defines a role for a servlet to use	Not supported. You must manually create roles using the SQL `CREATE ROLE`
`security-role-ref`	Java	Yes	A reference between a servlet and a role	N/A
`servlet`	Java	Yes	Configuration information for a servlet	N/A
`servlet-class`	Java	Yes	Specifies the classname for the Java servlet	N/A
`servlet-language`	Oracle	Yes	Specifies the programming language in which the servlet is written.	Either Java, C, or PL/SQL. Currently, only Java is supported for customer-defined servlets.
`servlet-mapping`	Java	Yes	Specifies a filename pattern with which to associate the servlet	All of the mappings defined by Java are supported
`servlet-name`	Java	Yes	String name for a servlet	Used by servlet APIs

Table 31-1 (Cont.) XML Elements Defined for Servlet Deployment Descriptors

XML Element Name	Defined By	Supported?	Description	Comment
`servlet-schema`	Oracle	Yes	The Oracle Schema in which the Java class is loaded. If not specified, then the schema is searched using the default resolver specification.	If this is not specified, then the servlet must be loaded into the `SYS` schema to ensure that everyone can access it, or the default Java class resolver must be altered. The servlet schema is capitalized unless the value is enclosed in double quotation marks.
`session-config`	Java	No	Configuration information for an `HTTPSession`	`HTTPSession` is not supported
`session-timeout`	Java	No	Timeout for an HTTP(S) session	`HTTPSession` is not supported
`small-icon`	Java	Yes	Small icon to associate with a servlet	N/A
`taglib`	Java	No	JSP tag library	JSPs currently not supported
`taglib-uri`	Java	No	URI for JSP tag library description file relative to file `web.xml`	JSPs currently not supported
`taglib-location`	Java	No	Path name relative to the root of the Web application where the tag library is stored	JSPs currently not supported
`url-pattern`	Java	Yes	URL pattern to associate with a servlet	See Section 10 of Java Servlet 2.2 spec
`web-app`	Java	No	Configuration for a Web application	Only one Web application is currently supported
`welcome-file`	Java	Yes	Specifies a welcome-file name	N/A
`welcome-file-list`	Java	Yes	Defines a list of files to display when a folder is referenced through an HTTP `GET` request	Example: `index.html`

> **Note:**
>
> - The following parameters defined for the `web.xml` file by Java are usable only by J2EE-compliant Enterprise Java Bean containers, and are not required for Java Servlet containers that do not support a full J2EE environment: `env-entry`, `env-entry-name`, `env-entry-value`, `env-entry-type`, `ejb-ref`, `ejb-ref-type`, `home`, `remote`, `ejb-link`, `resource-ref`, `res-ref-name`, `res-type`, `res-auth`.
> - The following elements are used to define access control for resources: `security-constraint`, `web-resource-collection`, `web-resource-name`, `http-method`, `user-data-constraint`, `transport-guarantee`, `auth-constrain`. Oracle XML DB provides this functionality through access control lists (ACLs). An ACL is a list of access control entries (ACEs) that determines which principals have access to a given resource or resources. A future release will support using a `web.xml` file to generate ACLs.

Related Topics

- Configuration of Oracle XML DB Using xdbconfig.xml
  Oracle XML DB is managed internally through a configuration file, `xdbconfig.xml`, which is stored as a resource in Oracle XML DB Repository. As an alternative to using Oracle Enterprise Manager to configure Oracle XML DB, you can configure it directly using the Oracle XML DB configuration file.

HTTP Request Processing for Oracle XML DB Servlets

Oracle XML DB processing of an HTTP request is described.

HTTP request handling proceeds as follows:

1. If a connection has not yet been established, then Oracle Listener hands the connection to a shared server dispatcher.
2. When a new HTTP request arrives, the dispatcher wakes up a shared server.
3. The HTTP headers are parsed into appropriate structures.
4. The shared server attempts to allocate a database session from the Oracle XML DB session pool, if available, but otherwise creates a new session.
5. A new database call and a new database transaction are started.
6. If HTTP(S) has included authentication headers, then the session is authenticated as that database user (just as if the user logged into SQL*Plus). If no authentication information is included, and the request is `GET` or `HEAD`, then Oracle XML DB attempts to authenticate the session as the `ANONYMOUS` user. If that database user account is locked, then no unauthenticated access is allowed.
7. The URL in the HTTP request is matched against the servlets in the `xdbconfig.xml` file, as specified by the Java Servlet 2.2 specification.
8. The Oracle XML DB Servlet container is invoked in the Java VM inside Oracle. If the specified servlet has not been initialized yet, then the servlet is initialized.
9. The Servlet reads input from the `ServletInputStream`, and writes output to the `ServletOutputStream`, and returns from method `service()`.

10. If no uncaught Oracle error occurred, then the session is put back into the session pool.

Related Topics

- Repository Access Using Protocols
  You can access Oracle XML DB Repository data using protocols FTP and HTTP(S)/ WebDAV.

Session Pool and Oracle XML DB Servlets

Oracle Database uses one Java virtual machine (VM) for each database session. A session that is reused from the session pool retains any state that is left over in the Java VM (Java static variables) from the last time that session was used.

This can be useful in caching Java state that is not user-specific, such as metadata, but do not store secure user data in Java static memory. This could turn into a security hole inadvertently introduced by your application if you are not careful.

Native XML Stream Support

Java node class `DOM` has Oracle-specific method `write()`, which provides native XML stream support.

Java method `write()` takes the following arguments and returns `void`:

- `java.io.OutputStream` stream: A Java stream for writing the XML text.
- String `charEncoding`: The character encoding for writing the XML text. If `NULL`, then the database character set is used.
- `Short indent` The number of characters to indent nested XML elements.

Method `write()` has a shortcut implementation if the stream is the `ServletOutputStream` provided inside the database. The contents of the Node are written as XML data in native code directly to the output socket. This bypasses any conversions into and out of Java objects or Unicode (required for Java strings), and provides very high performance.

Oracle XML DB Servlet APIs

The APIs supported by Oracle XML DB servlets are described. They are defined by the Java Servlet 2.2 specification.

The `Javadoc` for this is available at `http://download.oracle.com/javaee/1.2.1/api/index.html`.

Table 31-2 lists Java Servlet 2.2 methods that are not implemented. They result in run-time exceptions.

Table 31-2 Java Servlet 2.2 Methods that Are Not Implemented

Interface	Methods Not Implemented
`HttpServletRequest`	`getSession(), isRequestedSessionIdValid()`
`HttpSession`	all

Table 31-2 (Cont.) Java Servlet 2.2 Methods that Are Not Implemented

Interface	Methods Not Implemented
`HttpSessionBindingListener`	all

Oracle XML DB Servlet Example

Examples show the definition of a simple Oracle XML DB servlet that prints the content of a file resource, and how to register and map that servlet.

The servlet shown in Example 31-1 prints the content of file resource `/public/test/foo1.text`.

To install the servlet, you compile it, then load it into Oracle Database:

```
% loadjava -grant public -u quine/curry -r test.class
```

Finally, register and map the servlet, associating it with a URL, as shown in Example 31-2.

Example 31-1 An Oracle XML DB Servlet

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.util.*;
import java.io.*;
import java.util.*;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Reader;
import java.io.Writer;
import java.sql.DriverManager;
import java.sql.SQLException;
import oracle.jdbc.OracleConnection;
import oracle.jdbc.OracleDriver;
import oracle.jdbc.OraclePreparedStatement;
import oracle.jdbc.OracleResultSet;
import oracle.sql.CLOB;
import oracle.xdb.XMLType;
import oracle.xdb.spi.XDBResource;

public class test extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException,
           IOException {
      try {
        try {
          // Get the database connection for the current HTTP session
          DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
          OracleDriver ora = new OracleDriver();
          OracleConnection databaseConnection =
            (OracleConnection) ora.defaultConnection();
          String statementText =
            "SELECT XDBURIType('/public/test/foo1.txt').getClob() FROM DUAL";
          OraclePreparedStatement statement =
            (OraclePreparedStatement)
            databaseConnection.prepareStatement(statementText);
          OracleResultSet resultSet = null;
          CLOB content = null;
          // Execute the statement
          resultSet = (OracleResultSet) statement.executeQuery();
```

```
          while (resultSet.next())
            {// The statement returns a CLOB.
             // Copy content of CLOB to server's output stream.
             content = resultSet.getCLOB(1);
             Reader reader = content.getCharacterStream();
             Writer writer =
               new OutputStreamWriter(response.getOutputStream());
             int bytesSent = 0;
             int n;
             char[] buffer = new char[CLOB.MAX_CHUNK_SIZE];
             while (-1 != (n = reader.read(buffer)))
               { bytesSent = bytesSent + n;
                 writer.write(buffer, 0, n); }
             writer.flush();
             if (content.isOpen()) { content.close(); }}
          resultSet.close();
          statement.close();
          databaseConnection.close();
          response.getOutputStream().write('\n'); }
        catch (SQLException sql)
          { throw new ServletException(sql); }}
      catch (ServletException se )
        { se.printStackTrace(); }
      finally
        { System.out.flush(); }}}
```

Example 31-2 Registering and Mapping an Oracle XML DB Servlet

```
EXEC DBMS_XDB_CONFIG.addServlet('TestServletFoo', 'Java', 'TestServletFoo',
                                NULL, NULL, 'test', NULL, NULL, 'XDB');

EXEC DBMS_XDB_CONFIG.addServletMapping('/public/test/foo1.txt', 'TestServletFoo');

COMMIT;
```