XML Database Modelling (Part-II)

Unit - II

XML Database Solutions

- XML database solutions come in two flavors:
- Database mapping and native XML support

- **XML Database Mapping**
- provides a mapping between the XML document and the database field and vice versa
- Tool for mapping can be graphical or configuration file
- Eg. DB2 Extender, SQL Server 2000 Microsoft, Oracle 8i & 9i DataMirror DB/XML etc.

XML Database Solutions

- **Native XML Support**
- stores the XML data in the document in its native format.
- Use proprietary serialization technique to store the data.
- But, when the data is retrieved, it represents an XML document
- TEXTML, Oracle 8i and 9i, Excelon, dbXML Tamino etc.

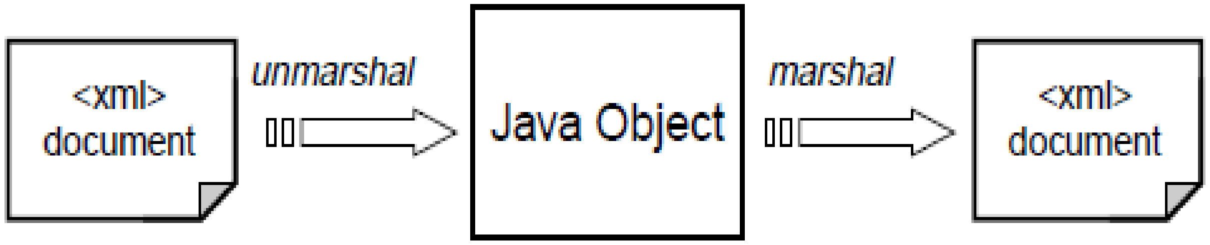
Modelling a database in XML using Java

- This example uses Java servlets and JDBC as server side components
- A key advantage to use servlets is thin-client interface
- JVM not preferred method due to compatibility issues
- XML data binding features of Java Architecture for XML Binding (JAXB) is used
- JAXB represents XML documents as Java objects
- Using the JAXB framework, well formed-ness can also be checked

Contd...

- JAXB framework, can parse XML documents into a suitable Java object – called *unmarshaling.*
- *The JAXB framework also provides the capability* to generate XML documents from Java objects – called *marshaling*
- Web Reference:

http://krazytech.com/programs/a-login-application-in-java-using-mo



JAXB over DOM & SAX

- SAX need content handler for each Xml document
- Complex Xml document, difficult development process

- *JAXB, parse an XML document by unmarshaling the data from an input stream.*

- DOM has complex API

- *Retrieve the data from XML document by calling a method on an object*
- *Also ensures type safety*

Steps to model database as an XML document

1. Review the database schema.
2. Construct the desired XML document.
3. Define a schema for the XML document.
4. Create the JAXB binding schema
5. Generate the JAXB classes based on the schema.
6. Develop a Data Access Object (DAO).
7. Develop a servlet for HTTP access.

Input XML

```
<rental_property>
<prop_id>1</prop_id>
<name>The Meadows</name>
<address>
<street>251 Eisenhower Blvd</street>
<city>Houston</city>
<state>TX</state>
<postal_code>77033</postal_code>
</address>
<contact>
<phone>555-555-1212</phone>
<fax>555-555-1414</fax>
</contact>
………………..
</rental_property>
```

```
<rental_property_list>
<rental_property> …
    </rental_property>
<rental_property> …
    </rental_property>
… …
</rental_property_list>
```

Step 4: Create JAXB binding schema

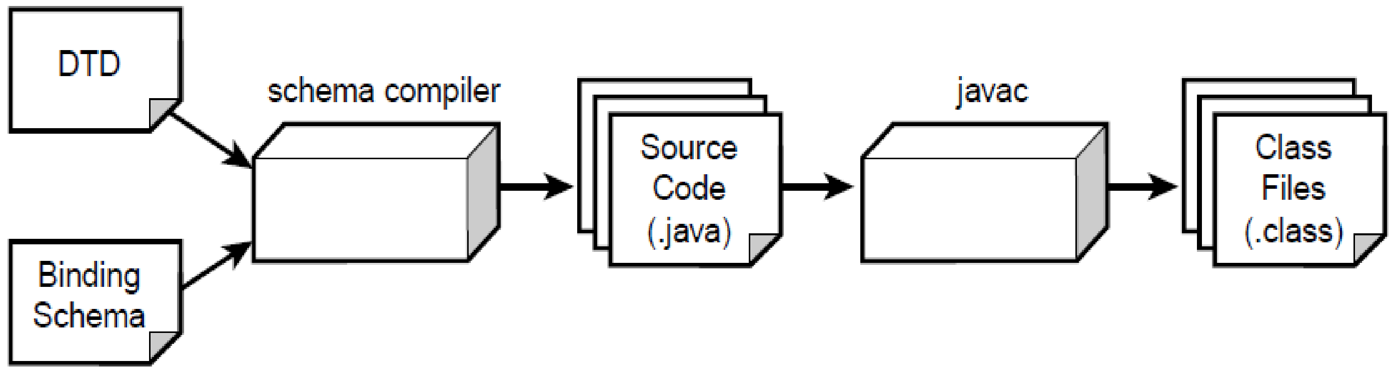
- The JAXB binding schema contains instructions on how to bind a DTD or Schema to a Java class
- Using this, we can define the names of the generated Java classes, map element names to specific properties in the Java class, and provide the mapping rules for attributes
- The following code example informs the JAXB system that the element <rental_property_list> should be mapped to a Java class and it is the root element for the XML document:

- ***<element name=" rental_property_list" type="class" root="true"/>***

Contd…

- Every element in the XML document not needed to be mapped
- JAXB uses a default binding schema that will *create properties in the Java class based on the XML element name*

- **<u>Generating the JAXB Classes Based on Schemas</u>**



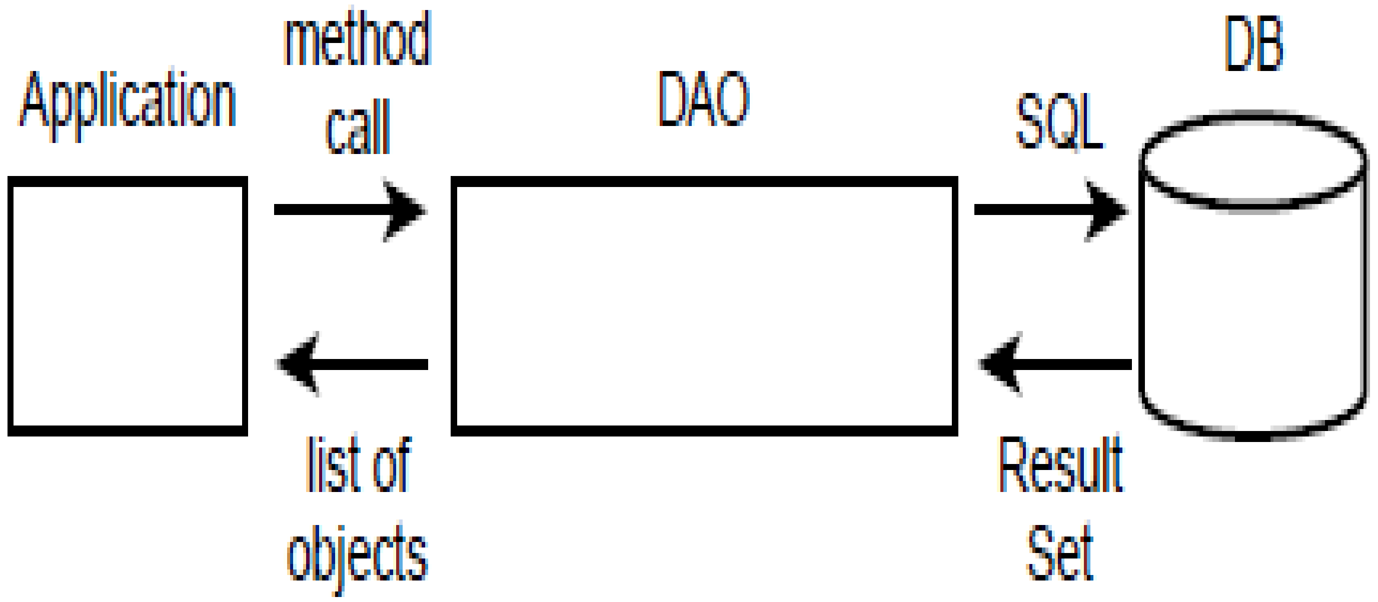
Contd…

- ***Command***
- *java com.sun.tools.xjc.Main rental_property.dtd rental_property.xjs -d source_code*
- Generates set of classes in source_code directory that models the element in XML
- **Output Classes generated**
- RentalPropertyList.java. This file models the <rental_property_list> element
- Ex. RentalProperty.java. This file models the <rental_property> element.
- Address.java. This file models the <address> subelement
- Contact.java. This file models the <contact> subelement

Developing a Data Access Object (DAO)

- DAO design pattern to provide higher level of abstraction for database access
- It provides access to the backend database
- DAO encapsulates the complex JDBC and SQL calls and provides access to backend database via public methods
- It converts a result set to a collection of objects.
- Objects model the data stored in the database

Data Access Design Pattern



Advantages of using DAO

- Details of the database are hidden from the application clients (schema, vendor) - form of Encapsulation

- Improved application maintenance – any modification / change in database need update only in DAO (no change in client program)

Example of DAO

```
public class RentalPropertyDAO {
…………..
public RentalPropertyDAO(String driverName, String dbUrl,
➥String user, String pass)
throws DAOException {
try {
// Load the driver
log(“Loading driver: “ + driverName);
Class.forName(driverName);
// Get a connection
log(“Connecting to the database: “ + dbUrl);
log(“User id: “ + user);
myConn = DriverManager.getConnection (dbUrl, user, pass);
…………………………………………………….
```

Contd…

```
public RentalPropertyList  getRentalProperties() throws DAOException {
RentalPropertyList theRentalPropertyList = new RentalPropertyList();
……………
try {
Statement myStmt = myConn.createStatement();
String rentalSql = “SELECT prop_num, name, street_address FROM
    rental_properties”;

ResultSet myRs = myStmt.executeQuery(rentalSql);
RentalProperty tempProperty = null;
// build a collection of JAXB RentalProperty objects
while (myRs.next()) {
tempProperty = createRentalProperty(myRs);
theList.add(tempProperty);
}
return theRentalPropertyList ;
```

Create JAXB objects based on result set (Mapping)

```
protected RentalProperty createRentalProperty (ResultSet theRs) throws DAOException {
RentalProperty theProperty = new RentalProperty();
Address theAddress = new Address();
Contact theContact = new Contact();
try {
// set the rental property number and name
theProperty.setPropId(theRs.getString(“prop_num”));
theProperty.setName(theRs.getString(“name”));

// set the address
theAddress.setStreet(theRs.getString(“street_address”));
theAddress.setCity(theRs.getString(“city”));
theAddress.setState(theRs.getString(“state”));
theAddress.setPostalCode(theRs.getString(“zip_code”));
theProperty.setAddress(theAddress);
…………………………………………………
```

Test Client

- public TestApp() throws DAOException {
- myRentalDAO = new RentalPropertyDAO();
- }
- public void process() throws DAOException, IOException {
- // Get the list of rental properties
- RentalPropertyList theList = myRentalDAO.getRentalProperties();
- // Send the XML data to standard out.
- theList.marshal(System.out);
- }

- public static void main(String[] args) {
- try {
- TestApp myApp = new TestApp();
- myApp.process();
- }

Output

```
<?xml version=”1.0” encoding=”UTF-8”?>
<rental_property_list>
<rental_property>
<prop_id>1</prop_id>
<name>The Meadows</name>
<address>
<street>251 Eisenhower Blvd</street>
<city>Houston</city>
<state>TX</state>
<postal_code>77033</postal_code>
</address>
<contact>
<phone>555-555-1212</phone>
<fax>555-555-1414</fax>
</contact>
</rental_property>
<rental_property>
…
</rental_property>
</rental_property_list>
```