DATABASE PROGRAMMING IN JAVA

Session: 6

Handling XML Data

Objectives

- Explain SQLXML and XML datatypes
- Explain XML Digital Signatures

Introduction

With the inclusion of the XML data type, an XML dataset could be one of the fields or column values in a row of database tables.

An SQL data type for handling XML type columns and a Java data type that maps to the SQL data type will enable the storing and retrieving of XML documents in a relational database.

JDBC version 4.0 and later supports a new data type, called SQLXML that maps to the new XML data type.

An XML digital signature can be used to sign XML data or any binary data, these are mainly used in Web based transactions.

SQLXML and XML Data Types 1-10

In the JDBC 4.0 specification, the `java.sql.Connection` interface allows the creation of an `SQLXML` object.

Add data to the object by invoking the `setString()`, `setBinaryStream()`, `setCharacterStream()`, or `setResult()` methods.

`getTypeInfo()` method of the `DatabaseMetaData` class, checks whether the data source associated with the connection supports SQL XML.

New methods have been added in the existing `Connection`, `ResultSet`, `CallableStatement`, and `PreparedStatement` interfaces to support XML data.

SQLXML and XML Data Types 2-10

- Following table shows the methods for handling XML data:

Method	Description
`Connection.createSQLXML()`	Creates an object with no data that implements the SQLXML interface.
`ResultSet.getSQLXML(int columnIndex)`	Retrieves the value as a SQLXML object from the specified column that is identified by column index.
`ResultSet.getSQLXML(String ColumnName)`	Retrieves the value as a SQLXML object from a specified column that is identified by column label.
`ResultSet.updateSQLXML(int ColumnIndex, SQLXML xmlObject)`	Updates the specified column that is identified by column index, with SQLXML value.
`ResultSet.updateSQLXML(String ColumnName,SQLXML xmlObject)`	Updates the specified column that is identified by column label, with SQLXML value.
`PreparedStatement .setSQLXML(int parameterIndex, SQLXML xmlObject)`	Sets the specified SQL parameter to the user-provided SQLXML object.
`CallableStatement.setSQLXML( String paramName, SQLXML xmlobject)`	Sets the specified SQL parameter to the user-provided SQLXML object.
`CallableStatement.getSQLXML(String paramName)`	Retrieves the value from a specified SQL parameter as a SQLXML object.

SQLXML and XML Data Types 3-10

- Following table describes methods in the new SQLXML interface for accessing XML values in databases as `String`, `Reader`, `Writer`, or `Stream` object:

Method	Description
`public InputStream getBinaryStream()`	Retrieves the XML value as an `InputStream` object. It returns InputStream object.
`public Reader getCharacterStream()`	Retrieves the XML value as a `Reader` object. It returns a Reader object.
`public String getString()`	Retrieves the XML value as a `String` object. It returns a String object.
`public OutputStream setBinaryStream()`	Retrieves a binary output stream that can be used to write the `XML` value as a stream. It returns a `OutputStream` object to which data can be written.
`public Writer setCharacterStream()`	Retrieves a binary output stream that can be used to write the `XML` value as a stream. It returns a `OutputStream` object to which data can be written.
`public void setString(String value)`	Sets the `XML` value to the provided `String` object.
`public void free()`	Releases all resources that are held up by an `SQLXML` object.

SQLXML and XML Data Types 4-10

- Following Code Snippet shows the details of the XML document `Orderdetails.xml`:

Code Snippet:

```
...
<order id="O101">
<items>
<item id="1">
<name>Book</name>
<quantity>5</quantity>
<unitprice>15.95</unitprice>
</item>
<item id="2">
<name>DVD Player</name>
<quantity>3</quantity>
<unitprice>103.95</unitprice>
</item>
</items>
</order>
...
```

SQLXML and XML Data Types 5-10

- Following Code Snippet shows how to create an SQLXML object and insert an XML document, `OrderDetails` into a column of a table using the `SQLXML` object:

Code Snippet:

```
...
public static void main(String[] args) {
String jdbcUrl =”jdbc:sqlserver://10.2.10.72\\SQLExpress;databaseName=MyR
hythmMusicLibrary;”;
String jdbcUser = “sa”;
String jdbcPassword = “playware_123”;
try {
Connection cn = DriverManager.getConnection(jdbcUrl, jdbcUser,
jdbcpassword);
Statement stmtDetails = cn.createStatement();
String strSQL = “Create table Invoice(Id int, OrderDetails xml)”;
boolean result = stmtDetails.execute(strSQL);
SQLXML order = cn.createSQLXML();
order.setString(“<order id =\”O101\”>”+
```

SQLXML and XML Data Types 6-10

```
"<items>" +
"<item id=\"1\">" +
"<name>Book</name>" +
"<quantity>5</quantity>" +
"<unitprice>15.95</unitprice>" +
"</item>"+
"<item id=\"2\">"+
"<name>DVD Player</name>"+
"<quantity>3</quantity>"+
"<unitprice>103.95</unitprice>"+
"</item>"+
"</items>"+
"</order>");
String SQLString = "Insert into Invoice(Id, OrderDetails) values(?,?)";
PreparedStatement pstDetails = cn.prepareStatement(SQLString);
pstDetails.setInt(1,1000);
pstDetails.setSQLXML(2,order);
pstDetails.executeUpdate();
order.free();
} catch(Exception e){}
}
}
...
```

SQLXML and XML Data Types 7-10

- Following Code Snippet shows how to retrieve an SQLXML object:

Code Snippet:

```
...
pstDetails = cn.prepareStatement("SELECT * FROM Invoice WHERE
Id=?");
pstDetails.setInt(1, 1000);
ResultSet rsDetails = pstDetails.executeQuery();
while(rsDetails.next()) {
SQLXML sqlXML = rsDetails.getSQLXML(2);
System.out.println(sqlXML.getString());
}
...
```

SQLXML and XML Data Types 8-10

- Following Code Snippet shows how to update an SQLXML object:

Code Snippet:

```
...
Connection cn = DriverManager.getConnection(jdbcUrl, jdbcUser,
  jdbcPassword);
Statement stmtDetails =
  cn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
SQLXML order = cn.createSQLXML();
order.setString("<order id=\"O101\">"+
"<items>"+
"<item id=\"3\">"+
"<name>MP3 Player</name>"+
"<quantity>3</quantity>"+
"<unitprice>12.45</unitprice>"+
"</item>"+
"</items>"+
"</order>");
```

SQLXML and XML Data Types 9-10

```
ResultSet rsDetails = stmtDetails.executeQuery("SELECT * from
  Invoice");
rsDetails.moveToInsertRow();
rsDetails.updateInt(1, 3);
rsDetails.updateSQLXML(2, order);
rsDetails.insertRow();
...
```

- Following Code Snippet shows how to retrieve an element type in XML: stream:

Code Snippet:

```
...
Connection cn = DriverManager.getConnection(jdbcUrl, jdbcUser,
jdbcPassword);
Statement stmtDetails =
cn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

SQLXML and XML Data Types 10-10

```
XMLStreamReader streamReader = null;
ResultSet rsDetails = stmtDetails.executeQuery("Select * from
  Invoice");
rsDetails.next();
SQLXML sqlXML = rsDetails.getSQLXML(2);
InputStream bStream = sqlXML.getBinaryStream();
XMLInputFactory factory = XMLInputFactory.newInstance();
XMLStreamReader streamReader =
  factory.createXMLStreamReader(bStream);
while (streamReader.hasNext()) {
int parseEvent = streamReader.next();
System.out.println("" + parseEvent);
}
...
```

XML Digital Signatures 1-4

A digital signature specifies the user that a digital message or a document is authentic. XML digital signatures can be used in XML transactions, to provide authentication and data integrity.

An XML signature can be used to sign a part of an XML document instead of the entire document.

With an XML signature you can sign different types of resources, such as HTML, JPG, XML document, and a part of an XML document.

The resource that needs to be digitally signed must have an Uniform Resource Identifier (URI).

XML Digital Signatures 2-4

Enveloped Signature

- An enveloped signature is a signature of a document that is embedded within the signed data.

Following figure graphically represents Enveloped Signature:

Enveloping Signature

- An enveloping signature is a signature within which the signed data is embedded.

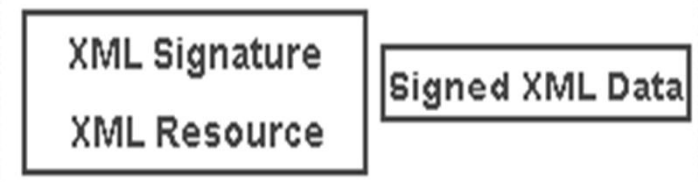
XML Digital Signatures 3-4

Following figure illustrates Enveloping Signature:

Detached Signature

- A detached signature is a signature where the signed data and signature stay separately from each other.

Following figure illustrates detached signature:



XML Digital Signatures 4-4

- An XML digital signature consists of different elements.
- Following figure shows various elements of XML Digital Signatures:

```
- <Signature>
  - <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
    - <Reference>
        <DigestMethod/>
      + <DigestValue>
      </Reference>
    </SignedInfo>
  + <SignatureValue>
  - <KeyInfo>
    + <Object>
    </KeyInfo>
  </Signature>
```

Creating XML Digital Signatures 1-6

Decide on the resources

- Decide on these resources that will require digital signatures in an HTML page.

Calculate resource digest

- The digest is a fixed-size bit string that represents the encoded resource. The resource digest is calculated using a relevant algorithm.

- Following Code Snippet demonstrates an XML signature:

Code Snippet:

```
...
<Reference URI="http://www.abc.com/image.gif">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>j6lxyz3rvEPO0vKtMup4ur=</DigestValue>
</Reference>
...
```

Creating XML Digital Signatures 2-6

Canonicalization

- An XML canonicalization algorithm ensures that the XML signature for two XML documents that differ only due to unnecessary white spaces will always be same.

- Following Code Snippet demonstrates canonicalization:

Code Snippet:

```
...
<SignedInfo Id = "s1">
<CanonicalizationMethod Algorithm = "http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
<SignatureMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#dsa-
sha1" />
<Reference URI = "http://www.abc.com/image.gif">
<DigestMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>j6lxyz3rvEPO0vKtMup4ur=</DigestValue>
</Reference>
</SignedInfo>
...
```

Creating XML Digital Signatures 3-6

Signing

- The digest of the <SignedInfo> element is calculated in this step, and then it is digitally signed and placed within a <SignatureValue> element.

Adding X.509 Certificate

- X.509 certificates include public keys for digital signature verification. This is included as <KeyInfo> element.

- Following Code Snippet shows the usage of <KeyInfo> element:

Code Snippet:

```
...
<KeyInfo>
<X509Data>
<X509SubjectName>CN=Robbert, O=Excel Inc.,ST=OTTAWA,C=CA
</X509SubjectName>
<X509Certificate>MXYZjabc0+gA...D56</X509Certificate>
</X509Data>
</KeyInfo>
...
```

Creating XML Digital Signatures 4-6

Wrapping of XML Signature

- The <Signature> element contains the generated XML digital signature.

- Following Code Snippet shows the structure of an XML document:

Code Snippet:

```
...
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo Id="s1">
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
<SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
<Reference URI="http://www.abc.com/image.gif">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
/>
```

Creating XML Digital Signatures 5-6

```
<DigestValue>j6lxyz3rvEPO0vKtMup4ur=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>XZ01TS=</SignatureValue>
<KeyInfo>
<X509Data>
<X509SubjectName>CN=Robbert, O=Excel Inc.,ST=OTTAWA,C=CA
</X509SubjectName>
<X509Certificate>MXYZjabc0+gA...D56</X509Certificate>
</X509Data>
</KeyInfo>
</Signature>
...
```

Creating XML Digital Signatures 6-6

Following figure shows the output of the XML document:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  - <SignedInfo Id="s1">
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
    - <Reference URI="http://www.abc.com/image.gif">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>j6lxyz3rvEPO0vKtMup4ur=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>XZ01TS=</SignatureValue>
  - <KeyInfo>
    - <X509Data>
        <X509SubjectName>CN=Robbert, O=Excel Inc.,ST=OTTAWA,C=CA</X509SubjectName>
        <X509Certificate>MXYZjabc0+gA...D56</X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
```

Validation of Digital Signatures 1-2

Reference Validation

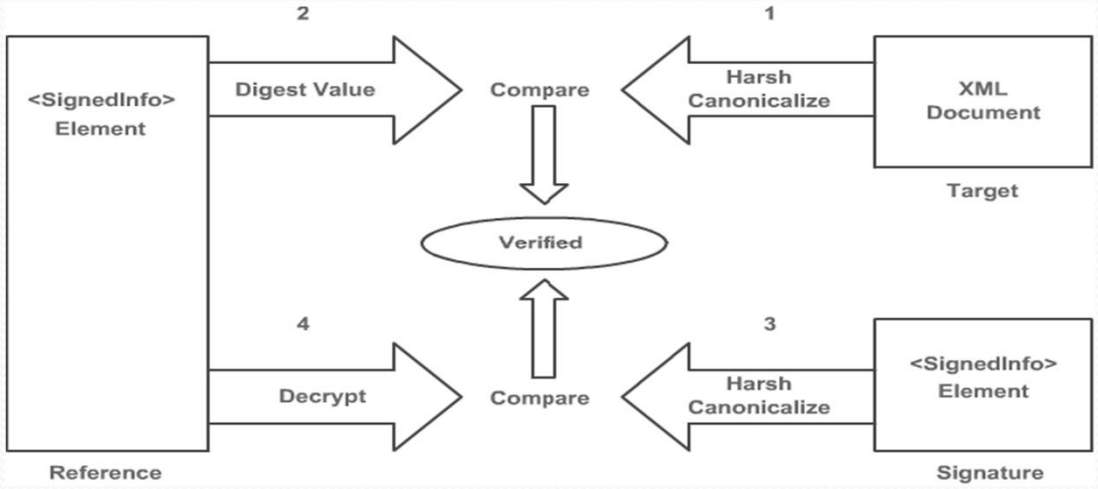
- Recalculate the digests of all references within the `<SignedInfo>` element and compare these to digest values in each `<Reference>` element's child element named `<DigestValue>`.

Signature Validation

- Recalculate the digest of the digital signature based on the algorithm in the `<SignatureMethod>` element. Then, compare the signature value to the value that is present in the `<SignatureValue>` element.

Validation of Digital Signatures 2-2

Following figure shows the validation process of an XML Signature:



An XML document can be tampered in any of the following ways:

- Modify the resource digest
- Modify the signature value
- Modify both resource digest and signature value

Java XML Digital Signature APIs 1-2

- The XML Digital Signature APIs consists of the following packages:

`javax.xml.crypto`	`javax.xml.crypto.dsig`	`javax.xml.crypto.dsig.keyinfo`
• This package consists of classes that can perform XML cryptographic operations, such as creating an XML signature or encrypting XML data.	• This package contains classes that represent the core classes and interfaces that can validate XML digital signatures.	• This package contains classes and interfaces for parsing and processing elements in `<KeyInfo>` structures in digitally-signed XML documents.

Java XML Digital Signature APIs 2-2

javax.xml.crypto.dsig.spec	javax.xml.crypto.dom and javax.xml.crypto.dsig.dom
• This package includes interfaces and classes that represent input parameters for algorithms, such as digest, signature, transform, and canonicalization.	• These packages contain DOM-specific classes for processing XML documents.

Classes and Interfaces 1-5

Following table lists the important classes and interfaces available in the XML digital signature API:

Class	Description
`XMLSignatureFactory`	A factory class for creating the `XMLSignature` objects. Its `getInstance()` method returns the `XMLSignatureFactory` object.
`Reference`	An interface that represents the `<Reference>` element in an XML document with digital signature. The `newReference()` method of `XMLSignatureFactory` creates Reference objects for resources, such as image files, XML documents.
`SignedInfo`	An interface that represents the `<SignedInfo>` element in an XML document with digital signature. The `newSignedInfo()` method of `XMLSignatureFactory` creates `SignedInfo` object.
`KeyPairGenerator`	The class generates pairs of public and private keys. Its `getInstance()` factory method creates an instance of `KeyPairGenerator` using algorithms, such as DSA. Its `initialize()` method initializes the key pair generator with specified parameter set.
`KeyPair`	The class stores public and private key pair.

Classes and Interfaces 2-5

Class	Description
KeyInfoFactory	The factory class that creates KeyInfo objects using its newKeyInfo() factory method. Its newKeyValue() method returns a KeyValue object from a public key. Its newKeyInfo() method returns a KeyInfo object for a public key.
KeyValue	The interface represents <KeyValue> element in an XML document with digital signature. It contains a public key that is required for validating a signature.
KeyInfo	The interface represents <KeyInfo> element in an XML document with digital signature.
XMLSignature	The interface represents <Signature> element in an XML document with digital signature. Its instance can be created using the newXMLSignature() method of XMLSignatureFactory factory class.
DocumentBuilderFactory	The factory class that allows applications to create a DOM tree representation of an XML document.
Document	The interface represents an entire HTML or XML document. It allows access to the data in a document.
DOMSignContext	This class contains methods to specify the location of an XMLSignature object in a DOM tree.
TransformerFactory	An instance of the class is used to create Transformer objects.
Transformer	An instance of the abstract class transforms a source DOM tree into a result DOM tree.

- Following Code Snippet shows how Java XML digital signatures APIs can be used to create an XML digital signature:

Code Snippet:

```
...
public class XMLDigitalSignature {
public static void main(String[] args) throws Exception {
XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
Reference ref =
fac.newReference("http://java.sun.com/javase/6/docs/api/
javax/xml/crypto/dsig/Reference.html",
fac.newDigestMethod(DigestMethod.SHA1, null));
SignedInfo si = fac.newSignedInfo(fac.newCanonicalizationMethod(
CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
(C14NMethodParameterSpec) null),
fac.newSignatureMethod(SignatureMethod.DSA_SHA1, null),
Collections. singletonList(ref));
KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
kpg.initialize(512);
```

Classes and Interfaces 4-5

```
KeyPair kp = kpg.generateKeyPair();
KeyInfoFactory kif = fac.getKeyInfoFactory();
KeyValue kv = kif.newKeyValue(kp.getPublic());
KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));
XMLSignature signature = fac.newXMLSignature(si, ki);
DocumentBuilderFactory dbf =
  DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
Document document = dbf.newDocumentBuilder().newDocument();
DOMSignContext digitalSignContext = new
  DOMSignContext(kp.getPrivate(), document);
OutputStream osStream = System.out;
signature.sign(digitalSignContext);
TransformerFactory tf = TransformerFactory.newInstance();
Transformer trans = tf.newTransformer();
trans.transform(new DOMSource(document), new
  StreamResult(osStream));
}
}
....
```

Following figure shows the output of the code:

Summary

- A new data type, called SQLXML that maps XML columns in a database has been introduced in JDBC version 4.0. The SQLXML interface implements methods for handling XML data types.
- XML Digital signatures provide authentication of source data and data integrity that are frequently used in XML transactions. XML digital signatures can be applied to both XML and non-XML data types.
- Java XML Digital Signatures APIs allow creating and verifying XML digital signatures.
- The Java XML Digital Signature APIs are the standard Java 7.0 APIs for creating and verifying XML Signatures.
- For creating a digital signature, a public and private key pair is generated using the generateKeyPair() method of KeyPairGenerator class using the DSA algorithm.