Dokumentation

6.6.5 - Webservices, Taschenbuch Datenbanken, Thomas Kudraß

Einführung:

In dem folgendem Beispiel, entnommen aus dem Taschenbuch Datenbanken von Thomas Kudraß aus dem Kapitel 6.6.5: Webservices von Thomas Rakow, geht es darum, einen Webservice zu simulieren, der vom Client aus auf eine Datenbank (Inhalte aus Kapitel 4, Taschenbuch Datenbanken) zugreift.

Dazu wird eine HTML-Website als Anfrage geschickt, wo zunächst eine kurze Beschreibung des HTTPs erfolgt, und danach eine Anfrage in Form von SOAP gesendet wird. Diese wird dann vom Server interpretiert und bearbeitet, und ebenfalls mit einer Website im selben Format beantwortet (s. S 236ff.).

Das Beispiel wurde hier mit der IDE Netbeans mit verschiedenen Frameworks nachgebildet. Benötigt werden dazu:

Netbeans 7.3.1 URL: https://netbeans.org/

JAX-WS (Netbeans-Framework) **Tutorial-URL**: https://netbeans.org/kb/docs/websvc/jax-ws.html

JAX-B (Netbeans-Framework) **Tutorial-URL**: https://netbeans.org/kb/docs/websvc/jaxb.html

In Netbeans muss zusätzlich ein Server eingerichtet werden. (Siehe Tutorial JAX-WS). In diesem Beispiel wurde dazu die Software von **Glassfish** verwendet. Außerdem müssen ggf. diverse **Web-Plugins** in Netbeans installiert werden. Dies kann einfach innerhalb von Netbeans über das Plugin-Menü (Tools -> Plugins) erledigt werden. Dort sollten der Einfachheit halber am besten sämtliche Plugins der Kategorie **Java Web and EE** ausgewählt werden.

Darüber hinaus ist ein Programm zur Erstellung einer Datenbank, in diesem Beispiel **Oracle** und der **SQL Developer** (http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html), sowie die notwendigen Kenntnisse in **SQL, Java, XML** und im Umgang mit **JDBC** erforderlich.

Falls das Beispiel manuell nachgebildet werden soll, wird empfohlen zunächst die **Tutorials** zu den beiden **Frameworks**, welche oben angegeben sind, durchzuarbeiten, da die Anwendung dieser im Beispiel sehr ähnlich erfolgt.

`Links vom Stand 18.10.2013`

Ziel:

Es geht darum, einen Webservice zu erstellen, der anhand der Übergabe eines Primärschlüssels, **via SOAP auf eine Datenbank zugreifen** kann und die entsprechenden Methoden auf diese anwendet, um dann die Ergebnisrelationen ebenfalls via SOAP an den Client zurück geben zu können. Dazu wird zunächst per **JDBC** auf die Datenbank zugegriffen und die so erhaltenen Daten werden in ein gerechtes Format für das SOAP gebracht. Dies geschieht über die Definition eines entsprechenden **WSDL**-Files in **XML**. Die so erhaltenen Daten können dann an den Client zurückgesendet werden.

Nachfolgend werden die hierzu notwendigen Schritte in der Reihenfolge erläutert, die auch für eine Implementierung notwendig wäre.

Erstellung der Datenbank:

Als erstes muss die Datenbank, welche später angefragt werden soll, erstellt werden. Hier wurde dazu der SQL Developer von Oracle verwendet.

Folgender Code ist dazu notwendig:

```
CREATE TABLE Artikel
(
  ANr NUMBER PRIMARY KEY,
  Bezeichnung VARCHAR2 (25)
);

CREATE TABLE Lieferant
(
  LNr NUMBER PRIMARY KEY,
  Name VARCHAR2(25)
);

CREATE TABLE Lieferung
(
  ANr NUMBER,
  LNr NUMBER,
  Preis NUMBER,
  PRIMARY KEY(ANr,LNr)
);
```

```
INSERT INTO Artikel VALUES (103,'Trinitron');

INSERT INTO Lieferant VALUES (3, 'Ziehm');
INSERT INTO Lieferant VALUES (4, 'Wegert');

INSERT INTO Lieferung VALUES (103,3,159.90);
INSERT INTO Lieferung VALUES (103,4,249.90);
```

Einrichten des XMLs:

Für die WSDL-Datei wurde folgender Code verwendet:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="db_artikellieferung" targetNamespace="http://j2ee.netbeans.org/wsdl/db_artikellieferung"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="http://j2ee.netbeans.org/wsdl/db_artikellieferung"
      xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
      xmlns:ns0="http://j2ee.netbeans.org/wsdl/db_artikellieferung">
      <types>
            <xsd:schema targetNamespace="http://j2ee.netbeans.org/wsdl/db_artikellieferung"
             xmlns:tns1="http://j2ee.netbeans.org/wsdl/db_artikellieferung">
                  <xsd:element name="Result">
                        <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                              <xsd:sequence>
                                    <xsd:element name="AnzZeilen">
                                          <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                                                <xsd:sequence>
                                                      <xsd:element name="AnzZeilen" type="xsd:int"></xsd:element>
                                                </xsd:sequence>
                                           </xsd:complexType>
                                    </xsd:element>
                                    <xsd:element name="Zeile">
                                          <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
                                                <xsd:sequence>
                                                      <xsd:element name="AnzL" type="xsd:int"></xsd:element>
                                                      <xsd:element name="ANr" type="xsd:int"></xsd:element>
                                                      <xsd:element name="LNr" type="xsd:int"></xsd:element>
                                                      <xsd:element name="Preis" type="xsd:double"></xsd:element>
                                                </xsd:sequence>
                                          </xsd:complexType>
                                    </xsd:element>
                              </xsd:sequence>
                        </xsd:complexType>
                  </xsd:element>
            </xsd:schema>
      </types>
</definitions>
```

Das so erstelle XML-Dokument wird als eine WSDL-Datei abgespeichert und später per JAX-B in das Projekt eingebunden.

Erstellung des Projekts und einrichten des WS

Zuerst wird ein neues Projekt erstellt und das Framework **JAX-WS** eingebunden. Hierzu kann das angegebene Tutorial mit entsprechenden Anpassungen verfolgt werden. Die Methode hello() sollte dabei umbenannt werden (z.B. in SelectLieferungVonArtikel()), denn diese umfasst später den kompletten Webservice. Um den Inhalt der Methode kümmern wir uns später.

Danach kann das Framework **JAX-B** geladen werden. Auch hier kann sich an das Tutorial gehalten werden. Dabei muss das XML Dokument, welches wir zuvor erstellt haben, eingelesen werden. Aus dem XML werden dadurch neue Java-Klassen erstellt. Es ist darauf zu achten, das Package, welches diese Klassen enthält, in die Webservice-Klasse zu importieren. *(Es kann passieren, dass in der Java-Klasse "Zeile" als Datentyp überall BigInteger verwendet wird. Wenn dies der Fall ist, müssen diese manuell in int geändert werden.)*

Einrichten von JDBC

Als nächstes richten wir JDBC ein, um über Java mit unserer Datenbank kommunizieren zu können.

Zunächst müssen wir dazu eine Verbindung zu unserer Datenbank erstellen. Vereinfacht können wir dazu diese Database Connector Klasse verwenden. Hier müssen nur noch die Einstellungen für den Login und evtl. den Port getätigt werden.

```
import java.sql.*;

public class Java_Database_Connector
{


    // Instanzvariablen muessen manuell angepasst werden!!
    private String host = "localhost";
    private String port =  "1521";
    private String sid = "xe";
    private String user, password;
    private String connectorString = "jdbc:oracle:thin:@" + host + ":" + port + ":" + sid;
    // Instanzvariable für den JDBC-Treiber
    // (siehe gleichnamiges Interface in API)
    private Connection con;
    // Instanzvariable für Metadatenabfrage
    // (siehe gleichnamiges Interface in API)
    private DatabaseMetaData dmd;

    /**
     * Konstruktor für das Erzeugen einer Datenbank-Connector-Instanz
     *
     * @param _user
     *            Benutzername der Datenbank
     * @param _password
     *            zugehöriges Passwort
     * @throws SQLException
     */
    Java_Database_Connector(String _user, String _password) throws SQLException {
      this.user = _user;
```

```
    this.password = _password;
    // Neue Instanz des JDBC-Treibers im Drivermanager registrieren
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    // Verbindung zum Datenbankserver herstellen
    this.openConnection();
  }

  /**
   * Prozedur openConnection öffnet eine neue Instanz des JDBC-Treibers und
   * stellt Verbindung zur Datenbank her
   *
   * @throws SQLException
   */
  public void openConnection() throws SQLException {
    // falls noch keine Verbindung existiert
    if (con == null) {
        // Neue Instanz einer Verbindung mit der Datenbank erzeugen
        con = DriverManager.getConnection(this.connectorString, this.user,
              this.password);
        // Instanz zum Abrufen der Metadaten erzeugen
        dmd = con.getMetaData();
        System.out.println("Database Connection opened.");

    } else
    // Instanz vorhanden und Verbindung geöffnet
    if (!con.isClosed()) {
        System.out.println("Database Connection already opened.");

    } // Instanz vorhanden aber Verbindung getrennt
    else {
        con = DriverManager.getConnection(this.connectorString, this.user,
              this.password);
        // Instanz zum Abrufen der Metadaten erzeugen
        dmd = con.getMetaData();
        System.out.println("Database Connection reopened.");
    }
  }
```

```
/**
 * Prozedur closeConnection beendet die Verbindung zum Datenbankserver
 *
 * @throws SQLException
 */
public void closeConnection() throws SQLException {
  // Wenn es noch keine Instanz zur Verbindung mit der Datenbank gibt.
  if (con == null) {
      System.out.println("Database Connection already closed.");
  } else
  // Wenn eine Instanz zur Verbindung mit der Datenbank besteht und diese
  // nicht geschlossen ist.
  if (!con.isClosed()) {
      this.con.close();
      System.out.println("Database Connection closed.");
  } else
      // Wenn eine Instanz zur Verbindung mit der Datenbank besteht und
      // diese bereits geschlossen ist.
      System.out.println("Database Connection already closed.");
}

/**
 * Prozedur printMetadata gibt einige Metadaten der initialisierten
 * Datenbankverbindung zurück.
 *
 * @throws SQLException
 */
public String printMetadata() throws SQLException {
  String return_value = "";

  return_value += "\nConnected with: \n";
  return_value += "DB Product:     "
        + this.dmd.getDatabaseProductVersion() + "\n";
  return_value += "DB-URL:         " + this.dmd.getURL() + "\n";
  return_value += "Username:       " + this.dmd.getUserName() + "\n";
  return_value += "DriverVersion:  " + this.dmd.getDriverVersion() + "\n";
  return_value += "DriverName:     " + this.dmd.getDriverName() + "\n";
```

```
      return return_value;
    }

    /**
     * Übergabe der Referenz auf eine Statement Instanz für SQL Kommunikation
     *
     * @return Statement - gibt Instanz eines Statements zurück
     * @throws SQLException
     */
    public Statement getStatement() throws SQLException {
      return this.con.createStatement();
    }
}
```

Diese Klasse wird in dem Package der Webservice-Klasse erstellt. Die restliche Konfiguration des JDBC-Treibers erfolgt im nächsten Kapitel in der Webservice-Klasse.

Konfigurieren des Webservices

Nun kümmern wir uns um das Herzstück des Webservices. Wir haben alle Vorbereitungen getroffen und können nun das eigentliche Programm verfassen. Dieses sieht wie folgt aus:

Anmerkung: an den markierten Stellen müssen ggf. je nach System Änderungen getätigt werden.
Bei den Streams kann ein beliebiger Speicherort verwendet werden. Dem Java_Database_Connector muss hingegen das Passwort der Datenbank übergeben werden.

```
import java.sql.SQLException;
import javax.jws.WebService;
import javax.jws.WebMethod;
import java.sql.*;
import javax.jws.WebParam;
import jaxb.*;
import java.io.FileWriter;
import java.io.FileReader;
```

```
import java.io.BufferedWriter;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;


@WebService(serviceName = "SelectLieferungVonArtikel")
public class SelectLieferungVonArtikel {

  /**
   * Lies alle Lieferdaten zur gewaehlten Artikelnummer aus
   */
  @WebMethod(operationName = "SelectLieferungVonArtikel")
  public String selectLieferungVonArtikel(@WebParam(name = "artikel") String artikel)
  {
          String ret = "";

          try
           {
                  //Streams werden verwendet, um die Ausgabe in SOAP zu ermoeglichen. Leider kann der marshaller nicht mit Strings arbeiten
                  FileWriter fw = new FileWriter("YourPath.xml");
                   BufferedWriter bw = new BufferedWriter(fw);
                  FileReader fr = new FileReader("YourPath.xml ");
                  BufferedReader br = new BufferedReader(fr);


                   // Benutzerdaten uebergeben und Instanz für Verbindung erzeugen
                  Java_Database_Connector myJDBC =
                  new Java_Database_Connector("Homeuser", "YourPW");

                   //SELECT Statement für AnzLieferungen
                     Statement count = myJDBC.getStatement();
                  String countStmt = "SELECT COUNT(*) FROM Lieferung WHERE ANr = " + artikel;

                  // ResultSet erzeugen und Query ausführen
                     ResultSet rset_c = count.executeQuery(countStmt);
```

```
int i = 0;
  //auslesen
while(rset_c.next())
            i = rset_c.getInt(1);

// Statement erzeugen für Lieferdaten
 Statement stmt = myJDBC.getStatement();
String queryStmt = "SELECT * FROM Lieferung WHERE ANr = " + artikel;

// ResultSet erzeugen und Query ausführen
 ResultSet rset = stmt.executeQuery(queryStmt);
// Ergebnis durchlaufen
  if(i != 0)
  {
          while(rset.next())
          {
                  //setzen von AnzZeilen
                 AnzZeilen az = new AnzZeilen();
                  az.setAnzZeilen(i);

                  //setzen von Zeile
                  Zeile z = new Zeile();
                  z.setAnzL(i);
                  z.setANr(rset.getInt(1));
                  z.setLNr(rset.getInt(2));
                 z.setPreis(rset.getDouble(3));

                 //Verarbeitung in XML Format durch Mashaller
                 try
                  {
                          javax.xml.bind.JAXBContext jaxbCtx = javax.xml.bind.JAXBContext.newInstance(z.getClass().getPackage().getName());
                         javax.xml.bind.Marshaller marshaller = jaxbCtx.createMarshaller();
                         marshaller.setProperty(javax.xml.bind.Marshaller.JAXB_ENCODING, "UTF-8"); //NOI18N
                         marshaller.setProperty(javax.xml.bind.Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
                          marshaller.marshal(az,fw);
                          marshaller.marshal(z, fw);
```

```
                                        }
                                        catch (javax.xml.bind.JAXBException ex)
                                        {
                                                java.util.logging.Logger.getLogger("global").log(java.util.logging.Level.SEVERE, null, ex);
                                        }
                                }
                        }
                         else //wenn i = 0 ist, ergab das COUNT Statement 0 und es gibt keine Lieferungen mit der Angegebenen Artikelnummer
                                 ret = "Keine Lieferungen mit der angegebenen Artikelnummer!";

                        //aufraeumen
                        rset_c.close();
                         rset.close();

                        //Beenden der Verbindung mit der Datenbank
                         myJDBC.closeConnection();

                          //Verarbeitung des XMLs über Streams und Einarbeitung in Strings
                        char[] buffer = new char[99];
                         int wortlaenge = br.read(buffer, 0, buffer.length);
                         while (wortlaenge != -1)
                        {
                                 //umwandeln des gelesenen Worts in einen String
                                String wort = new String(buffer,0,wortlaenge);
                                wortlaenge = br.read(buffer, 0, buffer.length);
                                ret += wort;
                          }
          }

        // Auffangen aller Ausnahmen und Behandlung (etwas kompliziert, da ueber return gearbeitet werden muss)
        catch (SQLException e)
         {
                        e.printStackTrace();
                         while (e != null)
                        {
                                ret+=("SQLState: " + e.getSQLState());
```

```
                                ret+=("Message:  " + e.getMessage());
                                ret+=("Error Code:   " + e.getErrorCode());
                                e = e.getNextException();
                                ret+=("error");
                        }
                }
                catch (FileNotFoundException e)
                {
                        System.out.println("Datei konnte nicht gefunden werden!");
                 }
                 catch (IOException e)
                {
                System.out.println("Fehler beim Einlesen der Datei!");
                }
                return  ret;
                 }
}
```

Der Code wird hier nicht komplett im einzelnen erklärt. Stattdessen wird kurz der Gesamtkontext erläutert.

Es wird zunächst eine Verbindung über den *Java Database Connector* zur Datenbank erstellt und sich entsprechend über den JDBC-Treiber eingeloggt. Danach können die entsprechenden Anfragen getätigt werden. Hier werden dabei zwei **SELECT-Statements** benutzt. Die so erhaltenen Daten sollen in das Format des WSDLs gebracht werden. Dies geschieht über die jeweiligen Klassen, die aus dem XML-Dokument durch das JAX-B Framework generiert wurden. Um dieses Format jedoch extrahieren zu können, ist ein **Marshaller** notwendig (siehe das JAX-B Tutorial). Hier kommt es zu dem Problem, dass der Marshaller eine Form von Output-Stream benötigt (es kann also nicht einfach in den return-String übertragen werden). Da uns jedoch im JAX-WS Framework keine angenehme Möglichkeit wie etwa System.out zur Verfügung steht, wird hier der Umweg über einen Writer gewählt. Die Ausgabe wird so in eine externe Datei ausgelagert, und später wieder durch einen Reader gelesen und weiterverarbeitet. Danach kann der return-String gefüllt und zurückgegeben werden.

Inbetriebnahme des Webservices

Nachdem alle Vorkehrungen getroffen wurden, kann der Webservice endlich in Betrieb genommen werden, dazu muss er als erstes *"deployed"* (gestartet) werden. Dies geschieht analog wie im Tutorial zu JAX-WS beschrieben, also durch einen Rechtsklick auf das Projekt und dort ein Linksklick auf ***"Deploy".*** Danach kann dann per Rechtsklick auf den eigentlichen Webservice ***"Test Webservice"*** angeklickt werden. Es öffnet sich ein Browserfenster mit folgendem Inhalt:

In das vorhandene Textfeld kann nun eine Artikelnummer eingetragen werden, nach der dann gesucht wird. In diesem Beispiel also die "103". Nach einem Klick auf den Button "*selectLieferungVonArtikel*" erscheinen dann die beiden Einträge SOAP Request und SOAP Response.

Diese sollten wie folgt aussehen:

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:SelectLieferungVonArtikel xmlns:ns2="http://ws/">
            <artikel>103</artikel>
        </ns2:SelectLieferungVonArtikel>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:SelectLieferungVonArtikelResponse xmlns:ns2="http://ws/">
            <return><?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AnzZeilen xmlns:ns2="http://j2ee.netbeans.org/wsdl/db_artikellieferung">
    <AnzZeilen>2</AnzZeilen>
</ns2:AnzZeilen>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:Zeile xmlns:ns2="http://j2ee.netbeans.org/wsdl/db_artikellieferung">
    <AnzL>2</AnzL>
    <ANr>103</ANr>
    <LNr>3</LNr>
    <Preis>159.9</Preis>
</ns2:Zeile>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:AnzZeilen xmlns:ns2="http://j2ee.netbeans.org/wsdl/db_artikellieferung">
    <AnzZeilen>2</AnzZeilen>
</ns2:AnzZeilen>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:Zeile xmlns:ns2="http://j2ee.netbeans.org/wsdl/db_artikellieferung">
    <AnzL>2</AnzL>
    <ANr>103</ANr>
    <LNr>4</LNr>
    <Preis>249.9</Preis>
</ns2:Zeile>
</return>
        </ns2:SelectLieferungVonArtikelResponse>
    </S:Body>
</S:Envelope>
```

Zwar entspricht die Ausgabe nicht zu einhundert Prozent der aus dem Beispiel. Dies ist jedoch den verwendeten Frameworks geschuldet. Im JAX-WS wird ein return-Statement in das SOAP eingebettet. Das return-Tag ersetzt somit das Result-Tag im Beispiel. Außerdem erfordert das JAX-B Framework, dass das AnzZeilen-Tag vor jeder Zeile ausgegeben wird, statt bloß am Anfang. Nichtsdestotrotz gibt es das angegebene Beispiel jedoch korrekt wieder.