Shopping

JDBC, Datasource, Jasper Report

Óbudai Egyetem, Java Enterprise Edition
Műszaki Informatika szak
Labor 12

Bedők Dávid
2016.03.26.
v0.2

Feladat

Készítsünk el egy **bevásárló listákat** (számlákat) nyilvántartó alkalmazást, mely a tranzakciókat listázó webes felületből, és azok részleteit tartalmazó dinamikus **PDF riportok**ból álljon.

A persistence réteg ezúttal nélkülözze mind a JPA-t, mind egyéb JDBC library-ra épülő framework használatát (pl. MyBattis), **natív JDBC lekérdezések**kel dolgozzunk, miközben természetesen a DataSource-ot a JBoss alkalmazás szerverben konfiguráljuk.

Készítsünk el egy standalone Java alkalmazást is, mely véletlenszerű tranzakciók létrehozását legyen hívatott megoldani, szintén mindenféle JDBC-re épülő framework használata nélkül.

Ismeretszerzés

A feladat során megismerkedünk a Java adatbáziskezelés “alapvető építőkövével”, a **JDBC API** használatával, mind standalone mind enterprise környezetben.

A PDF riportok előállítását pedig a **Jasper Report Library** API-ját fogjuk megismerni, és az **iReport** alkalmazás használatába is betekintünk.

Projekt

GIT \oejee\jboss\shopping

Adatbázis: database\create-schema.sql

Táblák:

- productcatalog
- product (FK: product_productcatalog_id)
- transaction
- item (FK: item_transaction_id, FK: item_product_id)

Stuktúra

shopping (root project)

- **sh-weblayer** [WAR] webmodule
- **sh-report** [JAR] ejbmodule
- **sh-loader** (standalone)

A **zöld** projektek részei az [EAR] deploymentnek.

Loader standalone alkalmazás

A “szokásos” Java plugin gradle script elemeken kívül elsősorban a megfelelő JDBC driver-t kell elhelyeznünk a classpath-on.

sh-loader/build.gradle

```
dependencies {
    compile group: 'org.postgresql', name: 'postgresql', version: '9.4.1208.
jre7'
}
```

Csatlakozni és natív lekérdezéseket végrehajtani egy RDBMS adatbázison (melyhez van JDBC driver), nagyon egyszerű. A Java SE egyértelmű API-t biztosít mindehhez.

Készítsünk el egy “wrapper” osztályt, mely kényelmesebbé teszi a csatlakozást és a lekérdezéseket (class DataService).

Csatlakozás

DataService.java

```
public class DataService implements AutoCloseable {
     private static final String HOST = "localhost";
     private static final int PORT = 5432;

     private final Connection connection;

     public DataService(String database, String user, String password) throws
ClassNotFoundException, SQLException {
          Class.forName("org.postgresql.Driver");
          this.connection = DriverManager.getConnection(this.buildJDBCUrl(database), user,
password);
          this.connection.setAutoCommit(true);
     }

     private String buildJDBCUrl(String database) {
          return "jdbc:postgresql://" + HOST + ":" + PORT + "/" + database;
     }
     @Override
     public void close() {
          try {
               this.connection.close();
          } catch (SQLException e) {
               e.printStackTrace();
          }
     }
}
```

A `Class.forName(..)` hívásra ma már nincs szükség, régebben ez inicializálta a JDBC driver-t, ma már a classloader ezt automatikusan megteszi.

A **kékkel** jelzett adatok szükségesek a csatlakozáshoz, illetve a JDBC URI előállításához.

Az `AutoCloseable` interface megvalósításával hatékonyabbá tehetjük a kódunkat, hiszen kisebb lesz az esélye annak, hogy elfelejtjük lezárni a megnyitott kapcsolatot (Java 7).

Működés

Ahhoz, hogy új tranzakciókat hozzunk létre, szükség lesz minden tranzakció esetén 1 transaction rekord, és néhány item rekord létrehozására.

A tranzakció elemei már katalógusban lévő termékekre (product) hivatkoznak, ezek előzetes betöltése szükséges, majd ezek közül a betöltő alkalmazás véletlenül fog választani, hogy épp melyik kerül fel az aktuális számlára.

A betöltött termékeket egy **ProductCatalog** példányban fogjuk tárolni, melyben egy listányi **Product** fog helyet kapni.

Bár a JDBC nem type-safe, az alkalmazásunk sem lesz az, viszont a nem type-safe viselkedést kizárólag a beolvasás során szenvedjük el.

Termék modellje

Product.java

```
public class Product {
     private long id;
     private String name;
     private ProductCategory category;
     private double price;

     public Product(long id, String name, ProductCategory category, double price) {
          this.id = id;
          this.name = name;
          this.category = category;
          this.price = price;
     }
     // getters/setters/toString()...
}
```

A **ProductCatalog** osztály egy `List<Product>`-ot kezel, melyet az `add(long id, String name, ProductCategory category, double price)` metódus segítségével tudunk bővíteni.

ProductCategory.java

```
public enum ProductCategory {
     BAKERY("bakery"),
     COLD_CUTS("cold cuts"),
     DAIRY_PRODUCTS("dairy products"),
     BAKING("baking"),
     HOUSEHOLD("household");

     private final String label;
     private ProductCategory(String label) {
          this.label = label;
     }
     public static ProductCategory fromLabel(String label) { … }
}
```

Adatbázisban a “label”-nek megfelelő értékek találhatóak meg. Ezek beolvasás során a fromLabel() metódus segítségével tudnak majd az enum egy-egy értéke lenni.

Termékek betöltése

DataService.java

```
public ProductCatalog loadProductCatalog() throws SQLException {
     final ProductCatalog catalog = new ProductCatalog();
     Statement statement = null;
     ResultSet rs = null;
     try {
          statement = this.connection.createStatement();

          final String query = "SELECT product_id, product_name, productcategory_name,
product_price FROM product INNER JOIN productcategory ON ( productcategory_id =
product_productcategory_id )";

          rs = statement.executeQuery(query);
          while (rs.next()) {
               final long id = rs.getLong("product_id");
               final String name = rs.getString("product_name");
               final String categoryName = rs.getString("productcategory_name");
               final double price = rs.getDouble("product_price");

               final ProductCategory category = ProductCategory.fromLabel(categoryName);
               catalog.add(id, name, category, price);
          }
     } catch (final SQLException e) {
          e.printStackTrace();
     } finally {
          if (rs != null) { rs.close(); }
          if (statement != null) { statement.close(); }
     }
     return catalog;
}
```

Véletlen adatok generálása

Az új tranzakció adatait pszeudo véletlen számok álal vezérelve fogjuk létrehozni.

Ennek érdekében a projekt tartalmaz egy ContentGenerator osztály, mely képes az alábbi műveletekre:

- Date generateDate() - véletlen dátum
- int generateQuantity() - véletlen mennyiség
- int generateNumberOfItems() - véletlen darabszám
- String generateSentence() - véletlen mondat

E segítségével már csupán egy **item** illetve egy **transaction** rekord beszúrását megvalósító metódust kell elkészíteni a DataService osztályba.

Tranzakció elem beszúrása

DataService.java

```
public void addItem(long transactionId, ProductCatalog catalog)  throws
SQLException {
    final String insertItem = "INSERT INTO item ( item_transaction_id,
item_product_id, item_quantity ) VALUES (?, ?, ?)" ;

    PreparedStatement statement =  this.connection.prepareStatement(insertItem);
    statement.setLong(1, transactionId);
    statement.setLong(2, catalog.getRandomProduct().getId());
    statement.setInt(3,  this.generator.generateQuantity());

    System.out.println(statement.toString());

    statement.executeUpdate();
}
```

A tranzakció létrehozása után tudjuk ezt a metódust tetszőleges számban meghívni (annak érdekében, hogy a tranzakcióhoz bekössük a tételeket).

Tranzakció beszúrása

DataService.java

```
public long addTransaction(final ProductCatalog catalog) throws SQLException {
     long transactionId = -1;
     final String insertTransaction = "INSERT INTO transaction ( transaction_date,
transaction_comment ) VALUES (?, ?)";

     PreparedStatement statement = this.connection.prepareStatement(insertTransaction,
Statement.RETURN_GENERATED_KEYS);
     statement.setDate(1, this.generator.generateDate());
     statement.setString(2, this.generator.generateSentence());

     System.out.println(statement.toString());

     statement.executeUpdate();

     try (ResultSet generatedKeys = statement.getGeneratedKeys()) {
          if (generatedKeys.next()) {
               transactionId = generatedKeys.getLong(1);
          }
     }

     final int numberOfItems = this.generator.generateNumberOfItems();

     for (int i = 0; i < numberOfItems; i++) {
          this.addItem(transactionId, catalog);
     }
     return transactionId;
}
```

A `RETURN_GENERATED_KEYS` paraméter végett lesz lehetőségünk az insert során létrejött új ID-k utólagos lekérdezésére.

Új elemek beszúrása.

Jasper Reports

http://community.jaspersoft.com/

Library (verzió: 6.2.1): http://community.jaspersoft.com/project/jasperreports-library

iReport (verzió: 5.6.0, tovább nem fejlesztik):

http://community.jaspersoft.com/project/ireport-designer

Jaspersoft Studio (verzió: 6.2.1):

http://community.jaspersoft.com/project/jaspersoft-studio

A riportokat előre egy e célra létrehozott alkalmazásban készítjük el. A riport része a natív lekérdezés, a paraméterek, és kinézet, grafikonok és egyéb allekérdezések, stb. Igen összetett riportok előállíthatóak e módon.

- Regisztrációt követően lehet letölteni
- A fenti komponensek ingyenesek
- Az iReport a régi “riport” készítő alkalmazás, mely kizárólag Java JRE 7-el működik, míg a Jaspersoft Studio az új termék (valójában egy Eclipse) erre a célra.
- Az iReport és a Jaspersoft Studio ***.jrxml** (forrás xml) állományok előállítására használható. Ennek lefordított változata a ***.jasper** állomány, melyet szintén lehet runtime használni (de runtime a *.jrxml-t le lehet fordítani *.jasper-re, így mi ezt az utat választjuk) (*Megjegyzés*: a *.jrxml verziókezelhető, lévén xml dokumentum, míg a jasper bináris).

Ha dinamikus riportokra van szükség, a http://www.dynamicreports.org/ oldalt érdemes meglátogatni, mely egy kiegészítő Library a Jasper Reports-hoz

Riport létrehozása

Főbb lépések:

- Adatbázis kapcsolat beállítása
- Riport nyelvének beállítása Java-ra (több lehetőség is van, pl. JavaScript, Groovy)
- Riport paramétereinek felvétele
- Riport lekérdezésének előállítása (a paraméterek itt használhatóak)
- Riport kinézetének, mezőinek felvitele
- Grafikonok, allekérdezések, subriportok elkészítése, stb.

Tranzakció részletek riport

```
SELECT
     product_name,
     product_price,
     productcategory_name,
     item_quantity,
     ( product_price * item_quantity)  AS total_price
FROM item
     INNER JOIN product ON
          ( product_id = item_product_id )
     INNER JOIN productcategory ON
          ( product_productcategory_id = productcategory_id )
WHERE ( 1 = 1 )
     AND ( item_transaction_id =  $P{TRANSACTION_ID} )
```

A **TRANSACTION_ID** egy `java.lang.Long` típusú paraméter. Mivel Java-ból fogjuk kezelni, Java típusok használhatóak (akár saját is).

JBoss - DataSource

standalone.xml

```
<datasource jndi-name="java:jboss/datasources/shoppingds" pool-name="
ShoppingDSPool" enabled="true" use-java-context="true">
    <connection-url>jdbc:postgresql://localhost:5432/shoppingdb</connection-
url>
    <driver>postgresql</driver>
    <security>
        <user-name>shopping_user</user-name>
        <password>123topSECret321</password>
    </security>
    <validation>
        <check-valid-connection-sql>SELECT 1</check-valid-connection-sql>
        <validate-on-match>true</validate-on-match>
        <background-validation>false</background-validation>
    </validation>
    <statement>
        <share-prepared-statements>false</share-prepared-statements>
    </statement>
</datasource>
```

EJB számára erőforrás csatolása

A DataSource is egyféle erőforrás (Resource)

sh-report/src/main/resources/META-INF/ejb-jar.xml

```
<ejb-jar ...>
    <enterprise-beans>
        <session>
            <ejb-name>ShoppingReport</ejb-name>
            <resource-ref>
                <res-ref-name>shopping-ds</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
            </resource-ref>
        </session>
    </enterprise-beans>
</ejb-jar>
```

Hivatkozunk az EJB nevére mindkét leíróban (**ShoppingReport**).

A Java EE leíróban megadjuk az erőforrás típusát.

sh-report/src/main/resources/META-INF/jboss-ejb3.xml

```
<jboss:ejb-jar ...>
    <enterprise-beans>
        <session>
            <ejb-name>ShoppingReport</ejb-name>
            <resource-ref>
                <res-ref-name>shopping-ds</res-ref-name>
                <jndi-name>java:jboss/datasources/shoppingds</jndi-name>
            </resource-ref>
        </session>
    </enterprise-beans>
</jboss:ejb-jar>
```

A JBoss specifikus leíróban hivatkozunk a JBoss-ban konfigurált DataSource JNDI nevére.

ShoppingReport SLSB

ShoppingReportImpl.java

```
package hu.qwaevisz.shopping.report.service;

import javax.sql.DataSource;

@Stateless(name = "ShoppingReport", mappedName = "ejb/shoppingReport")
public class ShoppingReportImpl  implements ShoppingReport {

    @Resource(name = "shopping-ds")
    private DataSource dataSource;

    ...
}
```

A dataSource birtokában már könnyen végrehajthatunk lekérdezéseket:

```
Connection connection =  this.dataSource.getConnection();
Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery( "SELECT transaction_id FROM
transaction");
while (rs.next()) {
    ...
}
```

Tranzakciók listája - Query

```
SELECT
    transaction_id,
    transaction_date,
    transaction_comment,
    (
        SELECT COUNT(1)
        FROM item
        WHERE item_transaction_id = transaction_id
    ) AS number_of_items,
    (
        SELECT SUM(item_quantity * product_price)
        FROM item
            INNER JOIN product ON ( product_id = item_product_id )
        WHERE item_transaction_id = transaction_id
    ) AS total_price
FROM transaction
```

Ez a lekérdezés bekerül a Java kódba.

Számított mezőként lekérjük a tranzakcióhoz tartozó elemek darabszámát és teljes árát.

Az ilyen natív query-k a forráskódban gyakorlatilag időzített bombák. Nagyon nehéz őket egy komplex alkalmazás során karbantartni (válozik a schema, változnak az elnevezési szabályok, stb.). Minden hiba csak futás időben derül ki. Ilyen alkalmazások profi acceptance tesztek nélkül nagyon veszélyesek.

Tranzakciók listája EJB service

ShoppingReportImpl.java

```
@Override
public List<Transaction> getTransactions()throws ReportException {
      Connection connection =null;
      Statement statement =null;
      ResultSet rs =null;
      try {
            final List<Transaction> transactions =new ArrayList<>();
            connection = this.dataSource.getConnection();
            statement = connection.createStatement();
            final String query = "...";
            rs = statement.executeQuery(query);
            while (rs.next()) {
                  final int id = rs.getInt("transaction_id");
                  final Date date = rs.getDate("transaction_date");
                  final String comment = rs.getString("transaction_comment");
                  final int numberOfItems = rs.getInt("number_of_items");
                  final int totalPrice = rs.getInt("total_price");

                  transactions.add(new Transaction(id, date, comment, numberOfItems, totalPrice));
            }
            return transactions;
      } catch (final SQLException e) {
            throw new ReportException(e.getLocalizedMessage());
      } finally {
            try {
                  if (rs != null) { rs.close(); }
                  if (statement != null) { statement.close(); }
                  if (connection != null) { connection.close(); }
            } catch (final SQLException e) {
                  throw new ReportException(e.getLocalizedMessage());
            }
      }
}
```

Az tranzakciókat egy e célra létrehozott `Transaction` osztályba helyezzük el, melyet az **sh-weblayer** project egyik servlet-e lekér és megjelenik. Minden sorban megjelenik a `Report?id=${transaction.id}` link, mely egy a riportot előállító servlet-re mutat.

Report Servlet
PDF Content Type

ReportServlet.java

```
@WebServlet("/Report")
public class ReportServlet extends HttpServlet {
    private static final Logger LOGGER = Logger.getLogger(ListController. class);
    @EJB
    private ShoppingReport report;
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException {
        byte[] data = null;
        final long id = Long.valueOf(request.getParameter("id"));
        try {
            data = this.report.getTransactionDetailsReport(id);
        } catch (final ReportException e) {
            LOGGER.error(e, e);
        }
        response.setHeader("Content-disposition", "inline; filename=\"transaction-
details.pdf\"");
        response.setContentType("application/pdf");
        response.getOutputStream().write(data);
    }
}
```

A PDF riport byte-tömbként jön vissza az EJB service-től. Természetesen a Jasper Reportban van mód a PDF riport állományba mentésére is a szerver oldalon, majd ezt egy linkként kitenni a weboldalra, de elegánsabb dinamikus “letöltést” e módon végrehajtani.

Jasper Reports + Gradle

sh-report/build.gradle

```
dependencies {
    compile group: 'com.lowagie', name: 'itext', version: itextVersion
    compile group: 'org.olap4j', name: 'olap4j', version: olap4jVersion
    compile group: 'net.sf.jasperreports', name: 'jasperreports', version:
jasperVersion
}
```

shopping root: build.gradle

```
allprojects {
    repositories {
        mavenCentral()
        maven { url "http://repository.pentaho.org/artifactory/repo" }
    }
}
dependencies {
    deploy project('sh-report')
    deploy project(path: 'sh-weblayer', configuration: 'archives')
    earlib group: 'com.lowagie', name: 'itext', version: itextVersion
    earlib group: 'org.olap4j', name: 'olap4j', version: olap4jVersion
    earlib group: 'net.sf.jasperreports', name: 'jasperreports', version:
jasperVersion
}
```

Az itext nem érhető el a MavenCentral repository-ban.

Ha nem készítünk Jboss module-t a Jasper számára, akkor at EAR lib könyvtárába csomagoljuk be.

Jasper ReportHelper

ReportHelper.java

```
public class ReportHelper {
```

JRXML fileból runtime JASPER állományt (JasperReport példány) állít elő. A JRXML file-okat az src/main/resources alá helyezzük el.

```
    public JasperReport compile(String jrxmlFileName)throws FileNotFoundException,JRException {
        InputStream inputStream = this.getClass().getClassLoader().getResourceAsStream
(jrxmlFileName);
        JasperDesign jasperDesign = JRXmlLoader.load(inputStream);
        return JasperCompileManager.compileReport(jasperDesign);
    }
```

A JasperReport példányt kitölti az átadott adatbázis kapcsolat és paraméterek segítségével. A riport ezzel már memóriában elkészül (JasperPrint példány).

```
    public JasperPrint fill(JasperReport jasperReport, Connection connection, Map<String,
Object> params) throws JRException {
        return JasperFillManager.fillReport(jasperReport, params, connection);
    }
```

A JasperExportManager sokféle exportálási lehetőséget kínál, sokféle formátumban. Számunkra most a PDF byte stream a legjobb választás.

```
    public byte[] exportToPdf(JasperPrint jasperPrint) throws JRException {
        return JasperExportManager.exportReportToPdf(jasperPrint);
    }
}
```

Transaction Details EJB service

ShoppingReportImpl.java

```
@Override
public byte[] getTransactionDetailsReport( long transactionId)  throws
ReportException {
    Transaction transaction = this.getTransaction(transactionId);
    try {
        ReportHelper reportHelper = new ReportHelper();

        Connection connection =  this.dataSource.getConnection();

        JasperReport report = reportHelper.compile( "reports
jrxml");
        Map<String, Object> params =  new HashMap<>();
        params.put("TRANSACTION_ID", transaction.getId());
        params.put("TRANSACTION_DATE", transaction.getDate());
        params.put("TRANSACTION_COMMENT", transaction.getComment());
        JasperPrint print = reportHelper.fill(report, connection, params);

        return reportHelper.exportToPdf(print);
    } catch (SQLException | FileNotFoundException | JRException e) {
        LOGGER.error(e, e);
        throw new ReportException(e.getLocalizedMessage());
    }
}
```

A paraméterek közül a **TRANSACTION_ID** a lekérdezésben is szerepel.