JDBC

Introducere

In acest laborator vom vorbi despre JDBC.

In primul rand, ce este JDBC? JDBC este o interfata (API) prin care o aplicatie (Java) poate comunica cu o alta aplicatie (serverul de baza de date) in functie de un protocol prestabilit. Nu uitati, ca doua sisteme sa comunice, este necesar crearea unui socket pe client, un socket corespunzator pe server si apoi datele sunt trimise prin streamul de out, respectiv primite prin in. Modul in care datele sunt serializate si in sine datele care trebuie trimise depind foarte mult de la sistemul de baze de date cu care vorbim. Pentru a ne facilita comunicarea astfel incat sa nu trebuiasca noi sa ne punem problema ce si cum serializam exista deja niste biblioteci specifice fiecarei baze de date numite “drivere”.

Lucrul cu o baza de date in java porneste de la un obiect de tipul java.sql.Connection. Acest obiect reprezinta atat un punct de comunicare cu baza de date cat si un container pentru sesiunea noastra cu baza de date (numit tranzactie) in cazul in care baza de date o suporta. Lucrul cu tranzactii nu intra in scopul acestui laborator. Daca acesta este punctul de intrare atunci primul nostru scop ar fi sa obtinem un astfel de obiect. In primul rand trebuie sa descarcam de pe internet un driver potrivit bazei de date cu care vom lucra (mysql). Pentru asta am atasat un fisier acestui laborator (vedeti link mai jos). Aceasta biblioteca trebuie apoi adaugata ca dependinta in proiectul nostru (asa cum ati facut cu jersey). Din acest punt, procedura difera considerabil intre aplicatiile web si cele non-web. Voi vorbi mai intai despre aplicatiile non-web.

Obtinerea unui Connection

Pentru varianta non-web, pornirea este foarte simpla. Trebuie sa apelam niste metode de la niste obiecte cu anumiti parametrii. Subliniez faptul ca acest laborator este gandit pentru baze de date MySQL, lucrurile pot sa difere pentru alte drivere. Pentru cazuri concrete, cititi informatiile de pe siteul de unde ati downloadat driverul pe care vreti sa il folositi. Eu personal am lucrat cu MySQL, Oractle, HSQL, MSSQL si MongoDB si va spun ca lucrurile difera destul de mult intre drivere.

```
final String DB_URL = "jdbc:mysql://[host]:3306/[dbName]";
        final String DB_USER = "username";
        final String DB_PASS = "password";

        Class.forName("com.mysql.jdbc.Driver");
```

java.sql.Connection myConn = java.sql.DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);

Am lasat tipurile vizibile in cod, dar evident puteti sa folositi import si versiunea scurta a lor. Odata ce obtinem aceasta conexiune, lucrurile devin comune intre web si non web, deci acum voi vorbi despre web.

Pentru web, este un pic mai complicat de configurat sistemul pentru a folosi baze de date. In primul rand, vom depinde de webserver in administrarea conexiunilor ceea ce inseamna ca trebuie sa il configuram, iar procedura nu este la fel pentru toate serverelor (desi idea de baza este identica). Exemplele de aici functioneaza cu Tomcat. Pentru alte servere, retineti ideea de baza si cautati cum sa configurati aplicatia/serverul pentru a folosi JDBC. Ideea de baza este ca vom dori sa ni se “injecteze” o resursa in obiectele cu care lucram, de cate webserver sau (daca din diverse motive nu dorim sa injectam resursa) sa o extragem din “Context”. Pe scurt, asa cum stiti ca serverul ofera sesiunea, requestul si multe alte obiecte exista si niste obiecte create pe server si accesibile noua prin JNDI (https://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface) . Pe scurt JNDI ne ofera posibilitatea sa asignam niste nume unor obiecte si apoi sa putem sa solicitam unui container sa ne dea acele obiecte. Tomcat va crea (ii vom spune noi cu ce parametrii) niste obiecte de tipul javax.sql.DataSource care sunt punctul de intrare in connection poolul administrat de Tomcat. Desi am tot spus Tomcat, ideea este comuna tuturor serverelor, modul in care configuram aceste resurse insa difera de la server la server. Pentru Tomcat, exista (sau facem noi) un fisier META-INF/context.xml. Initial acest fisier ar trebui sa contina doar:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/myApp" />
```

Noi vom expanda un pic tagul context pentru a include si alte informatii:

```
<Context path="/myApp">
        <Resource name="jdbc/myDb"
        auth="Container"
        type="javax.sql.DataSource"
        username="db_username"
        password="db_password"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://[host]:3306/myDb"
        maxActive="15"
        maxIdle="3"/>
</Context>
```

Aceste elemente il vor instrui pe Tomcat sa creeze un “connection pool” din care ne putem servi in aplicatie. Obiectul DataSource ne permite sa accesam acest connection pool.

Sa spunem ca suntem intr-un servlet si stim ca o sa avem nevoie de acces la baza de date. Tot ce trebuie sa facem in cod este sa “injectam” o resursa folosind numele ei JNDI.

```
@Resource(name = "jdbc/myDb")
   private DataSource dbRes;
```

Numele din adnotarea Resource trebuie sa se potriveasca cu numele din Resource din context.xml.

Dupa ce am injectat acest DataSource, pentru o obtine o conexiune trebuie sa facem doar:

```
java.sql.Connection myConn = dbRes.getConnection();
```

Folosirea conexiunii

In acest moment ne aflam in acelasi punct in care ne aflam in varianta non-web.

Va atrag atentia ca in acest laborator vom lucra cu niste resurse care trebuie inchise. java.sql.Connection este una dintre ele. Este o greseala FOARTE grava sa nu inchidem un connection dupa ce l-am folosit (Mare atentie la test si la proiect! Va voi taxa masiv daca nu le inchideti corespunzator!!!).

Desi pare mai simplu in varianta non-web (si da, “merge”, si in web sa facem la fel) exista niste diferente care ne vor strica ziua. In primul rand, observati ca la web se creeaza un connection pool. Asta inseamna ca Tomcat se conecteaza la baza de date de mai multe ori (vedeti configuratia) iar cand dorim o conexiune ne ofera una din cele multe pe care le-a facut. Tot asa, cand inchidem o conexiune, Tomcat nu o inchide efectiv, ci doar o “elibereaza” pentru a putea fi oferita altui proces (in web sunt multe procese….). In non-web, conexiunea chiar este stabilita cand este creat obiectul Connection. Se trimit username, password si multe altele, iar la close chiar se inchide si socketul este inchis si el. Dupa cum va puteti imagina, este foarte costisitor sa deschidem o conexiune la baza de date (timp, date, procesor etc) deci varianta Tomcat este o optimizare pe care….va trebui pana la urma sa o faceti voi de mana in aplicatia voastra (Sa va definiti un obiect care reprezinta connection poolul, sa definiti o interfata care reprezinta conexiunea etc). Deci desi la varianta non-web “pornim repede” la varianta web avem multe lucruri implementate de niste baieti destepti care ne ajuta mult.

Va rog sa observati ca tipul Connection este o interfata! Tocmai din acest motiv, in non-web implementarea efectiva, obiectul, poate sa fie conexiunea efectiva care se inchide complet iar in web este doar un instrument pentru a folosi conexiunea efectiva, iar close inseamna eliberare.

Alte containere au alte moduri de configurare a connection poolului, unele dintre ele permit chiar configurarea vizuala (dintr-un UI). De fapt, singurul container cu care am lucrat si nu are (sau nu stiu eu sa aiba…) o interfata web pentru astfel de lucruri este Tomcat.

Odata ce am obtinut conexiunea, trebuie sa trimitem interogari la baza de date. Interogarile se numesc Statement in engleza, de aici si numele obiectelor.

Pentru a obtine un obiect de tip Statement vom scrie:

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM student")
```

Unde conn este obiectul conexiune. Observati ca am folosit un PreparedStatement. PreparedStatement ne ofera cateva avantaje fata de un Statement normal, doua dintre ele fiind faptul ca putem folosi parametrii in query si suntem protejati la SQL Injection.

Din acest moment, depinde ce query este cel pe care l-am trimis bazei de date. Daca folosim un select (ca mai sus) scriem:

```
ResultSet rs = ps.executeQuery();
```

Unde ResultSet este obiectul care contine raspunsul. Va atrag atentia ca ResultSet si Statement sunt niste resurse care TREBUIE inchise!!!!!! Nu uitati try/catch/finally! (sau try with resources)

Pentru Update, Insert, Delete folosim:

```
int affectedRows = ps.executeUpdate();
```

Exista si o varianta care poate lucra cu orice query, inclusiv Alter table, Drop etc:

```
boolean resultType = ps.execute();
```

Result type este true daca queryul a produs un ResultSet sau false altfel. Personal, nu am folosit niciodata aceasta forma.

Pentru a obtine idul autogenerat al ultimei intrari folosim:

```
ResultSet rs = ps.getGeneratedKeys();
if(rs.next())
{
   int last_inserted_id = rs.getInt(1);
}
```

ATENTIE! Aceasta procedura este posibila (in principiu) doar pentru MySQL!

Pentru a citit apoi datele dintr-un ResultSet o sa folosim un soi de iterare peste result set, care se face in felul urmator:

```
ResultSet rs = ps.executeQuery();
while (rs.next())
            {
                System.out.println(rs.getString("nume"));
            }
```

Puteti folosi numele coloanei sau idul coloanei (numerotarea incepe de la 1!).

In cazul in care queryul nostru foloseste parametrii vom scrie:

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM student
WHERE nume = ? AND prezente = ?");
```

Observati semnele de intrebare. Ele marcheaza locul unde serverul de baze de date stie ca trebuie sa adauge valorile corespunzatoare.

Pentru a seta acei parametri folosim:

```
ps.setString(1, "Ion");
ps.setInt(2,5);
```

Observati faptul ca numarul de ordine al parametrilor porneste de la 1 nu de la 0!

MySQL driver:
https://drive.google.com/open?id=0B5ar2tHw-X9vUDNrc05mR1AycWs

Exercitii

Pentru exercitii, am pregatit o baza de date MySQL accesibila la IP:
82.76.115.105
port
3306
baza de date:
pao
Structura bazei de date este cam asa:
Tabela “student”

#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	**id_student**	**INT**	**11**	☐	☐	☐	**AUTO_INCREMENT**
2	nume	VARCHAR	200	☐	☐	☐	No default
3	email	VARCHAR	200	☐	☑	☐	NULL
4	cod_verificare	VARCHAR	100	☐	☑	☐	NULL
5	validat	INT	11	☐	☑	☐	NULL
6	nota_test	INT	11	☐	☑	☐	NULL
7	nota_proiect	INT	11	☐	☑	☐	NULL
8	prezente	INT	11	☐	☐	☐	No default

Tabela “proiecte”

#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
1	**id_proiect**	**INT**	**11**	☐	☐	☐	**AUTO_INCREMENT**
2	nume_proiect	VARCHAR	100	☐	☐	☐	No default
3	descriere	TEXT		☐	☐	☐	No default
4	specificatii_1p	TEXT		☐	☐	☐	No default
5	specificatii_2p	TEXT		☐	☐	☐	No default
6	specificatii_3p	TEXT		☐	☐	☐	No default

Tabela “proiectStudent”

#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill
1	**student_id**	**INT**	**11**	☐	☐	☐
2	**proiect_id**	**INT**	**11**	☐	☐	☐

Exercitiu 1

Faceti o aplicatie care sa se conecteze la baza de date si sa afiseze o lista cu studentii.

Exercitiu 2

Faceti o aplicatie in care sa apara un camp text unde este poate fi scris numele unui student, un buton si un text. Cand se apara pe buton, se cauta studentul cu numele respectiv in baza de date. Daca nu exista se afiseaza mesaj de eroare. Daca exista mai multi se afiseaza “Prea multe potriviri”. Daca exista fix un student, i se afiseaza numele complet si numarul de prezente.

Exercitiu 3

Faceti o aplicatie in care sa avem doua dropdown (JComboBox) un buton si un text.
In primul dropdown sa avem listati toti studentii.
In al doilea dropdown sa avem listate toate proiectele.
De fiecare data cand elementul curent din lista student e modificat, trebuie sa modificati textul astfel incat sa apara proiectul pentru respectivul student.
Cand se apasa pe buton (Save), studentului selectat i se atribuie proiectul selectat si apare textul “Salvat” in loc de vechiul proiect.