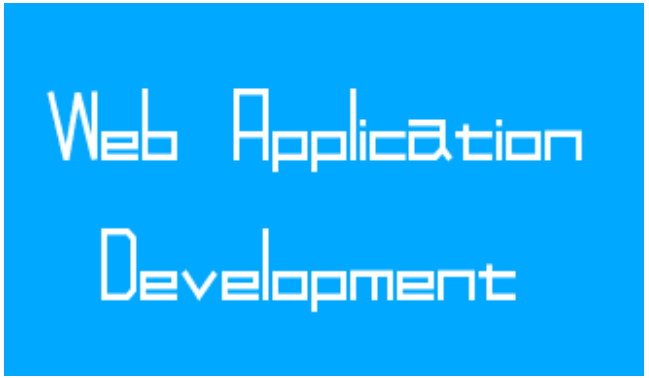
互联网应用开发技术

Web Application Development

第8课
WEB后端-访问关系型数据库

Episode Eight

Access to RDBMS With JDBC



陈昊鹏

chen-hp@sjtu.edu.cn

Overview

- Access Database via JDBC Reading
  - JDBC Specification
  - DriverManager vs. DataSource
  - Statements
  - RowSet vs. ResultSet
- Pros and Cons of
  - JDBC Reading and ORM

JDBC Specification

- The JDBC™ API provides
  - programmatic access to relational data from the Java™ programming language.
- The JDBC API is part of the Java platform
  - which includes the Java™ Standard Edition (Java SE™) and the Java™ Enterprise Edition (Java EE™).
- The JDBC 3.0 API is divided into two packages:
  - `java.sql` and `javax.sql`.
  - Both packages are included in the J2SE and J2EE platforms.

JDBC Overview

- Establishing Connection
  - The JDBC API defines the `Connection` interface to represent a connection to an underlying data source.
    - `DriverManager` or `Datasource`

- Executing SQL Statements and Manipulating Results
  - `DatabaseMetadata`
  - `Statement`, `PreparedStatement`, and `CallableStatement`.
  - `ResultSet` and `RowSet`

Establishing Connection

- To obtain a connection, the application may interact with either:
  - the `DriverManager` class working with one or more Driver implementations

    OR
  - a `DataSource` implementation

Establishing Connection

- DriverManager
  - registerDriver and getConnection

```
package sample;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class AccessDB {

      public static void main(String[] args) throws
                                        ClassNotFoundException, SQLException
      {
                Class.forName("com.mysql.jdbc.Driver");

                String url = "jdbc:mysql://localhost:3306";
                String user = "root";
                String passwd = "12345678";

                Connection con = DriverManager.getConnection(url, user, passwd);
                System.out.println(con.getTransactionIsolation());
      }
}
```

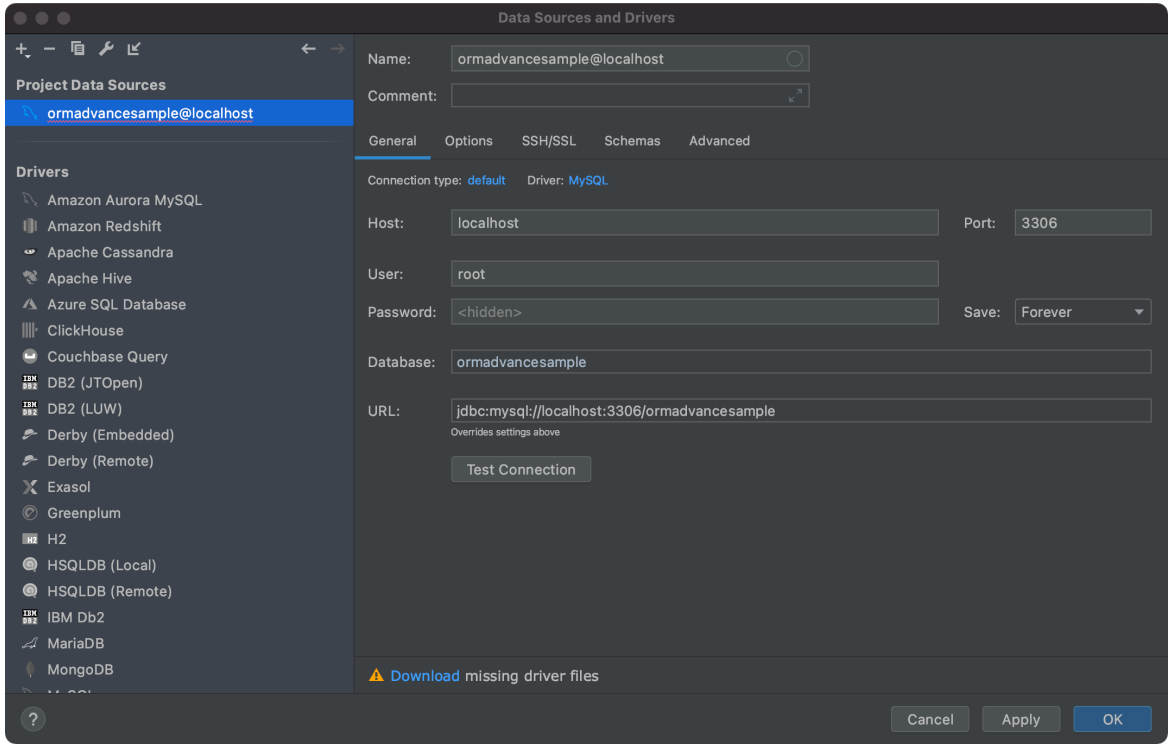
查找driver

声明数据库位置信息以及用户名密码

若使用spring则使用配置文件代替

Establishing Connection

- Necessary Configuration



Establishing Connection

- `DataSource`
  - A logical name is mapped to a `DataSource` object via a naming service that uses the Java Naming and Directory Interface ™ (JNDI).

Property Name	Type	Description
databaseName	String	name of a particular database on a server
dataSourceName	String	a data source name
description	String	description of this data source
networkProtocol	String	network protocol used to communicate with the server
password	String	a database password
portNumber	int	port number where a server is listening for requests
roleName	String	the initial SQL rolename
serverName	String	database server name
user	String	user's account name

Establishing Connection

- DataSource
  - A logical name is mapped to a DataSource object via a naming service that uses the Java Naming and Directory Interface ™ (JNDI).

```
package sample;
import java.sql.SQLException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;
public class Server {
      public static void main(String[] args) throws ClassNotFoundException,
                                                     SQLException, NamingException
      {
                MysqlDataSource ds = new MysqlDataSource();
                ds.setServerName("localhost");
                ds.setPortNumber(3306);
                ds.setUser("root");
                ds.setPassword("12345678");

                Context namingContext = new InitialContext();
                namingContext.bind("rmi://localhost:1099/datasource", ds);
      }
}
```

Establishing Connection

- DataSource
  - A logical name is mapped to a DataSource object via a naming service that uses the Java Naming and Directory Interface ™ (JNDI).

```
package sample;
import java.rmi.RemoteException;
import java.sql.Connection;
import java.sql.SQLException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;

public class Client {
   public static void main(String[] args) throws
              NamingException, RemoteException, SQLException
   {
      Context namingContext = new InitialContext();
      String url = "rmi://localhost:1099/datasource";
      MysqlDataSource ds = (MysqlDataSource) namingContext.lookup(url);
      Connection con = ds.getConnection("root","12345678");
      System.out.println(con.getTransactionIsolation());
   }
}
```

Establishing Connection

- Necessary Configuration
- Run the rmiregistry
  - `rmiregistry`
- Run the `Server`
- Run the `Client`

Statements

- Statement
  - defines methods for executing SQL statements that do not contain parameter markers
- PreparedStatement
  - adds methods for setting input parameters
- CallableStatement
  - adds methods for retrieving output parameter values returned from stored procedures

Statement


```
// get a connection from the DataSource object ds
Connection con = ds.getConnection(user, passwd);

// create two instances of Statement
Statement stmt1 = con.createStatement();
Statement stmt2 = con.createStatement();

// Setting ResultSet Characteristics
Statement stmt = con.createStatement(
          ResultSet.TYPE_SCROLL_INSENSITIVE,
          ResultSet.CONCUR_UPDATABLE,
          ResultSet.HOLD_CURSORS_OVER_COMMIT);
```

Statement


```
// Executing Statement and  return ResultSet
ResultSet rs = stmt.executeQuery(
    "select id, username, password, email from tbl_user");
while (rs.next()){
    ...
}

// Returning an Update Count
Statement stmt = con.createStatement();
int rows = stmt.executeUpdate(" update tbl_user
    set username = 'ADMIN' " + "where id = 1");
if (rows > 0) {
    ...
}
```

读写数据库得到的结果一般都是一个集合。

向这里这样直接拼接字符串生成sql语句是有危险的，因为如果添加的字符串中还具有引号，可能会改变语义。

Statement


```
// Using execute method
String sql;
...
Statement stmt = conn.createStatement();
boolean b = stmt.execute(sql);
if (b == true) {
     // b is true if a ResultSet is returned
     ResultSet rs;
     rs = stmt.getResultSet();
     while (rs.next()) {
               ...
     }
     rs.close();
}
else {
     // b is false if a UpdateCount is returned
     int rows = stmt.getUpdateCount();
      if (rows > 0) {
               ...
     }
}
stmt.close();
conn.close();
```

sql语句若能正确返回，则返回布尔值TRUE。

PreparedStatement

preparedstatement一般用于需要额外传递参数的sql语句的执行。

```
// Creating a PreparedStatement
PreparedStatement ps = con.prepareStatement("INSERT
INTO
    tbl_user (id, username, password, email)
    VALUES (?, ?, ?, ?)");
// Setting Parameters
ps.setInt(1, 3);
ps.setString(2, "Guest");
ps.setString(3, "guest");
ps.setString(4, "haha@163.com");
ps.execute();
```

这里的问号代表的就是参数具体取值，与set方法配套，每个？有一个索引值，从1开始；也可以使用拼接字符串，但如前所述，是不安全的。

setXXX,XXX的具体内容取决于该列的属性值的类型。

PreparedStatement

```
// ParameterMetaData
PreparedStatement pstmt = con.prepareStatement(
    "SELECT * FROM tbl_user WHERE id = ?
                                and username = ?");
pstmt.setInt(1, 3);
pstmt.setString(2, "Guest");
pstmt.execute();
ParameterMetaData pmd = pstmt.getParameterMetaData();
int number = pmd.getParameterCount();
```

metadata，元数据，主要是获取sql语句使用的具体参数信息。

PreparedStatement


```
// ResultSetMetaData
ResultSetMetaData rsmd = pstmt.getMetaData();
int colCount = rsmd.getColumnCount();
int colType;
String colLabel;
for (int i = 1; i <= colCount; i++) {
    colType = rsmd.getColumnType(i);
    colLabel = rsmd.getColumnLabel(i);
    System.out.println(colType + ":" + colLabel);
}
```

CallableStatement

(用于程序调用)

```
CREATE DEFINER=`root`@`localhost`
    PROCEDURE `insert_user`(
            in id int,
            inout username varchar(20),
            in password varchar(20),
            in email varchar(20))
BEGIN
    insert tbl_user (id,username,password,email)
            Values(id,username,password,email);
    select username from tbl_user where id = id;
END
```

CallableStatement


```
CallableStatement cstmt =
   con.prepareCall("{CALL insert_user(?, ?, ?, ?)}");
cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
cstmt.setInt(1, 4);
cstmt.setString(2, "Host");
cstmt.setString(3, "Host");
cstmt.setString(4, "Host@sjtu.edu.cn");

cstmt.execute();

// Retrieve OUT parameters
String username = cstmt.getNString(2);
System.out.println(username);
```

Manipulate result

- `ResultSet`
  - Types
    1. TYPE_FORWARD_ONLY
    2. TYPE_SCROLL_INSENSITIVE
    3. TYPE_SCROLL_SENSITIVE
  - Concurrency
    1. CONCUR_READ_ONLY
    2. CONCUR_UPDATABLE
  - Holdability
    1. HOLD_CURSORS_OVER_COMMIT
    2. CLOSE_CURSORS_AT_COMMIT

Manipulate result

- ResultSet

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(“select author, title, isbn"
                                 +"from booklist”);
```

next方法是一定有的，其他的不一定，依数据库而定

```
next()              beforeFirst()
previous()          afterLast()
first()             relative(int rows)
last()              absolute(int row)

int colIdx = rs.findColumn(“ISBN”);

ResultSetMetaData rsmd = rs.getMetaData();
int colType [] = new int[rsmd.getColumnCount()];
for (int idx = 0, int col = 1; idx < colType.length;
     idx++, col++)
    colType[idx] = rsmd.getColumnType(col);
```

Manipulate result

- ResultSet

```
// Update a row: two-phase process
Statement stmt =
conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                     ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery(“select author from
                     booklist " + "where isbn = 140185852”);
rs.next(); 获取要更新的行
rs.updateString(“author”, “Zamyatin, Evgenii Ivanovich”);
rs.updateRow();  updaterow相当于transaction中的commit、repository中的save、jdbctemplate中的update等一样，相当于对结果的保存，并上传到数据库

// Delete a row
rs.absolute(4);
rs.deleteRow();
```

Manipulate result

- ResultSet

```
// Insert a row: three steps
// select all the columns from the table booklist
ResultSet rs = stmt.executeQuery("select author, title,
                                  isbn " + "from booklist");
rs.moveToInsertRow();
// set values for each column
rs.updateString(1, "Huxley, Aldous");
rs.updateString(2, "Doors of Perception and Heaven and
                    Hell");
rs.updateLong(3, 60900075);
// insert the row
rs.insertRow();
// move the cursor back to its position in the result set
rs.moveToCurrentRow();
```

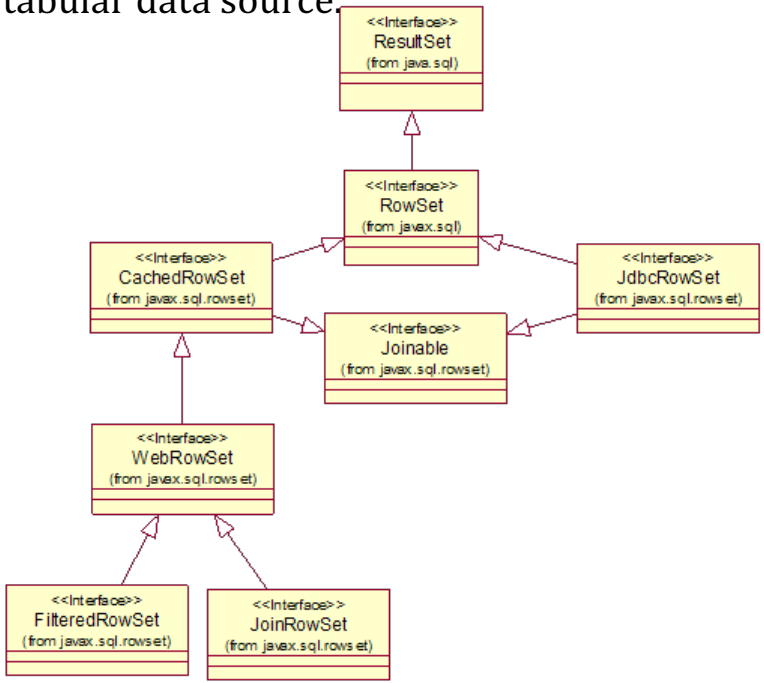
移动到要插入的行，默认是最后一行的下一行的位置

提交

光标回移

Manipulate result

- RowSet
  - A `javax.sql.RowSet` object encapsulates a set of rows that have been retrieved from a tabular data source.
  - JdbcRowSet - online
  - CachedRowSet
    - WebRowSet
    - FilteredRowSet
    - JoinRowSet



Manipulate result

- RowSet
  - JdbcRowSet – online

```
public class AccessDB {
    private JdbcRowSet rowset;
    public AccessDB(String url, String user, String pwd)
                            throws SQLException {
            RowSetFactory rowSetFactory =
                    RowSetProvider.newFactory();
            JdbcRowSet rowset =
                    rowSetFactory.createJdbcRowSet();
            rowset.setUrl(
                    "jdbc:mysql://localhost:3306/sample_one");
            rowset.setUsername("root");
            rowset.setPassword("12345678");
            rowset.setCommand("SELECT * FROM tbl_user");
            rowset.execute();
    }
```

Manipulate result

- RowSet
  - JdbcRowSet – online

```
public List<User> get() throws SQLException {
    List<User> records = new ArrayList<>();
    rowset.beforeFirst();
    while (rowset.next()) {
        User record = new User();
        record.setId(rowset.getLong(1));
        record.setUsername(rowset.getString(2));
        record.setPassword(rowset.getString(3));
        record.setEmail(rowset.getString(4));
        records.add(record);
    }
    return records;
}
```

rowset实际上就是对应于数据库中的一行，相当于一个实体类。

Manipulate result

- RowSet
  - JdbcRowSet – online

```
public void add(User user) throws SQLException {
    rowset.moveToInsertRow();
    rowset.updateInt(1, (int)user.getId());
    rowset.updateString(2, user.getUsername());
    rowset.updateString(3, user.getPassword());
    rowset.updateString(4, user.getEmail());
    rowset.insertRow();
}
```

Manipulate result

- RowSet
  - JdbcRowSet – online

```
public class User {

    private long id;
    private String username;
    private String password;
    private String email;

    public long getId(){return this.id;}
    public String getUsername(){return this.username;}
    public String getPassword(){return this.password;}
    public String getEmail(){return this.email;}

    public void setId(long id) {this.id = id;}
    public void setUsername(String username) {this.username = username;}
    public void setPassword(String password) {this.password = password;}
    public void setEmail(String email) {this.email = email;}
}
```

Manipulate result

- RowSet
  - JdbcRowSet – online

```
public static void main(String[] args) throws SQLException {
        AccessDB ad = new AccessDB(url,user,pwd);

        List<User> records = ad.get();
        Iterator<User> it = records.iterator();
        while(it.hasNext()){
                User use = it.next();
                ……
        }

        User usertoadd = new User();
        usertoadd.setId(10);
        ……
        ad.add(usertoadd);

        records = ad.get();
        it = records.iterator();
        while(it.hasNext()){
                User use = it.next();
                ……
        }
}
```

Manipulate result

- RowSet
  - CachedRowSet

```
RowSetFactory rowSetFactory =
        RowSetProvider.newFactory();
rowset = rowSetFactory.createCachedRowSet();
```

  - WebRowSet

```
RowSetFactory rowSetFactory =
        RowSetProvider.newFactory();
rowset = rowSetFactory.createWebRowSet();
rowset.execute();
rowset.writeXml(System.out);
```

Manipulate result

- RowSet
  - FilteredRowSet

```
RowSetFactory rowSetFactory =
                RowSetProvider.newFactory();
rowset =  rowSetFactory.createFilteredRowSet();
Range range = new Range();
rowset.setFilter(range);
```

Manipulate result

- RowSet
  - FilteredRowSet

```
class Range implements Predicate {

    public boolean evaluate(RowSet rs) {
        try {
            if (rs.getInt(1) > 1) {
                return true;
            }
        } catch (SQLException e) {
            // do nothing
        }
        return false;
    }

    public boolean evaluate(Object value, int column) throws SQLException {
        return false;
    }

    public boolean evaluate(Object value, String columnName)
            throws SQLException {
        return false;
    }
}
```

Manipulate result

- RowSet
  - Rowsets can generate three different types of events:

  1. Cursor movement events
  2. Row change events
  3. Rowset change events

  - To add a listener to Rowset

```
Listener listener = new Listener();
rowset.addRowSetListener(listener);
```

Manipulate result

- RowSet
  - RowSetListener: 与其余的listener类似，主要是监听sql语句执行后产生的新的事件，如更新数据等。

```
public class Listener implements RowSetListener {

    @Override
    public void cursorMoved(RowSetEvent arg0) {
        System.out.println("The cursor is moved");
    }

    @Override
    public void rowChanged(RowSetEvent arg0) {
        System.out.println("A row is changed");
    }

    @Override
    public void rowSetChanged(RowSetEvent arg0) {
        System.out.println("The rowset is changed");
    }

}
```

Pros & Cons of JDBC reading

- JDBC reading
  - Advantages:
    - Good performance, especially for accessing massive data
    - Take advantage of various functions provided by DBMS
    - Use stored procedures to implement complex logics
  - Disadvantages:
    - Coupling with DBMS
    - Coupling with data structure
    - Programming is complicate
  - How to avoid the disadvantages?

References

- JDBC™ 3.0 Specification Final Release,
  - Jon Ellis & Linda Ho with Maydene Fisher
- JNDI ™ 1.2.1 Javadoc,
  - http://java.sun.com/products/jndi/1.2/javadoc/

- *Web开发技术*
- *Web Application Development*

Thank You!