Java Backend Development Live-85

lecture-11

Agenda

- AOP (Aspect Oriented Programming)
- RDBMS(MySQL)
- JDBC
  - Statement
  - PreparedStatement
  - Transactions
- Problems of JDBC API

AOP (Aspect Oriented Programming)

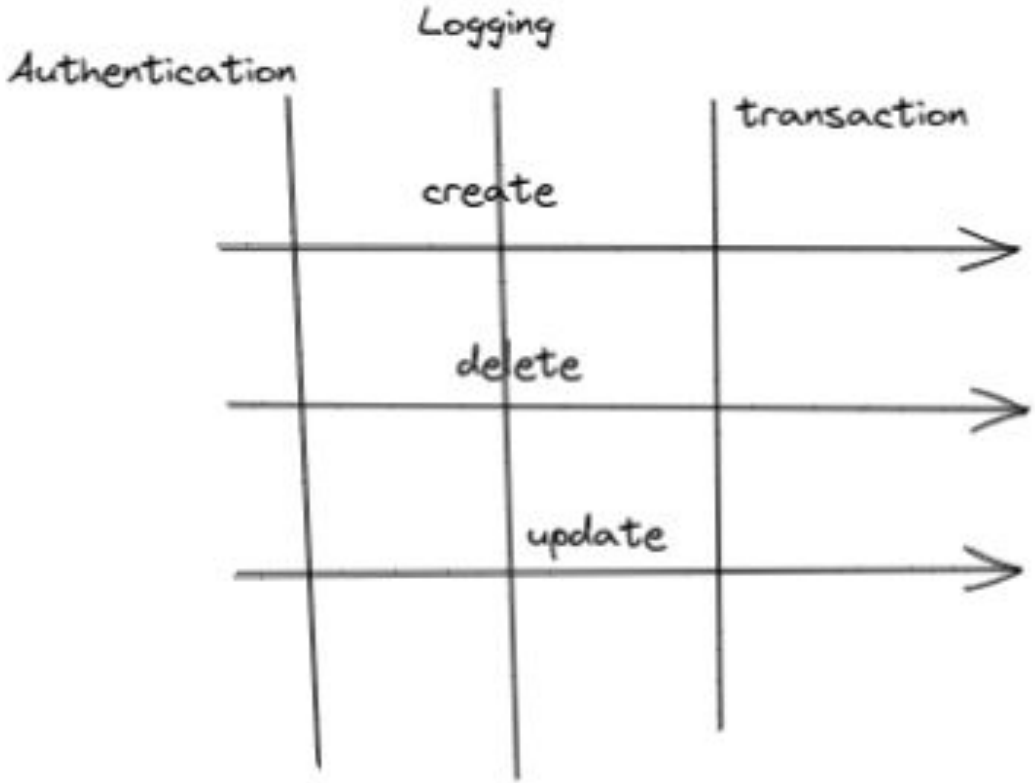
AOP is a programming paradigm that aims to increase modularity by allowing the separation of **cross-cutting concerns**. It does this by adding additional behavior to existing code **without modifying the code itself**.

Joinpoint: A Joinpoint is a point during the **execution of a program**, such as the execution of a method or the handling of an exception.

Advice: This is the **actual action** to be taken either before or after the method execution. This is the actual piece of code that is invoked during program execution by Spring AOP framework.

PointCut: This is a set of one or more join-points where an advice should be executed.

cross-cutting concerns



RDBMS(MySQL)

We use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using **primary keys** or other keys known as **Foreign Keys**.

Structured Query Language (SQL)

DDL (Data Definition Language): It allows you to perform various operations on the database such as CREATE, ALTER and DELETE objects.

DML (Data Manipulation Language): It allows you to access and manipulate data. It helps you to insert, update, delete and retrieve data from the database.

DCL (Data Control Language): It allows you to control access to the database. Example – Grant or Revoke access permissions.

TCL (Transaction Control Language): It allows you to deal with the transaction of the database. Example – Commit, Rollback, Savepoint, Set Transaction.

Important Commands

CREATE DATABASE - creates a new database

CREATE TABLE - creates a new table

ALTER TABLE - modifies a table

SELECT - extracts data from a table

UPDATE - updates data in a table

DELETE - deletes data from a table

INSERT INTO - inserts new data into a table

DROP TABLE - deletes a table

JDBC(Java Database Connectivity)

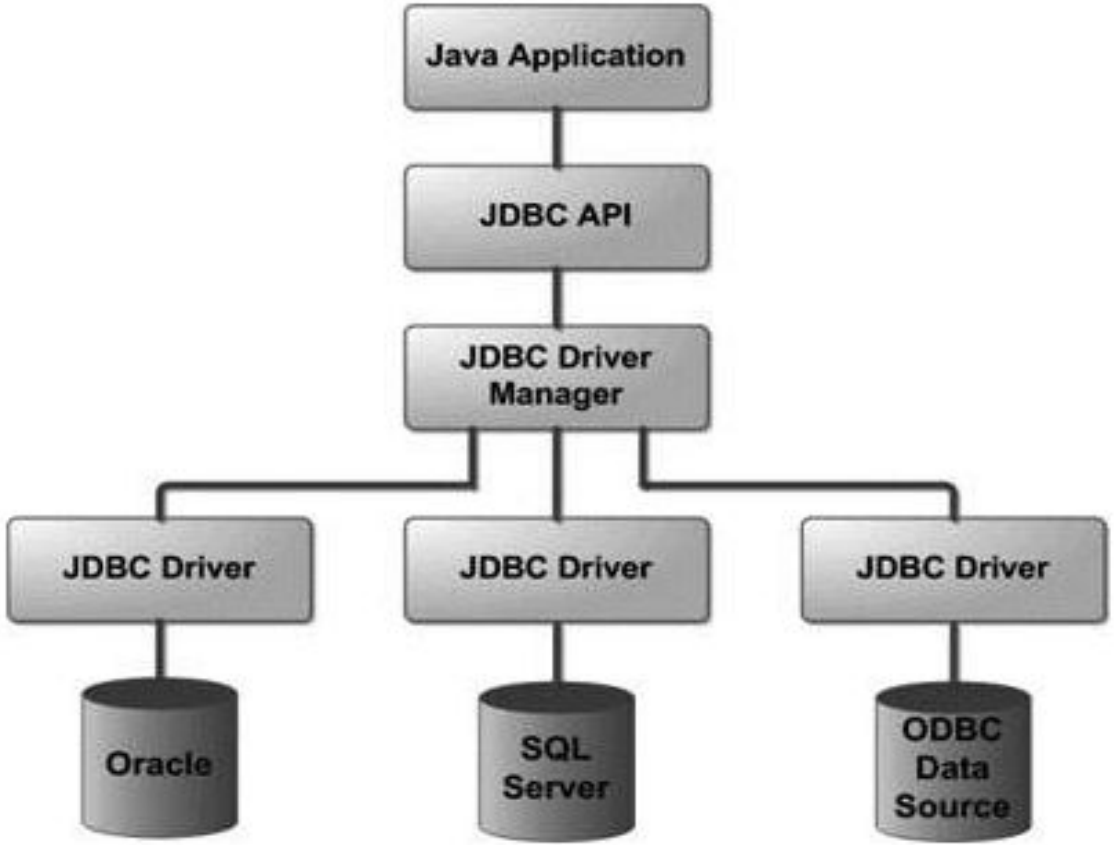
JDBC stands for Java Database Connectivity, which is a standard Java API for **database-independent connectivity** between the Java programming language and a wide range of databases

Driver: Driver is specific to DB vendors mysql, oracle, postgres etc.

DriverManager: The JDBC driver manager ensures that the correct driver is used to access each data source.

```
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDb", "user1", "pass")
Connection con = DriverManager.getConnection("jdbc:postgresql://localhost/myDb", "user1", "pass")
```

JDBC (Java Database Connectivity)



Statement

Statement stmt = con.createStatement()

Executing SQL instructions can be done through the use of three methods:

- executeQuery() for SELECT instructions
- executeUpdate() for updating the data or the database structure
- execute() can be used for both cases above when the result is unknown

PreparedStatement

PreparedStatement objects contain precompiled SQL sequences. They can have one or more parameters denoted by a question mark.

```
String updatePositionSql = "UPDATE employees SET position=? WHERE emp_id=?";

try (PreparedStatement pstmt = con.prepareStatement(updatePositionSql)) {

pstmt.setString(1, "lead developer");

pstmt.setInt(2, 1);

int rowsAffected = pstmt.executeUpdate();

}
```

Transactions

JDBC transaction make sure a set of SQL statements is executed as a unit, either all of the statements are executed successfully, or NONE of the statements are executed (rolled back all changes).

By default, each SQL statement is committed right after it is completed.

We need to set the autoCommit property of Connection to false, then use the commit() and rollback() methods to control the transaction.

The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, Isolation and Durability.

Transactions (Sample Code)

```
boolean autoCommit = con.getAutoCommit();
try {
    con.setAutoCommit( false);
    pstmt.executeUpdate();
    pstmt2.executeUpdate();
    con.commit();
} catch (SQLException exc) {
    con.rollback();
} finally {
    con.setAutoCommit(autoCommit);
}
```

Problems of JDBC API

We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.

We need to perform exception handling code on the database logic.

We need to handle transaction.

Repetition of all these codes from one to another database logic is a time consuming task.

Request Flow