Persistence layer using Spring JDBC and Spring Data

Created	@Sep 11, 2020 9:08 PM
Tags	
Updated	@Oct 6, 2020 5:38 PM

```
# mysql
docker pull mysql
docker run --name mysql -e MYSQL_ROOT_PASSWORD=pass -p 3306:3306 -d mysql

# mongo
docker pull mongo
docker run --name mongo -e MONGO_INITDB_ROOT_USERNAME=root -e MONGO_INITDB_ROOT_PASSWORD=pass -p 27017:27017 -d mongo

# create user for mongo!!!
docker exec -it mongo bash
mongo -u root
use {{database_name}}

db.createUser({user: "testUser", pwd: "pass", roles : [{role: "readWrite", db: "{{database_name}}"}]});
db.createUser({user: "user", pwd: "pass", roles : [{role: "readWrite", db: "{{database_name}}"}]});

# redis
docker pull redis
docker run --name redis -p 6379:6379 -d redis
```

JDBC

JDBC (Java Database Connectivity) is the Java API that manages connecting to a database, issuing queries and commands, and handling result sets obtained from the database. Released as part of JDK 1.1 in 1997, JDBC was one of the first components developed for the Java persistence layer.

JDBC was initially conceived as a client-side API, enabling a Java client to interact with a data source. That changed with JDCB 2.0, which included an optional package supporting server-side JDBC connections. Every new JDBC release since then has featured updates to both the client-side package (`java.sql`) and the server-side package (`javax.sql`). JDBC 4.3, the most current version as of this writing, was released as part of Java SE 9 in September 2017.

Steps:

1. Install or locate the database you want to access.
2. Include the JDBC library.
3. Ensure the JDBC driver you need is on your classpath.
4. Use the JDBC library to obtain a connection to the database.
5. Use the connection to issue SQL commands.
6. Close the connection when you're finished.

```
// Listing most common JDBC imports

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

// Example of usage
class WhatIsJdbc{
  public static void main(String[] args) {
    Connection conn = null;
    try {
      String url = "jdbc:sqlite:path-to-db-file/chinook/chinook.db";
      conn = DriverManager.getConnection(url);

      Statement stmt = null;
      String query = "select * from albums";
      try {
          stmt = conn.createStatement();
          ResultSet rs = stmt.executeQuery(query);
          while (rs.next()) {
              String name = rs.getString("title");
              System.out.println(name);
          }
      } catch (SQLException e ) {
          throw new Error("Problem", e);
      } finally {
          if (stmt != null) { stmt.close(); }
      }

    } catch (SQLException e) {
        throw new Error("Problem", e);
    } finally {
      try {
        if (conn != null) {
            conn.close();
        }
      } catch (SQLException ex) {
          System.out.println(ex.getMessage());
      }
    }
  }
}
```

PreparedStatements

One easy way to increase the flexibility of your code is to replace the `Statement` class with `PreparedStatement`

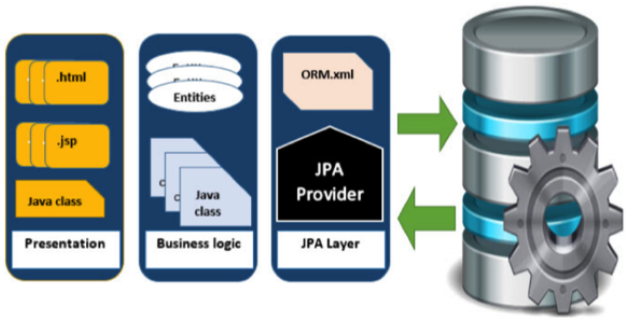
```
String prepState = "insert into albums values (?, ?);";

PreparedStatement  prepState  =
        connection.prepareStatement(sql);

prepState.setString(1, "Uprising");
prepState.setString(2, "Bob Marley and the Wailers  ");

int rowsAffected = preparedStatement.executeUpdate();
```

What is JPA and ORM frameworks

Java Persistence API is a collection of classes and methods to persistently store the vast amounts of data into a database which is provided by the Oracle Corporation.

Where to use JPA?

To reduce the burden of writing codes for relational object management, a programmer follows the 'JPA Provider' framework, which allows easy interaction with database instance. Here the required framework is taken over by JPA.



JPA History

Earlier versions of EJB, defined persistence layer combined with business logic layer using javax.ejb.EntityBean Interface.

- While introducing EJB 3.0, the persistence layer was separated and specified as JPA 1.0 (Java Persistence API). The specifications of this API were released along with the specifications of JAVA EE5 on May 11, 2006 using JSR 220.
- JPA 2.0 was released with the specifications of JAVA EE6 on December 10, 2009 as a part of Java Community Process JSR 317.
- JPA 2.1 was released with the specification of JAVA EE7 on April 22, 2013 using JSR 338.

JPA Providers

JPA is an open source API, therefore various enterprise vendors such as Oracle, Redhat, Eclipse, etc. provide new products by adding the JPA persistence flavor in them. Some of these products include:

Hibernate, Eclipselink, Toplink, Spring Data JPA.

Object-relational mapping (**ORM**, **O/RM**, and **O/R mapping tool**) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to construct their own ORM tools.

Object/Relational Mapping

Hibernate ORM enables developers to more easily write applications whose data outlives the application process. As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC). Unfamiliar with the notion of ORM? Read here.

JPA Provider

In addition to its own "native" API, Hibernate is also an implementation of the Java Persistence API (JPA) specification. As such, it can be easily used in any environment supporting JPA including Java SE applications, Java EE application servers, Enterprise OSGi containers, etc.

Idiomatic persistence

Hibernate enables you to develop persistent classes following natural Object-oriented idioms including inheritance, polymorphism, association, composition, and the Java collections framework. Hibernate requires no interfaces or base classes for persistent classes and enables any class or data structure to be persistent.

High Performance

Hibernate supports lazy initialization, numerous fetching strategies and optimistic locking with automatic versioning and time stamping. Hibernate requires no special database tables or fields and generates much of the SQL at system initialization time instead of at runtime.

Hibernate consistently offers superior performance over straight JDBC code, both in terms of developer productivity and runtime performance.

Scalability

Hibernate was designed to work in an application server cluster and deliver a highly scalable architecture. Hibernate scales well in any environment: Use it to drive your in-house Intranet that serves hundreds of users or for mission-critical applications that serve hundreds of thousands.

Reliable

Hibernate is well known for its excellent stability and quality, proven by the acceptance and use by tens of thousands of Java developers.

```
import org.thoughts.on.java.model.Book;

public interface BookRepository {

    Book getBookById(Long id);

    Book getBookByTitle(String title);

    Book saveBook(Book b);

    void deleteBook(Book b);
}



import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;

import org.thoughts.on.java.model.Book;

public class BookRepositoryImpl implements BookRepository {

    private EntityManager em;

    public BookRepositoryImpl(EntityManager em) {
        this.em = em;
    }

    @Override
    public Book getBookById(Long id) {
        return em.find(Book.class, id);
    }

    @Override
    public Book getBookByTitle(String title) {
        TypedQuery<Book> q = em.createQuery("SELECT b FROM Book b WHERE b.title = :title", Book.class);
        q.setParameter("title", title);
        return q.getSingleResult();
    }

    @Override
    public Book saveBook(Book b) {
        if (b.getId() == null) {
```

```
            em.persist(b);
        } else {
            b = em.merge(b);
        }
        return b;
    }

    @Override
    public void deleteBook(Book b) {
        if (em.contains(b)) {
            em.remove(b);
        } else {
            em.merge(b);
        }
    }
}
```

Usefull links:

- https://www.baeldung.com/spring-jdbc-jdbctemplate
- https://www.tutorialspoint.com/spring/spring_jdbc_example.htm
- https://thorben-janssen.com/implementing-the-repository-pattern-with-jpa-and-hibernate/
- https://www.baeldung.com/java-dao-vs-repository
- https://attacomsian.com/blog/accessing-data-spring-data-jpa-mysql