Contents

* Spring Data JPA
* Angular
* Spring REST
* Spring Microservices
* AWS

Spring Data JPA

Pre-requisites:

* SQL
* JDBC
* ORM/Hibernate/JPA
* Spring Boot

Drawbacks of JDBC

1. You need to write code to create the connection instance
2. You don’t follow any standards to maintain the datasource informations, because you can keep the datasource informations either in Java file/Properties/XML and so on.
3. Most of the checked exceptions should be handled for every operation’s
4. SQL queries are database dependent, the sql queries you write in the JDBC may not work in all the database
5. You write too much code, only to perform type conversion from Java to SQL and vice versa
6. Lot of boilerplate code are written in JDBC.
7. Queries will become more complex if the operations are complex, ex: join queries, sub-queries

ORM (Object Relational Mapping) Framework:

This overcomes the drawback of JDBC, ORM framework will map java objects to the database tables with some predefined functions provided by ORM framework, it avoids lot of things like

1. Type conversion is not required i.e., Java to SQL & vice versa
2. Follows the standard, i.e., datasource informations must be kept in the configuration file (XML)
3. Queries are database independent and in ORM you can avoid writing queries
4. It can create connection for you which you can use it
5. You don’t have to handle any checked exceptions
6. Boiler plate codes are avoided
7. Complex operations can be done with simple code like complex joins can be done with simple annotations

ORM is a specification of JPA (Java Persistence API) i.e., standard, there are many organizations provided the implementation to the ORM

Some of the ORM frameworks are:

1. Hibernate (red-hat)
2. JPA (sun microsystem)
3. Toplink
4. iBatis

Out of the above ORM frameworks, hibernate is the one which is more widely used

Interacting with the database using Hibernate

Steps:

1. you need to create maven project
2. you need to add the hibernate dependencies
3. you need to add the jdbc driver dependencies
4. you need to configure an xml file that will have the datasource information’s, by default the configuration name hibernate looks for is hibernate.cfg.xml
5. you need to create entity classes which will have tables & columns mappings through annotations
   1. @Entity, @Table, @Column, @Id

Note: Primary key is mandatory

Step1: Creating the maven project

Step2: Adding hibernate & mysql-connector dependencies



pom.xml



hibernate.cfg.xml



Create an employee table in mysql



Creating entity class & mentioning that class in the <mapping> tag of hibernate.cfg.xml

Step 1:

Employee.java



Source Code:

**package** com.cognizant;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

@Entity

**public** **class** Employee {

@Id

@Column(name = "id")

**private** **int** empId;

**private** String name; // @Column is not required because variable & column names are same

**private** **double** salary; // @Column is not required because variable & column names are same

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

}

Step 2: Mention the entity class in the hibernate.cfg.xml



Step 3:

Performing CRUD operations on the entity.

You will use following instances to perform the CRUD operations

* Configuration cfg = new Configuration();

cfg.configure(); // default chooses hibernate.cfg.xml

cfg.configure(“xml file”);

* SessionFactory factory = cfg.buildSessionFactory();
* Session session = factory.openSession();
* session.save(object1);
* session.save(object2)
* session.get(classname.class, primary\_key)

You will get SSL exception so use below url in the hibernate.cfg.xml



HibernateDemo.java

**package** com.cognizant;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.Configuration;

**public** **class** HibernateDemo {

**public** **static** **void** main(String[] args) {

// 1st step is to create Configuration object that loads the xml configuration file

Configuration cfg = **new** Configuration();

cfg.configure();

// 2nd step is to create SessionFactory

SessionFactory factory = cfg.buildSessionFactory();

// 3rd step is to create Session

Session session = factory.openSession();

Employee employee = **new** Employee();

employee.setEmpId(200);

employee.setName("Bruce");

employee.setSalary(34200);

// 4th step is to save the object but you must begin the transaction

Transaction tx = session.beginTransaction();

session.save(employee);

tx.commit();

session.close();

factory.close();

System.***out***.println("DONE!");

}

}

HibernateDemoToRetrieve.java

**package** com.cognizant;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.cfg.Configuration;

**public** **class** HibernateDemoToRetrieve {

**public** **static** **void** main(String[] args) {

// 1st step is to create Configuration object that loads the xml configuration file

Configuration cfg = **new** Configuration();

cfg.configure();

// 2nd step is to create SessionFactory

SessionFactory factory = cfg.buildSessionFactory();

// 3rd step is to create Session

Session session = factory.openSession();

Employee emp = session.get(Employee.**class**, 100);

System.***out***.println("Id: "+emp.getEmpId()+", Name: "+emp.getName()+", Salary: "+emp.getSalary());

session.close();

factory.close();

System.***out***.println("DONE!");

}

}

Criteria object in hibernate:

It allows you to fetch all the records in a Collection and can also apply some condition while fetching all the records

To create criteria you will use

Criteria crt = session.createCriteria(EntityClass.class);

List list = crt.list();

Exercise:

Create a menu driven program which shows following options

1: Insert

2: Retrieve by Id

3: Display All

4: Delete by Id

5: Update the salary based on Id

6: Exit

Note: Perform all the CRUD operations in a separate class but create this menu in main class

Note: Reuse the Session Factory instance in separate class

Applying the conditions in the criteria

In criteria you can apply lot of conditions like you apply in sql query.

Criteria crt = session.createCriteria(Employee.class)

crt.add(Restrictions.eq(“variableName”, value));

variableName is the property that maps to columns



Output



Till now we didn’t write any queries, but hibernate/ORM supports passing queries also, the queries write in ORM is called as JPQL/HQL

JPQL: Java Persistence Query Language

HQL: Hibernate Query Language, it is same as JPQL.

In JPQL/HQL you write queries for entities not for the tables.

Ex:

from com.cognizant.Employee : means you get all the entities

select emp from com.cognizant.Employee emp: emp is an alias to access entity property

Some of the conditions also you can write with that alias.

Select emp from com.cognizant.Employee emp

where emp.empId = 100

Above query will get employee entity for the employee id 100

The above query can ignore the package names if the classes don’t have any conflicts

i.e.,

select emp from Employee emp where emp.empId = 100;

Here emp refers to the object completely, however you can also access only few properties

Select emp.empId, emp.name from Employee emp;

Now you will get only empId and name but not salary



Output:



You can also get only few properties instead of getting whole entity



Here select e.empId, e.name from Employee e will only have part of the entity, hence it is going to create an object of type Object that will have 2 properties of Object type(id & name), which is why you will typecast to Object[]