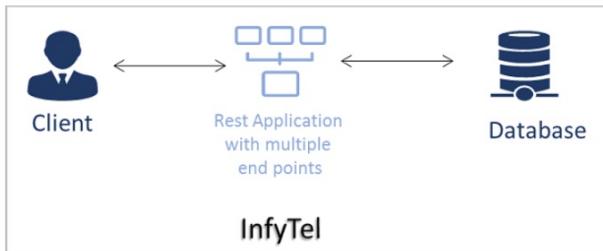


Spring data JPA with Boot

Prledge

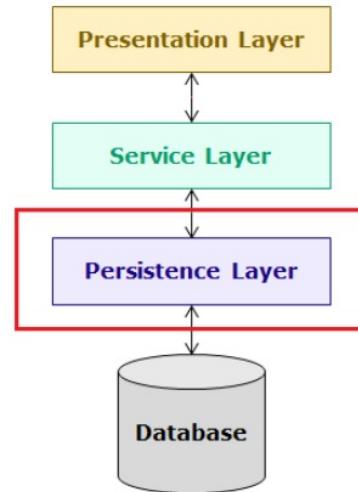
For implementing the Persistence Layer of an enterprise application Spring uses many approaches. A developer can choose appropriate Spring modules such as Spring JDBC, Spring ORM, or Spring Data JPA for implementing the data access layer of an enterprise application depending on the repository type. Spring simplifies transactions by allowing the developer to implement transactions in a declarative way using simple annotations and configurations. For the more fine-grained transactions, Spring also supports programmatic transactions.

In this course, we will focus on how to implement the data access layer of an enterprise application by using Spring Data JPA with Spring Boot.



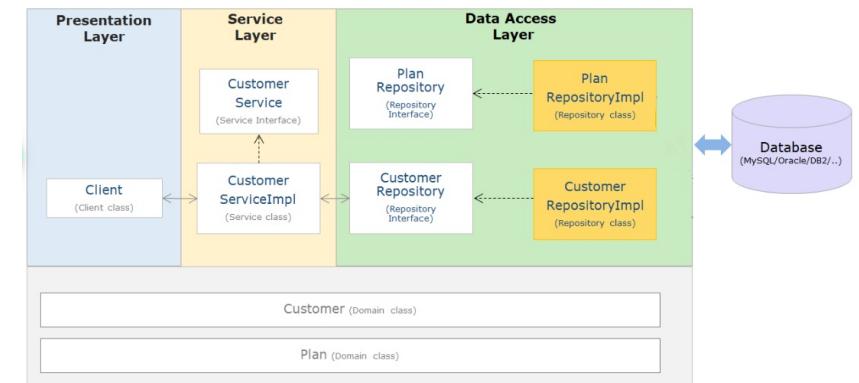
InfyTel is built as a three-tier application that consists of

- Presentation Layer
- Service/Business Layer
- Persistence Layer



features supported by appl^n

- Add Customer
- Edit Customer Details
- Search Customer
- View all Customer
- Remove Customer



Introduction to Spring Data JPA

Consider the InfyTel application. The Admin of the InfyTel application has to maintain its Customer and Plan details. In this course, we will see how to implement the persistence layer of the InfyTel application using Spring Data JPA.

- Add Customer - Adding customer records
- Remove Customer - Deleting customer records

The following are the steps to implement the previous requirements of the InfyTel application.

Step 1: Define an entity class Customer.java to represent customer details.

Step 2: Add an appropriate database connector dependency in pom.xml.

Step 3: Create a data access layer with an interface CustomerDAO and a repository class CustomerDAOImpl to implement the interface.

Step 4: Create a service layer with an interface CustomerService and CustomerServiceImpl class to implement the interface

Step 5: Create a presentation layer with client class to access database operations and display details on the console

Step 6: Create a table Customer in the database.

```
create table customer(
    phone_no bigint primary key,
    name varchar(50),
    age integer,
    gender char(10),
    address varchar(50),
    plan_id integer
);
```

→ ORM (object relational mapping)

Object Relational Mapping (ORM) is a technique or design pattern, which maps object models with the relational model.

- In Programming languages like Java, the related information or the data will be persisted in the form of hierarchical and interrelated objects.
- In the relational database, the data is persisted as table format or relations.

The greatest challenge in integrating the concepts of RDBMS and OOP is a mapping of the Java objects to databases. When object and relational paradigms work with each other, a lot of technical and conceptual difficulties arise, as mapping of an object to a table may not be possible in all the contexts. Storing and retrieving Java objects using a Relational database exposes a paradigm mismatch called "Object-Relational Impedance Mismatch". These differences are because of perception, style, and patterns involved in both the paradigms that lead to the following paradigm mismatches:

- Granularity: Mismatch between the number of classes in the object model and the number of tables in the relational model.
- Inheritance or Subtype: Inheritance is an object-oriented paradigm that is not available in RDBMS.
- Associations: In object-oriented programming, the association is represented using reference variables, whereas, in the relational model foreign keys are used for associating two tables.
- Identity: In Java, object equality is determined by the "==" operator or "equals()" method, whereas in RDBMS, uses the primary key to uniquely identify the records.
- Data Navigation: In Java, the dot(.) operator is used to travel through the object network, whereas, in RDBMS join operation is used to move between related records.

⇒ Limitations of JDBC API

Let us understand the limitations of JDBC API

- A developer needs to open and close the connection.
- A developer has to create, prepare, and execute the statement and also maintain the resultset.
- A developer needs to specify the SQL statement(s), prepare, and execute the statement.
- A developer has to set up a loop for iterating through the result (if any).
- A developer has to take care of exceptions and handle transactions.

The above limitations can be solved with the technologies Spring ORM.

What is ORM?

Object Relational Mapping (ORM) is a technique or design pattern, which maps object models with the relational model. It has the following features:

- It resolves the object-relational impedance mismatch by mapping
 - Java classes to tables in the database
 - Instance variables to columns
 - Objects to rows in the table
- It helps the developer to get rid of SQL queries. They can concentrate on the business logic and work with the object model which leads to faster development of the application.
- It is database independent. All database vendors provide support for ORM. Hence, the application becomes portable without worrying about the underlying database.

Benefits of ORM

- ORM provides a programmatic approach to implement database operations.
- ORM maps Java objects to the relational database tables in an easier way based on simple configuration.
- Supports simple query approaches like HQL(Hibernate Query language) and JPQL (Java Persistence Query Language)
- Supports object-oriented concepts such as inheritance, mapping, etc.

Java Persistence API (JPA) uses ORM to provide :-

- Defines an API for mapping the object model with the relational model
- Defines an API for performing CRUD operations
- Standardizes ORM features and functionalities in Java.
- Provides an object query language called Java Persistence Query Language (JPQL) for querying the database.
- Provides Criteria API to fetch data over an object graph.

Configuration Required :-

@Repository

It indicates the repository class(POJO class) in the persistence layer. Repository class is defined using `@Repository` annotation at the class level. This annotation is a specialization of `@Component` annotation for the persistence layer.

Benefits of the `@Repository`:

- Enables the Spring Framework auto scanning support to create a repository bean without the explicit bean definition in the configuration file.
- Causes Spring exception translation(translates checked exceptions into unchecked exceptions).

JPA needs an Entity class to perform all the CRUD operations because Entities can represent fine-grained persistent objects and they are not remotely accessible components.

Entity Class:

An Entity class is a class in Java that is mapped to a database table in a relational database.

```
package com.infotel.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity
public class Customer {

    @Id
    @Column(name = "phone_no")
    private long phoneNumber;
    private String name;
    private Integer age;
    private Character gender;
    private String address;
    @Column(name = "plan_id")
    private Integer planId;
    //constructors
    //getters and setters
}
```

`@Entity` :- declare as entity class

`@Id` :- " " primary key

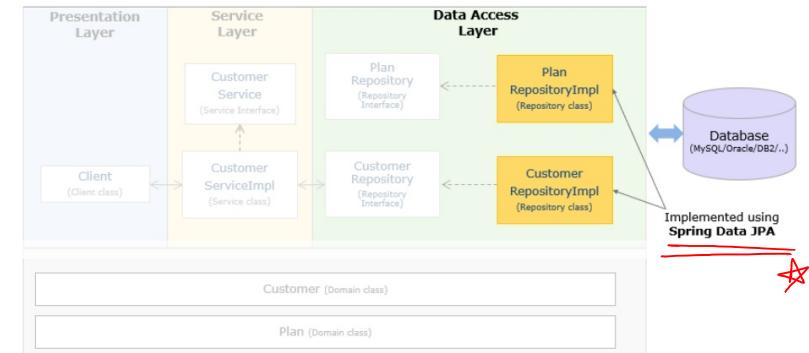
`@Column`:- declare mapping of a column in database

`@Table`:- declare mapping of a entity class to a table name

`@Temporal`:- converts the date & time values from java objects to compatible database type

`@Transient`:- it will not persisted to database

Why Spring data JPA ?



→ Spring data JPA overcome these limitations of Spring ORM JPA (with spring boot)

- The Programmer has to write the code to perform common database operations(CRUD operations) in repository class implementation.
- A developer can define the required database access methods in the repository implementation class in his/her own way, which leads to inconsistency in the data access layer implementation.

What is Spring data JPA

Spring data is high level project from Spring whose objective is to unify and ease of access to SQL & noSQL databases.

It removes DAO(repository) implementation entirely.

It helps to implement persistence layer and reducing the effort.

Spring data commons provide basic interfaces to support :-

- ↳ CRUD operations
- ↳ Sorting of data
- ↳ Pagination
- ↳ Specific abstraction
- ↳ JpaRepository interface to support JPA
- ↳ MongoRepository interface to support MongoDB

Introduction to Spring boot .

Spring is lightweight framework to develop enterprise applications :-
but it has difficult configurations and project dependency management

That's why Spring boot is developed :-

Spring Boot is a framework built on the top Spring framework that helps developers build Spring-based applications quickly and easily. The main goal of Spring Boot is to quickly create Spring-based applications without requiring developers to write the same boilerplate configuration again and again.

1. Spring Boot is an opinionated framework

Spring Boot forms opinions. It means that Spring Boot has some sensible defaults which you can use to quickly build your application.

default web container of Springboot is Tomcat.

2. Spring Boot is customizable

Though Spring Boot has its defaults, you can easily customize it at any time during your development based on your needs.

The main Spring Boot features are as follows:

1. Starter Dependencies
2. Automatic Configuration
3. Spring Boot Actuator
4. Easy-to-use Embedded Servlet Container Support

Note :-

The @SpringBootApplication annotation indicates that it is a configuration class and also triggers auto-configuration and component scanning. It is a combination of following annotations with their default attributes.

- @EnableAutoConfiguration - This annotation enables auto-configuration for Spring boot application which automatically configures our application based on the dependencies that a developer has already added.
- @ComponentScan - This enables the Spring bean dependency injection feature by using @Autowired annotation. All application components which are annotated with @Component, @Service, @Repository or @Controller are automatically registered as Spring Beans. These beans can be injected by using @Autowired annotation.
- @Configuration - This enables Java based configurations for Spring boot application.

→ Spring data JPA CRUD

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long> {
}
```

only need to extend the JpaRepository interface



Paging and Sorting

```
public interface PagingAndSortingRepository<T, ID extends Serializable> extends CrudRepository<T, ID> {  
    Iterable<T> findAll(Sort sort);  
    Page<T> findAll(Pageable pageable);  
}
```

A Page object provides the data for the requested page as well as additional information like total result count, page index, and so on.

Steps to create pagination

Step 1

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {  
}
```

Step 2

```
//First argument '0' indicates first page and second argument 4 represents number of records.  
Pageable pageable = PageRequest.of(0, 4);
```

In client code:-

```
Page<Customer> customers = customerRepository.findAll(pageable);
```

Steps to create sorting

Step 1

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {  
}
```

In client code :-

```
customerRepository.findAll(Sort.by(Sort.Direction.ASC, "name"));
```

In the Sort() method used above,

- The first parameter specifies the order of sorting i.e. is ascending order.
- The second parameter specifies the field value for sorting.

Demo to create pagination & sorting

Customer Service Interface

```
public interface CustomerService {  
  
    public void insertCustomer(CustomerDTO customer);  
    Page<Customer> findAll(Pageable page);  
    List<Customer> findAll(Sort sort);  
}
```

Implementation of same

```
@Service("customerService")  
public class CustomerServiceImpl implements CustomerService{  
    @Autowired  
    private CustomerRepository repository;  
  
    public void insertCustomer(CustomerDTO customer) {  
        repository.saveAndFlush(CustomerDTO.prepareCustomerEntity(customer));  
    }  
  
    @Override  
    public Page<Customer> findAll(Pageable page) {  
        return repository.findAll(page);  
    }  
    @Override  
    public List<Customer> findAll(Sort sort) {  
        return repository.findAll(sort);  
    }  
}
```

repository
that
extends
JpaRepository

