Spring Day-1

05 May 2022 12:45

1. A framework is a platform for developing s/w application.
2. A framework is a platform used as a foundation for developing app.
3. A framework is a software for creating or developing Java application quickly and efficiently and can be maintained easily.
4. A framework is built on top of some existing technologies.

What is Spring Framework and Why Spring ?

1. Spring is an application development framework.
2. Spring framework is a framework of frameworks.
3. Spring framework is developed in modular fashion - spring core, spring context, spring DAO, spring WEB, spring AOP, etc.
4. By using Spring framework we can develop several types of application like -
   a. Stand-alone app
   b. Web app (C2B)
   c. Distributed app (one app is accessing the functionality of another app.
      Example : For passport application -> It takes data of Aadhar. [B2B app])
5. Spring enables you to build app from "Plain Old Java Object" POJO.
6. Spring framework provides a light-weight solution to develop maintainable and reusable enterprise application.
7. It provides very simple and rich facilities to integrate various frameworks, technologies, and services in the application.

POJO

If a java class is not coupled with any technology or any framework then it is called POJO.
Example:

Class MyClass extends MyOtherClass	Not POJO
Class MyClass	POJO class
class MyClass implements java.io.serializable	POJO class

Tight Coupling vs Loose Coupling

When one class is calling the business logic or method of another class is called collaboration.
When one Class collaborate with another Class there it exist tight coupling between two classes.
When one class method wants to call the method of second class then 1st class needs to create object of the 2nd class.

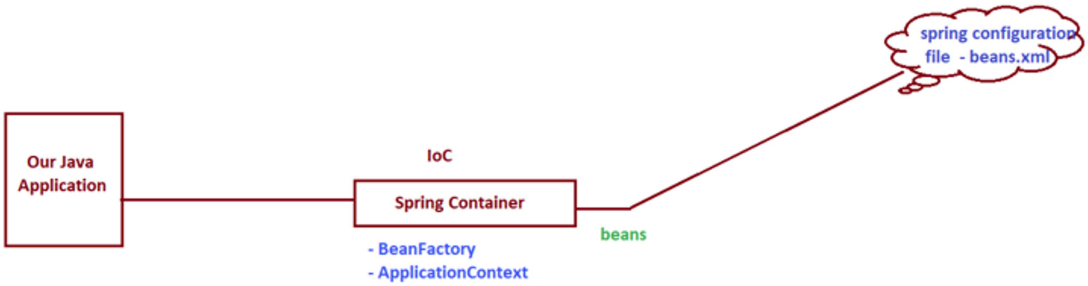
Traveler ---- Car

Class Car{
public void drive(){
//logic
}
}
	Class Traveler{
	Car car = new Car();
	public void startJourney(){
	car.drive();
	}
	}
		Here Traveler object is tightly coupled with car object.

Loose coupling : When one object is depending on another class object some external entity will provide the dependency object to the main object & that external
Entity can be called as a container.

Interface Vehicle{
public void drive();
}
	class Car implements Vehicle{
	public void drive(){
	// using car
	}
	}
		Class Bike implements Vehicle{
		public void drive(){
		// using bike
		}
		}
			Class Traveler{
			Vehicle vehicle;
			public void setVehicle (Vehicle v)
			{
			this.vehicle = v;
			}
			public void startJourney() {
			vehicle.drive();
			}
			}
				// vehicle can be car or bike & will be injected to this method.
				
				The dependent object can be passed through constructor & is called constructor injection.

Object which are created by Spring containers is called Beans.



Spring Configuration

Following are the 3 important ways to provide configuration metadata to the spring container

- XML based configuration
- Annotation based configuration
- JAVA based configuration

Spring says -> you do not create any object in your application. What all objects are required you specify that either through XML, Annotation or through Java based configuration.
The Spring container will create that object.

- Central to the Spring framework is its Inversion Of Control (IoC) containers, which provides a consistent mean of configuring and managing Java Objects.
- The container is responsible for managing object lifecycles of specific objects, calling their initialization methods, and configuring these objects by wiring them together.

Spring Container

It is the basic container. All other possible classes that act as containers implements BeanFactory.
It is the root container that loads all the beans and provides dependency injection to enterprise application.

Spring ApplicationContext container

It is widely used as a container in the enterprise applications with a number of features.
It is an advance container that extends the BeanFactory container with various enterprise-level features.

Transitive Dependency -> When we add 1 dependency and it add multiple dependencies/dependency.

Example : When we add spring-context -> it will add spring-aop, spring-beans, spring-core, etc.

In Bean configuration file "value" can be given as:

1. An attribute
2. An element
3. Using namespaces

Dependency Injection -> It is the process of providing the dependent object to the target object and the spring container does it for us.

Types:

1. Setter injection - required bean is injected through setter method for that bean.
2. Constructor injection - the required bean is injected through the constructor argument.
3. Field injection -

- Apart from injecting a bean into another bean, we can inject values for collection like Set, List, Map.
- We can inject inner bean into another bean.
- Create an alias for a bean and refer a bean using idref.

Inner beans are the beans that are defined within the scope of another bean.

idref in Spring

- The idref element is a way to pass the id (String value-not a reference) of another bean
  In the container to a <constructor-arg/> or <property/> element.

Usual way :	<property name= "targetName" value = "targetBean" /> </bean>

Right way:	<property name = "targetName"> <idref bean = "targetBean"/> </property>

```
</bean>
```

- Idref allows the container to validate at deployment time that the referenced, named bean actually exists.
- In the usual way no validation is performed on the value that is passed and typos are only discovered when the clientBean is actually instantiated.

Spring Bean Lifecycle

1. Through xml
2. through interfaces
3. Through annotation

- init-method in the xml file specifies a method which needs to be revoked immediately after the bean instantiation.
- destory-method in the xml says what to do in a method just before it is dying.

Autowiring

- Associating different beans together is known as wiring.
- In Spring application - we got class A which requires an instance of class B to do its work. (HAS-A-Relationship)
- By using Autowiring feature - spring container can automatically wire class B instance in class A.
- Spring can automatically resolve dependencies using autowiring.
- Autowiring feature of spring framework enables us to inject the dependent object implicitly.
- Less codes

Autowiring Modes:

1. byType
2. byName
3. Constructor

JPA

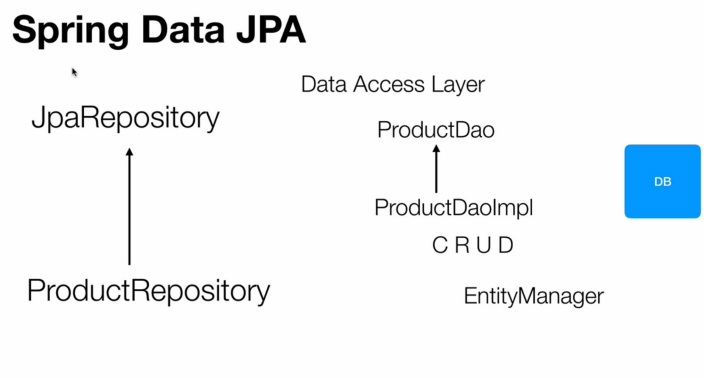
29 April 2022 11:55

Spring data JPA and spring boot make our life as a developer easy to create data access layer for our applications.

Before spring data JPA was introduced when we used then we use ORM tools like Hibernate to perform object relational mapping and execute the crud operations against the database that is create read update and delete.
We typically create a dao interface that will have all the crud operations and then a implementation class that we use entity manager from the JPA standard.

JPA stands for Java persistence API.
It is a standard from Oracle to use tools like Hibernate eclipse link etc. to do object relational mapping in our application.

So typically we'll create a DAO per entity or per table in the database then we use the entity manager inside that DAO impl and we create multiple methods which perform create read update delete etc. And we repeat the same code again for different model classes different entities or different database tables. And this duplication happens across our projects.

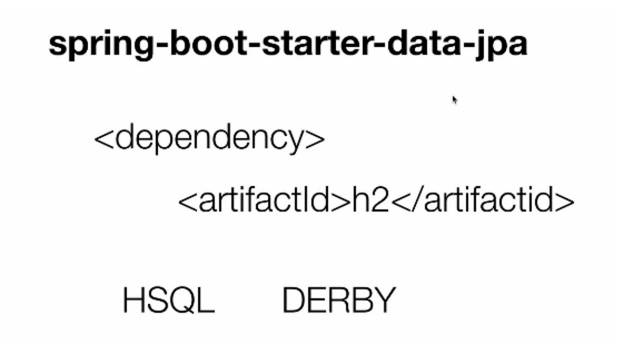


So spring data JPA use as an opportunity to avoid all the duplication. All we need to do to perform crud operations against the database is simply create a product repository or a repository interface per entity that we create if we have a order entity.
We create order repository and these repository interfaces extend interfaces from the spring data.

JPA API like JPA repository.

That is all we need to do. We need not duplicate any code once we do this. We can use this repository interfaces in the other players like web layer etc. and automatically spring will give us an opportunity to perform crud operations against the database using this interface.
And the starter project to use Spring data JPA in our spring boot applications is spring-boot-starter-data-jpa.
Once we include this we'll be able to use that repository interfaces etc and extend them in our application.

So this is the first step to include this dependency in the pom.xml
Secondly we need to add the appropriate database JDBC driver in the pom.xml
So if we are using in-memory or embedded databases like H2 then spring will automatically create all the data access layer related configuration for us.
So without writing any amount of code by simply adding this dependencies like this h2, HSQL or derby we can directly connect to this databases and work with them. When we run our applications Spring will automatically create the tables from the entity mapping that we do if you create a product.Java then automatically Spring will create a product table and will be able to execute our SQL statements against that product table in memory all that magic will happen by simply including this dependency and then the appropriate jdbc jar dependency in the Maven pom.xml



And when we use databases like MYSQL or Oracle we have some additional work where we need to provide the jdbc connection url, the user name, and password to connect to the database in the application.properties

MYSQL

Spring data JPA is a super easy way to create data access layer for our applications. Instead of repeating the crud operation code across DAO's we simply create an interface per entity in our application.
Those interfaces extend from one of the interfaces in the spring data JPA API.
The first step is to include the spring boot starter project which will bring in all the other jars that are required called spring boot starter data JPA.
And the second dependency is our database dependency.

JPA

29 April 2022 12:52

Spring boot data JPA is used for developing Persistence layer.

Persistence layer is responsible to communicate with Persistence store (Database).

Persistence Operations:

1. Creating (or) Inserting Data
2. Update (or) Updating Data.
3. Delete (or) Deleting Data.
4. Retrieve (or) Retrieving Data.

CRUD

Persistence Technologies

1. Persistence Technologies are used to develop Persistence logic.
2. Persistence logic is used to perform Persistence operations on Persistence store.
   Example: Java JDBC, ORM Frameworks, etc.

Best Practices to Develop Persistence logic

- We should maintain a separate layer for Persistence logic - DAO Layer/ Repo. Layer - Data Access Layer
- In Persistence logic we should follow Table per DAO Concept.

1. For every DB Table we should create a dedicated DAO (Single Responsibility Principle).

USER_MASTER (DB Table)	UserMasterDao.java (Interface) + UserMasterDaoImpl.java (class)
ROLE_MASTER (DB Table)	RoleMasterDao.java (Interface) + RoleMasterDaoImpl.java (class)

2. For every database table :
   a. it is recommended to have at least one Primary Key column (to avoid duplication/ to maintain unique data).
   b. It is recommended to maintain the following additional column for auditing.

CREATED_BY	CREATED_DATE	UPDATED_BY	UPDATED_DATE

3. PRIMARY_KEY value should be generated automatically.

Common operations in every DB Table:

1. Insert record(s)
2. Update record(s)
3. Retrieve record based on primary key
4. Retrieve few or all records
5. Delete record based on primary key
6. Delete few or all records
7. Check the presence of a record
8. Get count of records
9. Etc

-> To perform above operations we need to write methods in DAO interface and we should implement those methods in DAO implementation class.

What is the best practice?
Scenario-1
-> Suppose we have one DB table in project - then we will create one DAO with all the above methods and we will provide the implementation
Scenario-2
-> Suppose we have 2 DB table in project - then we will create 2 DAO with all the above methods and we will provide the implementation

Scenario-3
-> Suppose we have 100 DB table in project - then we will create 100 DAOs with all the above methods and we will provide the implementation

100 Daos * 8 methods = 800 methods we should write

Note: All the 800 method logics will be same. All the implementation classes contains the same logic (boiler plate code)

-> To avoid boiler plate code in DAO layer, Spring Team provided 'Spring Data'
-> If we are using Spring Data, then we dont need to write any methods to perform above CRUD operations on DB Table.
-> In Data JPA, already methods are available to perform CRUD operations

Spring Data provided 2 predefined repository interfaces for us:

1. CrudRepository
2. JpaRepository

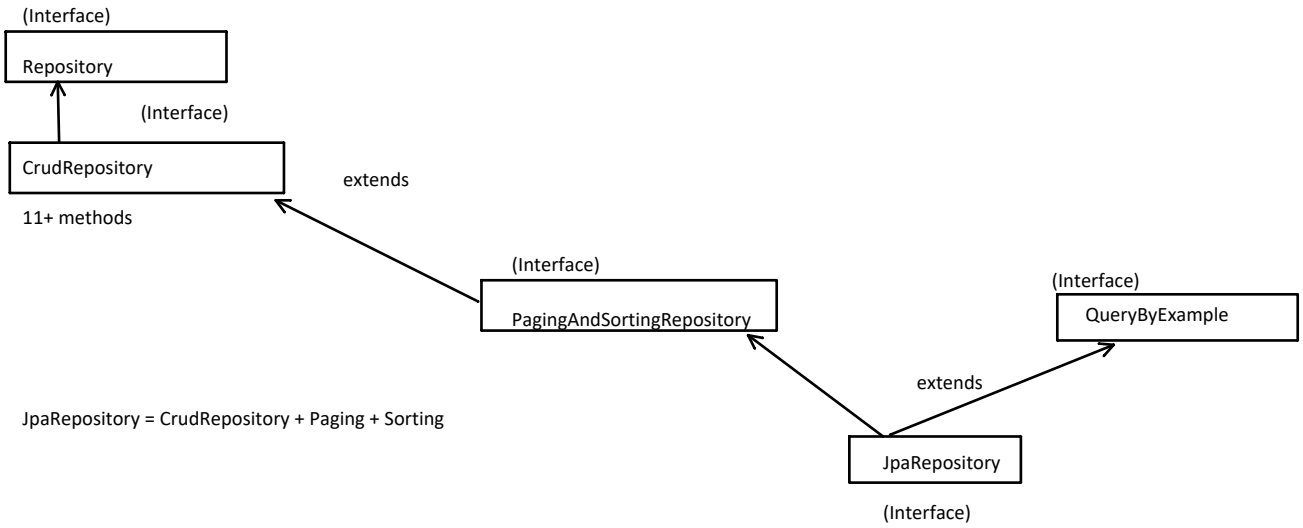
-> these 2 interfaces provided several methods to perform CRUD operations on DB Table

-> To work with Spring Data , we should create our repository interface by extending any one of the repository interfaces provided by SPring Data.

```
public interface UserMasterRepository extends CrudRepository<Entity, Serializable ID>{}

public interface UserMasterRepository extends JpaRepository<Entity, Serializable ID>{}
```

-> CrudRepository provided methods to perform CRUD operations
-> JpaRepository provided methods to perform CRUD operations with Pagination and Sorting



Steps to build Spring Boot Application using Data JPA

1) Install Database Software
2) Create Table in the Database
3) Create Spring Boot Application with the following dependencies
   Spring Data JPA, Database Driver (MySQL Driver), Project Lombok
4) Configure Data Source properties in application.properties (or) application.yml
5) Create Entity class to map with Database Table
6) Create Repository Interface for our entity class by extending Spring Data Repository -> CrudRepository (or) JpaRepository
7) Get the Repository Bean in start class main method and call CRUD methods
8) Execute Application