Innovation week : Spring Framework Summary

Jonathan Fairbanks

July 19, 2021

Contents

1 Spring Framework

1.1 Spring overview

Spring is a IoC (Inversion of Control) framework which is a design pattern that has the intention of removing dependencies from your code.

Examples often showing between an object being isntantiated via setter or a constructor .

1.2 AOP (Aspect Oriented Programming)

Spring is also what is called a Aspect oriented framework which means it seperates things out in a way. (described as cross cutting concerns) an example is having something like logging being part of the application running and the "business" logic written somewhere else.

1.3 Beans

Beans are objects that are managed by Spring

2 JPA (Java Persistence API)

2.1 Defining Entities (Models)

Entities are persistent domain objects that usually represent a table in a database. and each instance of an entity corresponds with a row.

2.1.1 Decorators

1. @Entity Decorator that is placed over a class definition. which marks the object as a JPA Entity

2. @Id The decorator that lets JPA know which of the objects ID is is its ID (Usually a Long or some other number)

3. @GeneratedValue [1] This is paired with the @Id decorator and lets JPA know how the ID should be generated. ex: @GeneratedValue(strategy=GenerationType.AUT

   if this decorator is ommited it will selected AUTO as its generation type

   (a) AUTO The default generation type and will be based on the persistence provider.

   (b) IDENTITY Relies on an auto incremented database column and generates a new value for each insert operation. Not a good option for hibernate as it requires the insert immediety reducing performance.

   (c) SEQUENCE uses a database sequence to generate unique values, but requires an additonal select statement when doing so. via use of the @SequenceGenerator(x="",y="") or if that is ommitted it will request the next value from the default sequence.

   (d) TABLE simulates sequence by using by using transactional locks which will reduce performance, use SEQUENCE if DB supports it

4. @SequenceGenerator you can optionally specifiy additional arguments to

5. @Column good practice to specify the name of a column in a table, can also specify the properties of a column

Identifier	Data Type	Default
name	String	""
insertable	Boolean	true
length	int	255
precision	int	0
scale	int	0
nullable	Boolean	true
columnDefinition	String	""
unique	Boolean	false
updatable	Boolean	true

   (a) name : just the name of the column

   (b) insertable : whether SQL insert statements can be used

   (c) length : column length

(d) precision : decimal precision

(e) scale : scale for a decimal column

(f) nullable : Defines if it is nullable

(g) columnDefinition : change definition ex: varChar(256)

(h) unique : if it has a uniqueness constraint

(i) updatable : if SQL update statements can be used on it

2.1.2 Entity Requirements [2]

1. Declared with @Entity
2. has a public/protected no args constructor
3. Class, methods, and variables must not be declared Final
4. Persistence variables must be private/protected and can only be modified by the entities methods
5. Entities can extend or be extended by non entity classes
6. If an entity is passed by value as a detached object the class needs to implement serializable

2.1.3 Example from `https://spring.io/guides/gs/accessing-data-jpa/#initial`:

package com.example.accessingdatajpa;
import javax.persistence.Entity; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id;
@Entity public class Customer {
@Id @GeneratedValue(strategy=GenerationType.AUTO) private Long id; private String firstName; private String lastName;
protected Customer() {}
public Customer(String firstName, String lastName) { this.firstName = firstName; this.lastName = lastName; }
@Override public String toString() { return String.format(“Customer[id=%d, firstName='%s', lastName='%s']”, id, firstName, lastName); }
public Long getId() { return id; }
public String getFirstName() { return firstName; }
public String getLastName() { return lastName; } }

2.2 Join Types

2.3 Creating Queries

the repository implementations are created automatically from an interface the interface incldues methods for saving, deleting and finding customer entities.

with this this implementation of it does not need to be written after you write the interface.

2.3.1 Example from `https://spring.io/guides/gs/accessing-data-jpa/#initial`:

package com.example.accessingdatajpa;
import java.util.List; import org.springframework.data.repository.CrudRepository;
public interface CustomerRepository extends CrudRepository<Customer, Long> {
List<Customer> findByLastName(String lastName);
Customer findById(long id); }

3 DSL Query

4 MasterControl

5 Docs

5.1 `https://docs.spring.io/spring-framework/docs/current/reference/html/`

5.2 `https://www.tutorialspoint.com/spring/spring_overview.htm`

6 Good Stack overflow pages

6.1 [1] `https://stackoverflow.com/questions/47676403/spring-generatedvalue-anno`

6.2 [2] `https://stackoverflow.com/questions/63414381/what-is-entity-in-spring-j`

6.3 [3] `https://stackoverflow.com/questions/3058/what-is-inversion-of-control`

6.4 [4] `https://stackoverflow.com/questions/2990799/difference-between-fetchtyp`