POLITECHNIKA GDAŃSKA

Spring Data JPA

Internet Services Architectures

Michał Wójcik

Spring Data

Spring Data is an umbrella module providing consisted assess mechanism to different storage types.

Spring Data

Main modules:

- Spring Data Commons - core concepts for every Spring Data module
- Spring Data JDBC - SQL database access using JDBC,
- Spring Data JDBC Ext - support for databases specific extensions,
- Spring Data JPA - SQL database access using JPA,
- Spring Data KeyValue - support for key-value stores (non-relational databases, map-reduce frameworks),
- Spring Data LDAP - support for LDAP catalog,
- Spring Data MongoDB - support for document oriented database MongoDB,
- Spring Data Redis - support for in-memory key-value database Redis,
- Spring Data REST - automatically exports repositories as rest resources,
- Spring Data for Apache Cassandra - support for wide-column store Apache Casandra,
- Spring Data for Apache Geode - support for in-memory data grid Apache Geode,
- Spring Data for Apache Solr - support for Lucene based search platform Apache Solr,
- Spring Data for Pivotal GemFire - support for distributed data management GemFire.

Java Persistence API

Java Persistence API (JPA):

- specification for object-relational mapping (ORM) libraries,
- popular in Java frameworks:
  - Java SE, Java EE, Spring Framework, Play Framework;
- does not require creating complex DAO objects (Data Access Object),
- supports ACID transactions (Atomicity, Consistency, Isolation, Durability),
- databases system provider independent:
  - JDBC drivers for all most popular database servers,
  - in simpler cases, it is possible to avoid playing with SQL.

JPA - implementations

JPA is only a standard, there is a number of implementations:

- Hibernate from Red Hat,
- Toplink from Oracle,
- OpenJPA from Apache Software Fundation,
- EclipseLink from Eclipse Fundation, reference implementation.

Entity classes

Entity classes:

- classes mapped to tables stored in the database,
- simple POJO (Plain Old Java Object) classes,
- class fields should not be public,
- each entity object must have unique identifying key:
  - complex keys are represented with separate classes implementing `hashCode()` i `equals()` methods;
- defined with annotations.

JPA - important annotations

Annotations on **class** level:

- `@Entity` – mark class as entity (requried),
- `@Table` – table properties, e.g.:
  - `name` – table name,
  - `indexes` – additional indexes (besides default index for primary key).

JPA - important annotations

Annotations on **field** level:

- `@Id` – primary key,
- `@GeneratedValue` – automatically generated value for primary key,
- `@Column` – column properties, e.g.:
  - `name` – column name,
  - `nullable` – if can be null or if is required,
  - `unique` – if column values are unique,
  - `updatable` – if column value can up updated after row creation;
- `@Temporal` – required for `Date` i `Calendar` types:
  - allows to user database date/time types to be used to store values (if database supports them);
- `@Transient` – fields which will be skipped during object-relational mapping.

Example entity

```java
@Getter
@Setter
@NoArgsConstructor
@Entity
@Table(name = "users")
public class User {

    @Id
    private String login;

    @Column(name = "user_name")
    private String name;

    private String password;

    @Column(unique = true)
    private String email;

}
```

Generating primary key

The `@GeneratedValue` annotation `strategy` attribute defined how the primary key is generated:

- `GenerationType.IDENTITY`
  - MySQL: `AUTO_INCREMENT`,
  - PostgreSQL: `SERIAL`,
  - MS SQL: `IDENTITY(1,1)`;
- `GenerationType.SEQUENCE`:
  - value generated using database sequence,
  - np. Oracle Database: `CREATE SEQUENCE invoice_seq START WITH 1;`
- `GenerationType.TABLE`:
  - value generated using additional table;;
- `GenerationType.AUTO`:
  - selected by the JPA implementation.

Entity relationships

Database tables **relationships** can be mapped as **connection** betweeen entity classes:

- directional - one class contains reference to the second one,
- bidirectional- both classes contain reference to each other.

Annotations definiujące **relatioships**:

- `@OneToOne`,
- `@OneToMany`,
- `@ManyToOne`
- `@ManyToMany`.

Selected properties of annotations:

- `mappedBy` - marks field being owner of the relationship,
- `cascade` - cascade operations on elements being in relationship.

Example relationship

```java
@Getter
@Setter
@NoArgsConstructor
@Entity
@Table(name = "users")
public class User {

    @Id
    private String login;

    @Column(name = "user_name")
    private String name;

    private String password;

    @Column(unique = true)
    private String email;

    @OneToMany(mappedBy = "user")
    private List<Character> characters;

}
```

```java
@Getter
@Setter
@NoArgsConstructor
@Entity
@Table(name = "characters")
public class Character {

    @Id
    @GeneratedValue(
        strategy = GenerationType.TABLE)
    private Long id;

    private String name;

    @ManyToOne
    @JoinColumn(name ="user")
    private User user;

}
```

Connection configuration

There are several methods for defining connection to the database in Spring. One of the easiest is to user `application.properties` configuration file stored in project sources.

Data source settings:

```
spring.datasource.url=jdbc:h2:mem:simple-rpg
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=admin
spring.datasource.password=adminadmin
```

JPA settings:

```
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.generate-ddl=true
```

Hibernate specific settings:

```
spring.jpa.hibernate.ddl-auto=create-drop
```

H2 specific settings:

```
spring.h2.console.enabled=true
```

Using H2 database

H2 is small easy in use in-memory database.

Required dependency:

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

In order to use built-in web console http server must be enabled.

Repositories

Spring offers automatic data repositories creation based on interface implementation:

```
@Repository
public interface UserRepository extends JpaRepository<User, String> {

}
```

The `JpaRepository` is only one of a number of different repositories types which can be used.

Custom queries

On of a number of ways adding custom queries is by using derived query methods:

```
@Repository
public interface UserRepository extends JpaRepository<User, String> {

    Optional<User> findByLoginAndPassword(String login, String password);

}
```

Custom queries

In case preparing derived query method is too complicated, queries in JPQL can be used:

```
@Repository
public interface UserRepository extends JpaRepository<User, String> {

    @Query("select u from User u where u.login = :login and u.password = :password")
    Optional<User> find(@Param("login") String login, @Param("password") String password);

}
```

JPQL - basic queries

```java
@Entity
public class Product {
    @Id
    UUID id = UUID.randomUUID();
    String name;
    String description;
    Integer price;
    Integer amount;
}
```

`SELECT` query syntax:

```
select_statement ::= select_clause from_clause
                     [where_clause]
                     [group_by_clause]
                     [having_clause]
                     [orderby_clause]
```

JPQL - basic queries

Select all products:

```
SELECT p FROM Product p
```

Select all products with specified name:

```
SELECT p FROM Product p WHERE p.name = :name
```

JPQL - basic queries

Select products with name matching regular expression - `LIKE` operator:

```
SELECT p FROM Product p WHERE p.name LIKE :name
```

Select products with name containing `VT-x` :

```
SELECT p FROM Product p WHERE p.description LIKE '% VT-x %'
```

Ignore case:

```
SELECT p FROM Product p WHERE LOWER(p.name) LIKE LOWER('%Laptop%')
```

Get products with low stock sorted by name:

```
SELECT p FROM Product p WHERE p.amount < 5 ORDER BY p.name
```

Derived methods vs JPQL

Derived query methods and JPQL syntax:

keyword	derived method	JPQL
And	findByLastNameAndFirstName	where x.lastName = ?1 and x.firstName = ?2
Or	findByLastNameOrFirstName	where x.lastName = ?1 or x.firstName = ?2
Is,Equals	findByFirstName, findByFirstNameIs, findByFirstNameEquals	where x.firstName = ?1
Between	findByStartDateBetween	where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	where x.age <= ?1
GreaterThan	findByAgeGreaterThan	where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	where x.age >= ?1
After	findByStartDateAfter	where x.startDate > ?1
Before	findByStartDateBefore	where x.startDate < ?1
IsNull	findByAgeIsNull	where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	where x.age not null

Derived methods vs JPQL

Derived query methods and JPQL syntax:

keyword	derived method	JPQL
Like	findByFirstNameLike	where x.firstName like ?1
NotLike	findByFirstNameNotLike	where x.firstName not like ?1
StartingWith	findByFirstNameStartingWith	where x.firstName like ?1 (parameter bound with appended %)
EndingWith	findByFirstNameEndingWith	where x.firstName like ?1 (parameter bound with prepended %)
Containing	findByFirstNameContaining	where x.firstName like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastNameDesc	where x.age = ?1 order by x.lastName desc
Not	findByLastNameNot	where x.lastName <> ?1
In	findByAgeIn(Collection ages)	where x.age in ?1
NotIn	findByAgeNotIn(Collection ages)	where x.age not in ?1
True	findByActiveTrue()	where x.active = true
False	findByActiveFalse()	where x.active = false
IgnoreCase	findByFirstNameIgnoreCase	where UPPER(x.firstName) = UPPER(?1)
First,Top	queryFirst10ByLastName	where x.lastName asc limit ?

Helpful resources

- Baeldung, *Introduction to Spring Data JPA*, https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa.
- Baeldung, *JPA Tutorials*, https://www.baeldung.com/tag/jpa/.
- Baeldung, *Spring Data Tutorials*, https://www.baeldung.com/category/persistence/spring-persistence/spring-data/.
- *Java Persistence*, https://en.wikibooks.org/wiki/Java_Persistence.
- RedHat, *Hibernate ORC Documentation*, https://hibernate.org/orm/documentation/5.4/.