Making the Data Persistence Layer Cleaner with Repositories

Dan Geabunea
SENIOR SOFTWARE DEVELOPER

@romaniancoder www.romaniancoder.com

Overview

Discover Spring Mongo Repositories

Query Methods

Insert, Update, and Delete Documents

Demo: Implement Mongo Operations Using Repositories

What Are Repositories?

Spring (Mongo) Repository

Abstraction (interface) that significantly reduces the amount of boilerplate code needed to implement the data access layer.

Repository Properties

An injectable interface that can be used in Spring applications

Provide basic CRUD operations capabilities out of the box

Can be expanded with ease

Mongo Repository Interface

Repository<T, ID>

CrudRepository<T, ID>

save, saveAll, findAll, findById, existsById, count, deleteById, deleteAll

PagingAndSortingRepository<T, ID>

findAll(Sort s), findAll(Pageable p)

Spring Data

MongoRepository<T, ID>

insert(T o), insert(Iterable<T> list)

Mongo

Document

Airport

```
@Document
public class Airport{
    @Id String id;
    String country;
    int nbFlightsPerDay;
    //...
}
```

Repository

AirportRepository

```
@Repository
public interface AirportRepository extends
    MongoRepository<Airport, String> {

}
```

Usage

AirportController

```
List<Airport> airports = repository.findAll();


Airport a = new Airport("London", 400);

repository.insert(a);


repository.deleteAll();
```

Alright, but where is the actual implementation of this interface?

You

For each repository interface, Spring registers the technology specific factory bean to create the appropriate proxy implementations.

Query Methods

Query Methods

Declarative way to add functionality to a Spring repository.

Airport

```
@Document public class Airport {
    @Id String id;
    int flightsPerDay;
    String name;
    boolean closed;
    Location location;
}
public class Location {
    String city; String country; String email;
}
```

Add a New Query Method

```java
@Repository
public interface AirportRepository
    extends MongoRepository<Airtport, String> {

    List<Airport> findByFlightsPerDayGreaterThan(int value);
}

// Call the method, implementation will be taken care by the proxy
List<Airport> airports = repository.findByFlightsPerDayGreaterThan(200);
```

The Spring Framework knows how to create a proxy based on the query method signatures.

How to Build Query Methods

Return Type

Method Prefix *(findBy)*

Property Name

Filter(s)

Build Query Methods for Numeric Properties

```
List<Airport> findByFlightsPerDayBetween(int min, int max);

List<Airport> findByFlightsPerDayGreaterThan(int value)

List<Airport> findByFlightsPerDayGreaterThanEqual(int value);

List<Airport> findByFlightsPerDayLessThanEqual(int value);
```

Build Query Methods for String Properties

```
List<Airport> findByNameLike(String airportName);

List<Airport> findByNameNotNull();

List<Airport> findByNameNull();


Optional<Airport> findByName(String airportName);

Airport findByName(String airportName);
```

Build Query Methods for Boolean Properties

```
List<Airport> findByClosedTrue();

List<Airport> findByClosedFalse();
```

Build Complex Query Methods

```
// The ‘And’ operator can be used to combine multiple filters

List<Airport> findByClosedTrueAndFlightsPerDayGreaterThan(int minFlights);
```

Alright, but does this approach work for all cases? What about even more complex queries?

You

You can also make repository methods execute custom Mongo Query Language constructs.

Declaring Custom Queries

```
@Repository
public interface AirportRepository
    extends MongoRepository<Airtport, String> {

    @Query("{'location.city' : ?0}")
    List<Airport> findByCity(String city);     // method name not relevant

    @Query("{'flightsPerDay' : {$lte : 50}}")
    List<Airport> findSmallAirports();         // method name not relevant

}
```

Prefer standard query methods instead of @Query where possible. Use @Query just for more complex scenarios.

Create, Update, and Delete Documents

The repository interfaces take care of inserts, updates, and deletes.

Built-in Methods

insert	save
delete	deleteAll

An ‘Empty’ Repository

```
@Repository
public interface AirportRepository
    extends MongoRepository<Airport, String> {

}
```

```
Airport a = new Airport(“Madrid”, 250);

repository.insert(a);
```

Inserting a New Document

_id	name	flightsPerDay

```
Airport a = new Airport(“Madrid”, 250);

repository.insert(a);
```

Inserting a New Document

_id	name	flightsPerDay
3d7745b3-2589-4976-a1fa-9f49ed31a673	Madrid	250

```
Airport a1 = new Airport("Madrid", 250);
Airport a2 = new Airport("Valencia", 120);
List<Airport> airports = Arrays.asList(a1,a2);

repository.insert(airports);
```

Batch Insert

_id	name	flightsPerDay

```
Airport a1 = new Airport("Madrid", 250);
Airport a2 = new Airport("Valencia", 120);
List<Airport> airports = Arrays.asList(a1,a2);

repository.insert(airports);
```

Batch Insert

_id	name	flightsPerDay
3d7745b3-2589-...	Madrid	250
4976-a1fa-9f49-...	Valencia	120

```
// Find document first
Airport a = repository.findById("c9db4315-ca22-4e31-a7af-8ac930a34d77");
// Set new values
a.setFlightsPerDay(300);
// Update
repository.save(a);
```

Updating an Existing Document

_id	name	flightsPerDay
c9db4315-ca22-...	Madrid	250

```
// Find document first
Airport a = repository.findById("c9db4315-ca22-4e31-a7af-8ac930a34d77");
// Set new values
a.setFlightsPerDay(300);
// Update
repository.save(a);
```

Updating an Existing Document

_id	name	flightsPerDay
c9db4315-ca22-...	Madrid	300

How Save Works

Scans the collection and tries to find a document with a matching ID

If no document is found for the given ID, then Save acts like Insert. A new document is created with the provided ID.

If a document is found, then it is completely replaced with the provided one.

```
// Find document first
Airport a = repository.findById("c9db4315-ca22-4e31-a7af-8ac930a34d77");
// Delete it
repository.delete(a);
```

Deleting a Document

_id	name	flightsPerDay
c9db4315-ca22-...	Madrid	300
59a726f4-c571-...	Paris	420

```
// Find document first
Airport a = repository.findById("c9db4315-ca22-4e31-a7af-8ac930a34d77");
// Delete it
repository.delete(a);
```

Deleting a Document

_id	name	flightsPerDay
59a726f4-c571-...	Paris	420

```
// Delete by ID

Airport a = repository.deleteById("59a726f4-c571-421d-a587-8b0b28fd29d1");
```

Deleting a Document

_id	name	flightsPerDay
59a726f4-c571-...	Paris	420

```
// Delete by ID
Airport a = repository.deleteById("59a726f4-c571-421d-a587-8b0b28fd29d1");
```

Deleting a Document

_id	name	flightsPerDay

```
repository.deleteAll();
```

Deleting All Documents

_id	name	flightsPerDay
c9db4315-ca22-...	Madrid	300
59a726f4-c571-...	Paris	420

```
repository.deleteAll();
```

Deleting All Documents

_id	name	flightsPerDay

Demo

Implement Mongo Operations Using Repositories

- Queries
- Inserts
- Updates
- Deletes

Summary

Repositories are great at abstracting the persistence details

Similar syntax with relational database counterparts (JPA Repository)

Create query methods using conventions

Discovered how to execute inserts, updates and deletions using repositories

Repositories vs. Mongo Template

Repositories

Great at abstracting the persistence layer. Improved type safety and cleaner code. Not suitable for complex queries or projections.

Mongo Template

More flexible. Can tackle any database operations. But they rely on strings and are more error prone. However, they allow us to access low level database APIs.

What Component to Use?

Spring Mongo Repositories

Great for inserting and deleting data

Easy to use for most queries, especially when you can use query methods

Works great in 80%-90% of use cases

Mongo Template

Exhaustive support for batch updates and partial updates

Can built extremely complex queries

Can access low level database APIs