(/)

Spring Data Reactive Repositories with MongoDB

Last updated: October 9, 2021

Written by: baeldung (https://www.baeldung.com/author/baeldung)

NoSQL (https://www.baeldung.com/category/persistence/nosql)

Spring Data (https://www.baeldung.com/category/persistence/spring-persistence/spring-data)

MongoDB (https://www.baeldung.com/tag/mongodb)

1. Introduction

In this tutorial, we're going to see how to configure and implement database operations using Reactive Programming through Spring Data Reactive Repositories with MongoDB.

We'll go over the basic usages of *ReactiveCrudRepository, ReactiveMongoRepository,* as well as *ReactiveMongoTemplate.*

Even though these implementations use reactive programming (/spring-5-functional-web), that isn't the primary focus of this tutorial.

2. Environment

In order to use Reactive MongoDB, we need to add the dependency to our *pom.xml.*

We'll also add an embedded MongoDB for testing:

```
<dependencies>
    // ...
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
    </dependency>
    <dependency>
        <groupId>de.flapdoodle.embed</groupId>
        <artifactId>de.flapdoodle.embed.mongo</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

3. Configuration

In order to activate the reactive support, we need to use the *@EnableReactiveMongoRepositories* alongside with some infrastructure setup:

```
@EnableReactiveMongoRepositories
public class MongoReactiveApplication
  extends AbstractReactiveMongoConfiguration {

    @Bean
    public MongoClient mongoClient() {
        return MongoClients.create();
    }

    @Override
    protected String getDatabaseName() {
        return "reactive";
    }
}
```

Note that the above would be necessary if we were using the standalone MongoDB installation. But, as we're using Spring Boot with embedded MongoDB in our example, the above configuration is not necessary.

4. Creating a *Document*

For the examples below, let's create an *Account* class and annotate it with *@Document* to use it in the database operations:

```
@Document
public class Account {

    @Id
    private String id;
    private String owner;
    private Double value;

    // getters and setters
}
```

5. Using Reactive Repositories

We are already familiar with the repositories programming model (/spring-data-mongodb-tutorial), with the CRUD methods already defined plus support for some other common things as well.

Now with the Reactive model, we get the same set of methods and specifications, except that we'll deal with the results and parameters in a reactive way.

5.1. *ReactiveCrudRepository*

We can use this repository the same way as the blocking *CrudRepository*:

```
@Repository
public interface AccountCrudRepository
  extends ReactiveCrudRepository<Account, String> {

    Flux<Account> findAllByValue(String value);
    Mono<Account> findFirstByOwner(Mono<String> owner);
}
```

We can pass different types of arguments like plain (*String*), wrapped (*Optional*, *Stream*), or reactive (*Mono*, *Flux*) as we can see in the *findFirstByOwner()* method.

5.2. *ReactiveMongoRepository*

There's also the *ReactiveMongoRepository* interface, which inherits from *ReactiveCrudRepository* and adds some new query methods:

```
@Repository
public interface AccountReactiveRepository
  extends ReactiveMongoRepository<Account, String> { }
```

Using the *ReactiveMongoRepository*, we can query by example:

```
Flux<Account> accountFlux = repository
  .findAll(Example.of(new Account(null, "owner", null)));
```

As a result, we'll get every *Account* that is the same as the example passed.

With our repositories created, they already have defined methods to perform some database operations that we don't need to implement:

```
Mono<Account> accountMono
  = repository.save(new Account(null, "owner", 12.3));
Mono<Account> accountMono2 = repository
  .findById("123456");
```

5.3. *RxJava2CrudRepository*

With *RxJava2CrudRepository*, we have the same behavior as the *ReactiveCrudRepository*, but with the results and parameter types from *RxJava*:

```
@Repository
public interface AccountRxJavaRepository
  extends RxJava2CrudRepository<Account, String> {

    Observable<Account> findAllByValue(Double value);
    Single<Account> findFirstByOwner(Single<String> owner);
}
```

5.4. Testing Our Basic Operations

In order to test our repository methods, we'll use the test subscriber:

```
@Test                          (/)
public void givenValue_whenFindAllByValue_thenFindAccount() {
    repository.save(new Account(null, "Bill", 12.3)).block();
    Flux<Account> accountFlux = repository.findAllByValue(12.3);

    StepVerifier
      .create(accountFlux)
      .assertNext(account -> {
          assertEquals("Bill", account.getOwner());
          assertEquals(Double.valueOf(12.3) , account.getValue());
          assertNotNull(account.getId());
      })
      .expectComplete()
      .verify();
}

@Test
public void givenOwner_whenFindFirstByOwner_thenFindAccount() {
    repository.save(new Account(null, "Bill", 12.3)).block();
    Mono<Account> accountMono = repository
      .findFirstByOwner(Mono.just("Bill"));

    StepVerifier
      .create(accountMono)
      .assertNext(account -> {
          assertEquals("Bill", account.getOwner());
          assertEquals(Double.valueOf(12.3) , account.getValue());
          assertNotNull(account.getId());
      })
      .expectComplete()
      .verify();
}

@Test
public void givenAccount_whenSave_thenSaveAccount() {
    Mono<Account> accountMono = repository.save(new Account(null, "Bill", 12.3));

    StepVerifier
      .create(accountMono)
      .assertNext(account -> assertNotNull(account.getId()))
      .expectComplete()
      .verify();
}
```

6. *ReactiveMongoTemplate*

Besides the repositories approach, we have the *ReactiveMongoTemplate*.

First of all, we need to register *ReactiveMongoTemplate* as a bean:

```
@Configuration
public class ReactiveMongoConfig {

    @Autowired
    MongoClient mongoClient;

    @Bean
    public ReactiveMongoTemplate reactiveMongoTemplate() {
        return new ReactiveMongoTemplate(mongoClient, "test");
    }
}
```

And then, we can inject this bean into our service to perform the database operations:

```
@Service (/)
public class AccountTemplateOperations {

    @Autowired
    ReactiveMongoTemplate template;

    public Mono<Account> findById(String id) {
        return template.findById(id, Account.class);
    }

    public Flux<Account> findAll() {
        return template.findAll(Account.class);
    }
    public Mono<Account> save(Mono<Account> account) {
        return template.save(account);
    }
}
```

ReactiveMongoTemplate also has a number of methods that do not relate to the domain we have, you can check them out in the documentation (https://docs.spring.io/spring-data/mongodb/docs/current/api/org/springframework/data/mongodb/core/ReactiveMongoTemplate.html).

7. Conclusion

In this brief tutorial, we've covered the use of repositories and templates using reactive programming with MongoDB with Spring Data Reactive Repositories framework.

The full source code for the examples is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/persistence-modules/spring-data-mongodb-reactive).

Comments are closed on this article!

(/)

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)
PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)