Using Spring Data Reactive

Zoltan Altfatter

Reactive Spring Data modules

- MongoDB
- Apache Cassandra
- Redis
- Couchbase

Neo4j (possible in the future)

Solr (possible in the future)

Elasticsearch (possible in the future)

Reactive Spring Data

- Reactive Template API
- Reactive Repository support
- reduced feature set

Reactive MongoDB

Reactive MongoDB

Starter for using MongoDB and Spring Data MongoDB Reactive

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
</dependency>
```

Starter for using MongoDB and Spring Data MongoDB

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

Spring Boot Reactive MongoDB starter

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver</artifactId>
</dependency>
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-async</artifactId>
</dependency>
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-reactivestreams</artifactId>
</dependency>
<dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-core</artifactId>
</dependency>
```

ReactiveMongoOperations

```
public interface ReactiveMongoOperations extends ReactiveFluentMongoOperations {

  ReactiveIndexOperations indexOps(String collectionName);

  Mono<Document> executeCommand(String jsonCommand);

  <T> Flux<T> execute(ReactiveDatabaseCallback<T> action);

  <T> Flux<T> findAll(Class<T> entityClass);

  <T> Mono<T> findOne(Query query, Class<T> entityClass);

  Mono<Boolean> exists(Query query, Class<?> entityClass);

  <T> Mono<T> findById(Object id, Class<T> entityClass);

  ...
}
```

ReactiveMongoTemplate

- Implements **ReactiveMongoOperations**
- Methods like: **`find, findAndModify, findOne, insert, remove, save, update, and updateMulti`**
- exception translation of exceptions thrown in the MongoDB Java driver into Spring's portable Data Access Exception hierarchy

Instantiating ReactiveMongoTemplate

```
@Configuration
public class AppConfig {

    public @Bean
    MongoClient reactiveMongoClient() {
        return MongoClients.create("mongodb://localhost");
    }

    public @Bean ReactiveMongoTemplate reactiveMongoTemplate() {
        return new ReactiveMongoTemplate(reactiveMongoClient(), "mydatabase");
    }
}
```

Reactive MongoDB Repositories

- ReactiveCrudRepository
- ReactiveSortingRepository
- RxJava2CrudRepository
- RxJava2SortingRepository

```
public interface ReactiveSortingRepository<T, ID> extends ReactiveCrudRepository<T, ID> {
   Flux<T> findAll(Sort sort);
}
```

Sample Measurement entity

```
@Document
public class Measurement {

    @Id
    private String id;

    private String sensorName;

    private BigDecimal temperature;

    private LocalDateTime time;

}
```

Sample MongoDB Repository

```
public interface MeasurementRepository extends ReactiveMongoRepository<Measurement, String> {

   // - For capped collections, you can use a Tailable Cursor that remains open after the client
consumes all initially returned data.
   // - Using tailable cursors with a reactive data types allows construction of infinite streams.
   // - A tailable cursor remains open until it is closed externally.
   // - It emits data as new documents arrive in a capped collection.

   @Tailable
   Flux<Measurement> findAllByTimeGreaterThan(LocalDateTime time);


   Flux<Measurement> findBySensorName(String name);

   Flux<Measurement> findBySensorNameOrderByTemperature(String name);

   Mono<Measurement> findFirstBySensorName(String name);

}
```

Reactive Redis

Reactive Redis

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis-reactive</artifactId>
</dependency>

------->

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>

------->

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>io.lettuce</groupId>
   <artifactId>lettuce-core</artifactId>
</dependency>
```

Reactive Redis support

- Spring Data Redis integrates with **Lettuce** as the only reactive Java connector
- Project Reactor is used as reactive composition library
- **`ReactiveRedisConnection`**
  - handles the communication with the Redis back-end
  - translates the underlying driver exceptions to Spring's consistent DAO exception hierarchy
- **`ReactiveRedisConnectionFactory`**
  - creates active ReactiveRedisConnection instances.
  - depending on the underlying configuration, the factory can return a new connection or an existing connection

ReactiveRedisConnectionFactory

```
@Configuration
class RedisConfig {

   // easy setup
   @Bean
   public ReactiveRedisConnectionFactory connectionFactory() {
       return new LettuceConnectionFactory("localhost", 6379);
   }

   // more sophisticated configuration, including SSL and timeouts
   @Bean
   public ReactiveRedisConnectionFactory lettuceConnectionFactory() {

       LettuceClientConfiguration clientConfig = LettuceClientConfiguration.builder()
               .useSsl().and()
               .commandTimeout(Duration.ofSeconds(2))
               .shutdownTimeout(Duration.ZERO)
               .build();

       return new LettuceConnectionFactory(new RedisStandaloneConfiguration("localhost", 6379), clientConfig);
   }
}
```

ReactiveRedisTemplate

- The template offers a high-level abstraction for Redis interactions.
- Takes care of the serialization and connection management.
- provides operation views
  - `ReactiveHashOperations`
  - `ReactiveSetOperations`
  - `ReactiveListOperations`
  - `ReactiveValueOperations`
- uses a Java-based serializer for most of its operations

```
public interface RedisSerializer<T> {

   byte[] serialize(@Nullable T t) throws SerializationException;

   T deserialize(@Nullable byte[] bytes) throws SerializationException;
}
```

ReactiveRedisTemplate

```
@Bean
public ReactiveRedisTemplate<String, Customer> reactiveJsonCustomerRedisTemplate(
       ReactiveRedisConnectionFactory connectionFactory) {

   RedisSerializationContext<String, Customer> serializationContext =
RedisSerializationContext
           .<String, Customer>newSerializationContext(new StringRedisSerializer())
           .hashKey(new StringRedisSerializer())
           .hashValue(new Jackson2JsonRedisSerializer<>(Customer.class))
           .build();

   return new ReactiveRedisTemplate<>(connectionFactory, serializationContext);
}
```

Spring Boot with Spring Data Redis

- RedisProperties (`spring.redis.*`)
- RedisRectiveAutoConfiguration -> configures a ReactiveRedisTemplate
- RedisRepositoriesAutoConfiguration -> auto configures Redis Repositories
- **No Reactive Redis Repositories support**

Reactive relation database

Reactive relational database support

- JDBC is a blocking API
- Two experiments:
  - **ADBA** (Asynchronous Database Access)
    - Driven by Oracle
    - Uses **CompletableFuture**
  - **R2DBC** (Reactive Relational Database Connectivity Driver)
    - Driven by Pivotal
    - Current support for PostgreSQL

R2DBC – Reactive Relational Database Connectivity

- Design principles:
  - Utilize Reactive Streams Types and Patterns.
  - Be completely non-blocking, all the way to the database.
  - Shrink the driver SPI to the minimal set of operations that are implementation specific, regardless of usability.
  - Enable multiple "humane" APIs to be built on top of the driver SPI.
- R2DBC is an experimental playground, not for production (yet)
- PostgreSQL implementation
- Ben Hale from SpringOne 2018
  https://www.youtube.com/watch?v=idApf9DMdfk

R2DBC example

https://github.com/altfatterz/spring-data-r2dbc-sample