Spring Data Elasticsearch CRUD Examples Using Spring Boot – Part 1

Description

Introduction

Are you looking for an easy step by step guide which shows how to perform CRUD operations in Elasticsearch? Then you are at the right website. By the end of this post you will learn how to perform CRUD (Create, Read, Update, Delete) operations in Elasticsearch without any doubts.

There are two ways how our application can talk to Elasticsearch.

1. ElasticsearchRepository
2. ElasticsearchRestTemplate

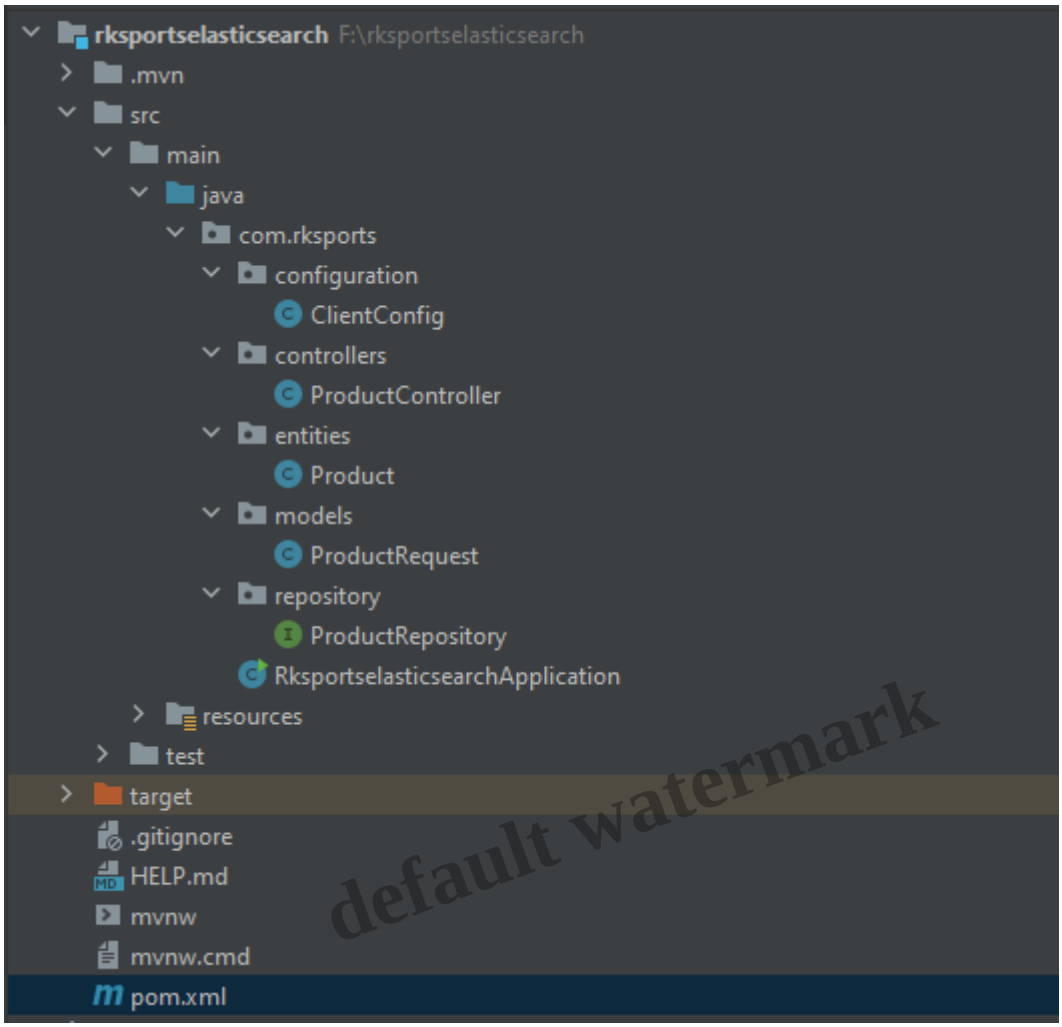
We will restrict this post to learning the first method i.e. ElasticsearchRepository. If you have used Spring Data JPA with other databases, then you must be familiar with JpaRepository or CrudRepository. Basically you will extend your custom repository interface with these Spring provided interfaces, and then you will auto wire your interface inside your service layer class and call methods like save(), saveAll(), findById..etc to perform database operations. Just by using those methods, database specific queries will get generated. In the similar way Elasticsearch queries will also get generated automatically just by calling these repository methods.

In this tutorial, we are going to create an application which will perform CRUD operations around product details.

Technologies used

1. Spring Boot version – 2.5.5
2. Java version – 11
3. Intellij Idea IDEA

Product Directory Structure



pom.xml

Below are the dependencies we have added for our application. **ModelMapper** and **Lombok** are the addons which will help reduce the boilerplate code. Using **Modemapper** we can directly map and set the data from one object to other instead of writing getter setter for each model attribute. **Lombok** is used to annotate our model classes to avoid writing getters and setters.

```
<dependencies>
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
 </dependency>
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
 </dependency>
 <dependency>
  <groupId>org.modelmapper</groupId>
```

```
   <artifactId>modelmapper</artifactId>
   <version>2.3.5</version>
  </dependency>
  <dependency>
   <groupId>org.projectlombok</groupId>
   <artifactId>lombok</artifactId>
   <optional>true</optional>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-test</artifactId>
   <scope>test</scope>
  </dependency>
 </dependencies>
```

Connecting to Elasticsearch

Spring Data uses **Java High Level REST Client** in order to connect with Elasticsearch. Below code snippet serves the purpose. Adding Socket timeout to the client configuration is optional. If you are facing timeout while connecting then you can add this timeout.

@EnableElasticsearchRepositories annotation is used to enable Elasticsearch repositories and scan the package of the annotated configuration class for Spring Data repositories by default.


```
package com.rksports.configuration;

import org.apache.http.client.config.RequestConfig;
import org.elasticsearch.client.RestHighLevelClient;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.elasticsearch.client.ClientConfiguration;
import org.springframework.data.elasticsearch.client.RestClients;
import org.springframework.data.elasticsearch.config.AbstractElasticsearchConf
import org.springframework.data.elasticsearch.repository.config.EnableElastics

@Configuration
@EnableElasticsearchRepositories(basePackages = "com.rksports")
public class ClientConfig extends AbstractElasticsearchConfiguration {

    @Override
    @Bean
    public RestHighLevelClient elasticsearchClient() {
        final ClientConfiguration clientConfiguration =
                ClientConfiguration
                .builder()
                .connectedTo("localhost:9200")
                .withSocketTimeout(30000)
                .build();
        return RestClients.create(clientConfiguration).rest();
    }
```

```
}
```

Entity Class (Elasticsearch document)

We are going to prepare a Product entity class against which we will perform the CRUD operations like searchBydId , searchByName etc. We will annotate our class with Elasticsearch specific annotations.

```
package com.rksports.entities;

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.elasticsearch.annotations.Document;
import org.springframework.data.elasticsearch.annotations.Field;
import org.springframework.data.elasticsearch.annotations.FieldType;
import java.math.BigDecimal;

@Document(indexName="productindex")
@Data
public class Product {

    @Id
    private String id;
    @Field(type= FieldType.Text, name="productName")
    private String productName;

    @Field(type= FieldType.Text, name="productDescription")
    private String productDescription;

    @Field(type= FieldType.Double, name="productPrice")
    private BigDecimal productPrice;

    @Field(type= FieldType.Integer, name="quantity")
    private Integer quantity;

    @Field(type= FieldType.Text, name="sportsCategory")
    private String sportsCategory;

    @Field(type= FieldType.Text, name="manufacturer")
    private String manufacturer;
}
```


@Document – We use this annotation annotate above the entity class that has to be persisted to Elasticsearch. We use the indexName as the key to specify which index the document has to be saved.
@Field – This annotation is used above the class attributes. It will map the attribute with the key (column in the RDBMS terms) in the Elasticsearch document. FieldType is used to specify the type of Field. All the possible list of FieldTypes can be found in
https://www.elastic.co/guide/en/elasticsearch/reference/7.15/mapping-types.html

Request Object

Request object where you will get the data from the caller and then we will map it against our entity class.

```
package com.rksports.models;
import lombok.Data;
import java.math.BigDecimal;

@Data
public class ProductRequest {

    private String id;
    private String productName;
    private String productDescription;
    private BigDecimal productPrice;
    private Integer quantity;
    private String sportsCategory;
    private String manufacturer;
}
```


Repository Layer

The repository interface will allow us to write the method names and the queries will automatically get generated under the hood. For example findById, findByProductName will enable you to fetch the document which matches the Id and Name. find(), findAll(), save(), saveAll() are the default methods that are inherited to ProductRepository interface.

```
package com.rksports.repository;

import com.rksports.entities.Product;
import org.springframework.data.elasticsearch.repository.ElasticsearchReposito

public interface ProductRepository extends ElasticsearchRepository<Product, St
}
```

Controller

We will create GET http method to get the list of all products or a specific product. To create a new product we will use a POST method.

```
package com.rksports.controllers;
```

```
import com.rksports.entities.Product;
import com.rksports.models.ProductRequest;
import com.rksports.repository.ProductRepository;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class ProductController {

    @Autowired
    ProductRepository productRepository;

    @PostMapping("v1/product")
    public Product createProduct(@RequestBody ProductRequest productRequest) {

        ModelMapper modelMapper = new ModelMapper();
        Product product = modelMapper.map(productRequest, Product.class);
        return productRepository.save(product);
    }

    @GetMapping("v1/product/{id}")
    public Product getProduct(@PathVariable("id") String productId) {
       return productRepository.findById(productId).get();
    }

    @GetMapping("v1/product")
    public Iterable<Product> getAllProduct(@PathVariable("id") String productI
        return productRepository.findAll();
    }
}
```


Wrapping up

With that being said, we have come to an end of this basic CRUD operations Elasticsearch blog post. If you have any suggestions or doubts feel free to comment and I will be happy to answer. Keep learning ðŸ™‚

Category

1. Elasticsearch

Tags

1. Elasticsearch

Date Created
November 2, 2021
Author
kk-ravi144gmail-com