Spring Data Elasticsearch CRUD Examples Using Spring Boot â€“ Part 2

Description

Introduction

In our previous post we saw how to perform CRUD operations using Spring Data repository. There is another way we can interact with Elasticsearch and that is what we are going to discuss in the blog post. If you have worked on Spring Data JPA with databases like PostgreSQL or Oracle, there will be a point where you will prefer writing SQL native queries inside your repository class for more complex business requirements. In the similar way here we have the option of using ElasticsearchRestTemplate. Using ElasticsearchRestTemplate we have more control and flexibility over our queries. Sometimes you can directly use Elasticsearch queries if you are proficient on it.

There are 3 types of Query(Interface) implementations. In this post we are going to implement only NativeSearchQuery.

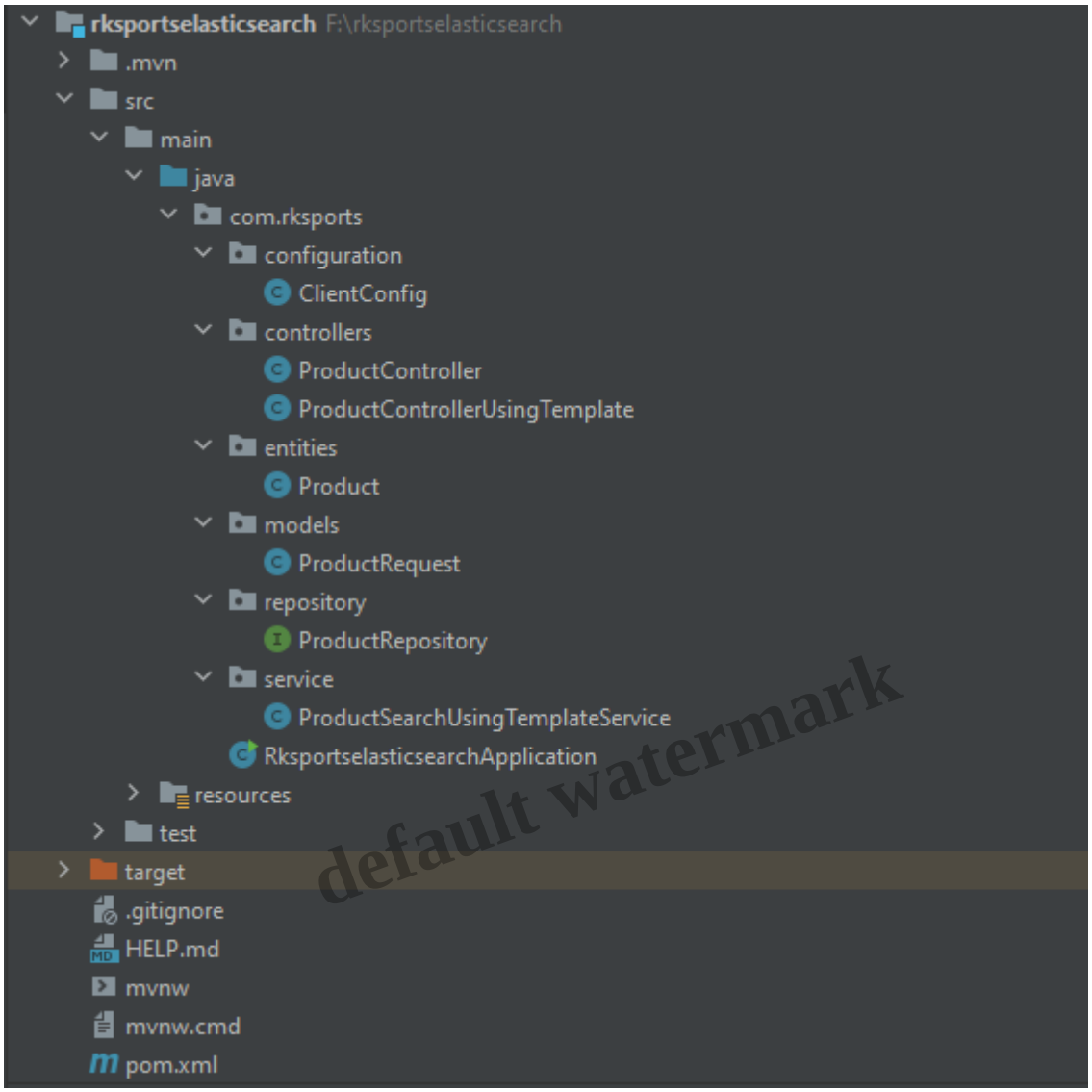
- CriteriaQuery
- StringQuery
- NativeSearchQuery

In this tutorial, we are going to create an application which will perform CRUD operations around product details. By the end of this post you will learn how to perform CRUD (Create, Read, Update, Delete) operations in Elasticsearch without any doubts.

Technologies used

Spring Boot version â€“ 2.5.5
Java version â€“ 11
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

```
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 â€“ 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 â€“ 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;
}
```


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.service.ProductSearchUsingTemplateService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class ProductControllerUsingTemplate {
```

```
    @Autowired
    ProductSearchUsingTemplateService productService;

    @PostMapping("v2/product")
    public void createProduct(@RequestBody ProductRequest productRequest) {
        productService.createProductIndex(productRequest);
    }

    @GetMapping("v2/product/manufacturer/{manufacturer}")
    public List<Product> findProductByBrand(@PathVariable("manufacturer") Stri
        return productService.findProductByManufacturer(manufacturer);
    }
}
```

Service layer


```
package com.rksports.service;

import com.rksports.entities.Product;
import com.rksports.models.ProductRequest;
import org.elasticsearch.index.query.QueryBuilder;
import org.elasticsearch.index.query.QueryBuilders;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.elasticsearch.core.ElasticsearchOperations;
import org.springframework.data.elasticsearch.core.SearchHits;
import org.springframework.data.elasticsearch.core.mapping.IndexCoordinates;
import org.springframework.data.elasticsearch.core.query.IndexQuery;
import org.springframework.data.elasticsearch.core.query.IndexQueryBuilder;
import org.springframework.data.elasticsearch.core.query.NativeSearchQueryBuil
import org.springframework.data.elasticsearch.core.query.Query;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class ProductSearchUsingTemplateService {

    @Autowired
    ElasticsearchOperations elasticsearchOperations;

    public String createProductIndex(ProductRequest productRequest) {

        ModelMapper modelMapper = new ModelMapper();
        Product product = modelMapper.map(productRequest, Product.class);

        IndexQuery indexQuery = new IndexQueryBuilder()
                .withId(product.getId())
                .withObject(product).build();
```

```
        String documentId = elasticsearchOperations.index(indexQuery, IndexCoo
        return documentId;
    }

    public List<Product> findProductByManufacturer(final String manufacturer)

        QueryBuilder queryBulider = QueryBuilders.matchQuery("manufacturer",ma

        Query searchQuery = new NativeSearchQueryBuilder()
                .withQuery(queryBulider)
                .build();

        SearchHits<Product> productHits = elasticsearchOperations.search(searc
                Product.class,
                IndexCoordinates.of("productindex"));

        List<Product> productList = new ArrayList<Product>();
        productHits.forEach( productHit -> {
            productList.add(productHit.getContent());
        });
        return productList;
    }
}
```


ElasticsearchRestTemplate implements **ElasticsearchOperations** interface which is actually responsible for performing the operations.

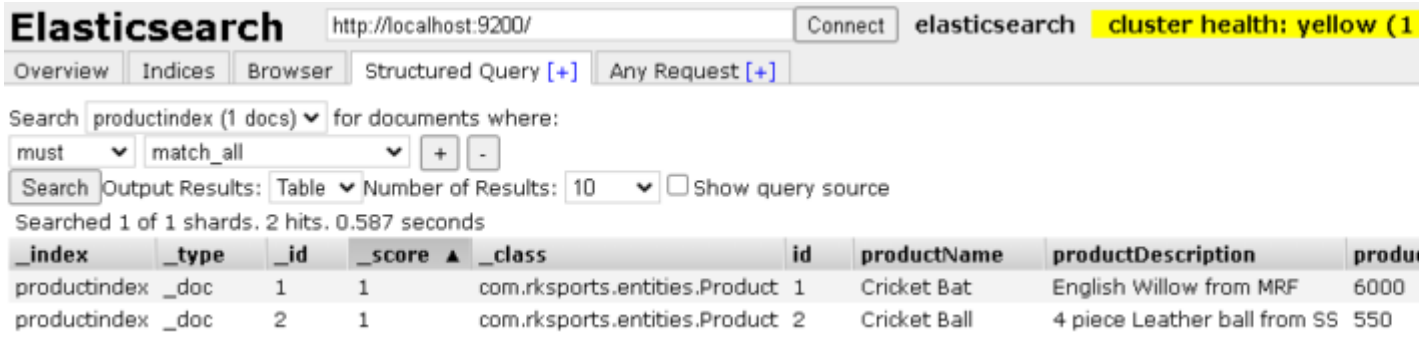
Create Product Operation
In the first method, we are using IndexQuery to construct the document we want to save. Using the IndexQueryBuilder we first add the Id and the the actual object. Then later we use index() method to add a document to the index.

Search Product Operation
Firstly we will write a QueryBuilder based on which field we want to search. Out of the 3 types of Query discussed above, we are going to use NativeSearchQueryBuilder

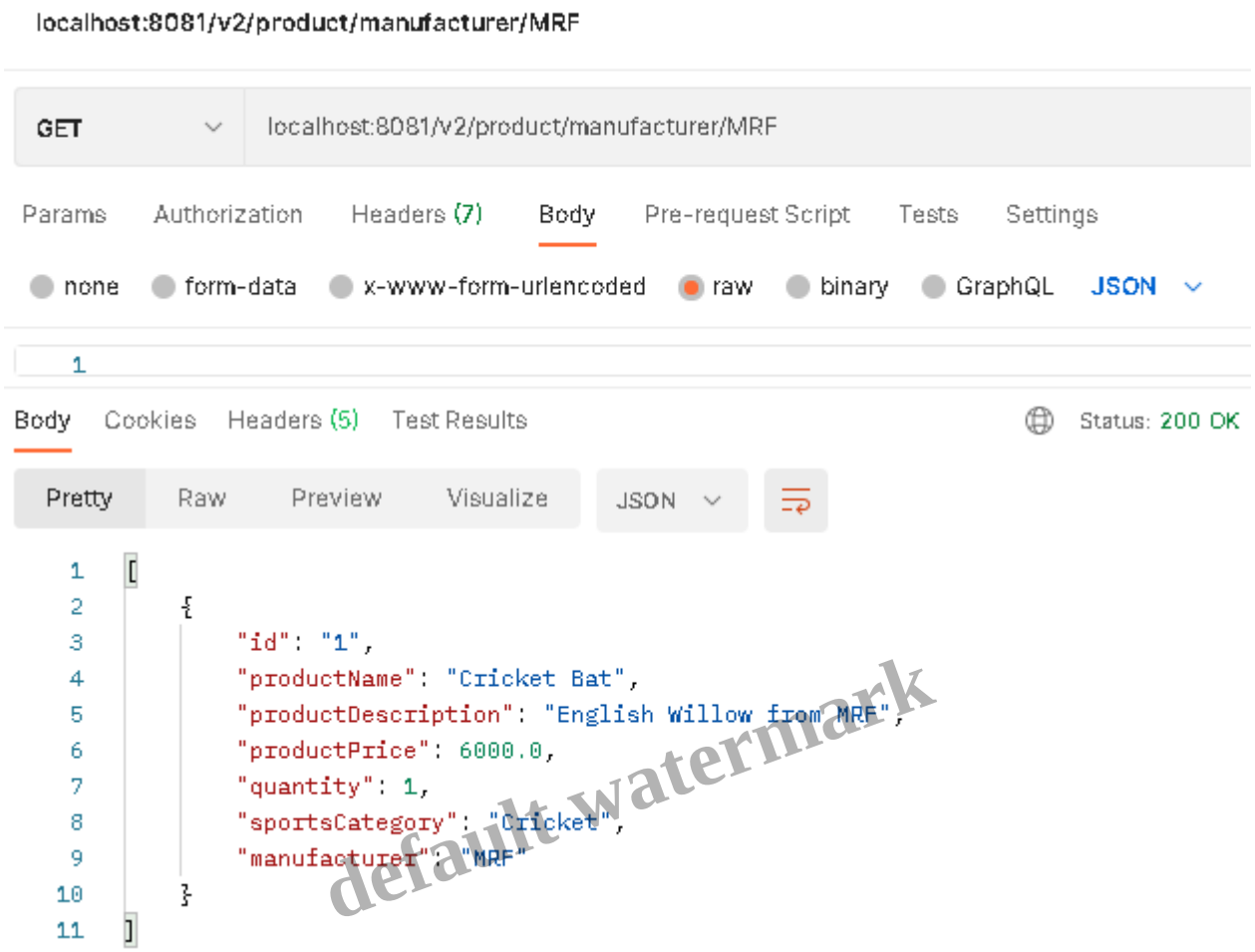
Its time to test our app now ðŸ™‚,

Create Product using postman

Product created successfully



Get Product



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

Date Created
November 6, 2021
Author
kk-ravi144gmail-com