此 demo 主要演示了 Spring Boot 如何集成 elasticsearch-rest-high-level-client 完成对 ElasticSearch 6.5.4 版本的基本 CURD 操作。

参考：https://github.com/xkcoding/spring-boot-demo/tree/master/spring-boot-demo-elasticsearch-rest-high-level-client

ElasticSearch 官方文档：https://www.elastic.co/guide/en/elasticsearch/reference/index.html

Java High Level REST Client：https://www.elastic.co/guide/en/elasticsearch/client/java-rest/index.html

注意：本demo中，ElasticSearch版本为 6.5.4。而spring-boot-demo-elasticsearch-rest-high-level-client是对 ElasticSearch 7.x 版本的基本 CURD 操作。

这里我对demo做了部分修改（添加了类型和映射），并且添加了一些测试方法。

elasticsearch 5.x 6.x 7.x的主要区别：

```
# ElasticSearch5.X 一个索引index可以有多个类型type
# ElasticSearch6.X 一个索引index只有一个类型type
# ElasticSearch7.X 没有了类型type的概念，去除了类型
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>spring-boot-demo</artifactId>
    <groupId>com.xkcoding</groupId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>

  <artifactId>spring-boot-demo-elasticsearch-rest-high-level-client</artifactId>
  <name>spring-boot-demo-elasticsearch-rest-high-level-client</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>

    <dependency>
```

```
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- test -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <!-- validator -->
    <dependency>
      <groupId>org.hibernate.validator</groupId>
      <artifactId>hibernate-validator</artifactId>
      <scope>compile</scope>
    </dependency>

    <!--
        You can easily generate your own configuration metadata file from items
annotated with
        @ConfigurationProperties by using the spring-boot-configuration-
processor jar.
        -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-configuration-processor</artifactId>
    </dependency>

    <!-- 工具类 -->
    <dependency>
      <groupId>cn.hutool</groupId>
      <artifactId>hutool-all</artifactId>
    </dependency>

    <!-- elasticsearch -->
    <dependency>
      <groupId>org.elasticsearch</groupId>
      <artifactId>elasticsearch</artifactId>
      <version>6.5.4</version>
    </dependency>

    <!-- elasticsearch-rest-client -->
    <dependency>
      <groupId>org.elasticsearch.client</groupId>
      <artifactId>elasticsearch-rest-client</artifactId>
      <version>6.5.4</version>
    </dependency>

    <!-- elasticsearch-rest-high-level-client -->
    <dependency>
      <groupId>org.elasticsearch.client</groupId>
      <artifactId>elasticsearch-rest-high-level-client</artifactId>
      <version>6.5.4</version>
      <exclusions>
```

```
        <exclusion>
          <groupId>org.elasticsearch.client</groupId>
          <artifactId>elasticsearch-rest-client</artifactId>
        </exclusion>
        <exclusion>
          <groupId>org.elasticsearch</groupId>
          <artifactId>elasticsearch</artifactId>
        </exclusion>
      </exclusions>
    </dependency>

    <!-- lombok -->
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>

  </dependencies>

  <build>
    <finalName>spring-boot-demo-elasticsearch-rest-high-level-client</finalName>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

</project>
```

application.yml

```
demo:
  data:
    elasticsearch:
      cluster-name: ahhs6.5.4
      cluster-nodes: 192.168.1.128:9200,192.168.1.129:9200,192.168.1.130:9200
      index:
        number-of-replicas: 0
        number-of-shards: 3
```

config配置类

ElasticsearchProperties.java

```
package com.xkcoding.elasticsearch.config;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import javax.validation.constraints.NotNull;
import java.util.ArrayList;
import java.util.List;

/**
 * ElasticsearchProperties
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/9/15 22:58
 */
@Data
@Builder
@Component
@NoArgsConstructor
@AllArgsConstructor
@ConfigurationProperties(prefix = "demo.data.elasticsearch")
public class ElasticsearchProperties {

    /**
     * 请求协议
     */
    private String schema = "http";

    /**
     * 集群名称
     */
    private String clusterName = "elasticsearch";

    /**
     * 集群节点
     */
    @NotNull(message = "集群节点不允许为空")
    private List<String> clusterNodes = new ArrayList<>();

    /**
     * 连接超时时间(毫秒)
     */
    private Integer connectTimeout = 1000;

    /**
     * socket 超时时间
```

```
     */
    private Integer socketTimeout = 30000;

    /**
     * 连接请求超时时间
     */
    private Integer connectionRequestTimeout = 500;

    /**
     * 每个路由的最大连接数量
     */
    private Integer maxConnectPerRoute = 10;

    /**
     * 最大连接总数量
     */
    private Integer maxConnectTotal = 30;

    /**
     * 索引配置信息
     */
    private Index index = new Index();

    /**
     * 认证账户
     */
    private Account account = new Account();

    /**
     * 索引配置信息
     */
    @Data
    public static class Index {

        /**
         * 分片数量
         */
        private Integer numberOfShards = 3;

        /**
         * 副本数量
         */
        private Integer numberOfReplicas = 2;

    }

    /**
     * 认证账户
     */
    @Data
    public static class Account {

        /**
         * 认证用户
         */
        private String username;

        /**
```

```
         * 认证密码
         */
        private String password;

    }

}
```

ElasticsearchAutoConfiguration.java

```
package com.xkcoding.elasticsearch.config;

import lombok.RequiredArgsConstructor;
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.util.Assert;
import org.springframework.util.StringUtils;

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

/**
 * ElasticsearchAutoConfiguration
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/9/15 22:59
 */
@Configuration
@RequiredArgsConstructor(onConstructor_ = @Autowired)
@EnableConfigurationProperties(ElasticsearchProperties.class)
public class ElasticsearchAutoConfiguration {

    private final ElasticsearchProperties elasticsearchProperties;

    private List<HttpHost> httpHosts = new ArrayList<>();

    @Bean
    @ConditionalOnMissingBean
```

```
    public RestHighLevelClient restHighLevelClient() {

        List<String> clusterNodes = elasticsearchProperties.getClusterNodes();
        clusterNodes.forEach(node -> {
            try {
                String[] parts = StringUtils.split(node, ":");
                Assert.notNull(parts, "Must defined");
                Assert.state(parts.length == 2, "Must be defined as
'host:port'");
                httpHosts.add(new HttpHost(parts[0], Integer.parseInt(parts[1]),
elasticsearchProperties.getSchema()));
            } catch (Exception e) {
                throw new IllegalStateException("Invalid ES nodes " + "property
'" + node + "'", e);
            }
        });
        RestClientBuilder builder = RestClient.builder(httpHosts.toArray(new
HttpHost[0]));

        return getRestHighLevelClient(builder, elasticsearchProperties);
    }


    /**
     * get restHistLevelClient
     *
     * @param builder                RestClientBuilder
     * @param elasticsearchProperties elasticsearch default properties
     * @return {@link org.elasticsearch.client.RestHighLevelClient}
     * @author fxbin
     */
    private static RestHighLevelClient getRestHighLevelClient(RestClientBuilder
builder, ElasticsearchProperties elasticsearchProperties) {

        // Callback used the default {@link RequestConfig} being set to the
{@link CloseableHttpClient}
        builder.setRequestConfigCallback(requestConfigBuilder -> {

 requestConfigBuilder.setConnectTimeout(elasticsearchProperties.getConnectTimeou
t());

 requestConfigBuilder.setSocketTimeout(elasticsearchProperties.getSocketTimeout(
));

 requestConfigBuilder.setConnectionRequestTimeout(elasticsearchProperties.getCon
nectionRequestTimeout());
            return requestConfigBuilder;
        });

        // Callback used to customize the {@link CloseableHttpClient} instance
used by a {@link RestClient} instance.
        builder.setHttpClientConfigCallback(httpClientBuilder -> {

 httpClientBuilder.setMaxConnTotal(elasticsearchProperties.getMaxConnectTotal())
;

 httpClientBuilder.setMaxConnPerRoute(elasticsearchProperties.getMaxConnectPerRo
ute());
```

```
            return httpClientBuilder;
        });

        // Callback used the basic credential auth
        ElasticsearchProperties.Account account =
elasticsearchProperties.getAccount();
        if (!StringUtils.isEmpty(account.getUsername()) &&
!StringUtils.isEmpty(account.getUsername())) {
            final CredentialsProvider credentialsProvider = new
BasicCredentialsProvider();

            credentialsProvider.setCredentials(AuthScope.ANY, new
UsernamePasswordCredentials(account.getUsername(), account.getPassword()));
        }
        return new RestHighLevelClient(builder);
    }

}
```

ElasticsearchConstant.java

```
package com.xkcoding.elasticsearch.contants;

/**
 * ElasticsearchConstant
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/9/15 23:03
 */
public interface ElasticsearchConstant {

    /**
     * 索引名称
     */
    String INDEX_NAME = "person";

    String INDEX_TYPE = "person";

}
```

自定义返回结果集和异常类

Result.java

```java
package com.xkcoding.elasticsearch.common;

import lombok.Data;
import org.springframework.lang.Nullable;

import java.io.Serializable;

/**
 * Result
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/8/26 1:44
 */
@Data
public class Result<T> implements Serializable {

    private static final long serialVersionUID = 1696194043024336235L;

    /**
     * 错误码
     */
    private int errcode;

    /**
     * 错误信息
     */
    private String errmsg;

    /**
     * 响应数据
     */
    private T data;

    public Result() {
    }

    private Result(ResultCode resultCode) {
        this(resultCode.code, resultCode.msg);
    }

    private Result(ResultCode resultCode, T data) {
        this(resultCode.code, resultCode.msg, data);
    }

    private Result(int errcode, String errmsg) {
        this(errcode, errmsg, null);
    }

    private Result(int errcode, String errmsg, T data) {
        this.errcode = errcode;
        this.errmsg = errmsg;
        this.data = data;
    }
```

```
    /**
     * 返回成功
     *
     * @param <T> 泛型标记
     * @return 响应信息 {@code Result}
     */
    public static <T> Result<T> success() {
        return new Result<>(ResultCode.SUCCESS);
    }


    /**
     * 返回成功-携带数据
     *
     * @param data 响应数据
     * @param <T> 泛型标记
     * @return 响应信息 {@code Result}
     */
    public static <T> Result<T> success(@Nullable T data) {
        return new Result<>(ResultCode.SUCCESS, data);
    }




}
```

ResultCode.java

```
package com.xkcoding.elasticsearch.common;

import lombok.AllArgsConstructor;
import lombok.Getter;

/**
 * ResultCode
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/8/26 1:47
 */
@Getter
@AllArgsConstructor
public enum ResultCode {

    /**
     * 接口调用成功
     */
    SUCCESS(0, "Request Successful"),

    /**
```

```
     * 服务器暂不可用，建议稍候重试。建议重试次数不超过3次。
     */
    FAILURE(-1, "System Busy");

    final int code;

    final String msg;

}
```

ElasticsearchException.java

```
package com.xkcoding.elasticsearch.exception;

import com.xkcoding.elasticsearch.common.ResultCode;
import lombok.Getter;

/**
 * ElasticsearchException
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/8/26 1:53
 */
public class ElasticsearchException extends RuntimeException {

    @Getter
    private int errcode;

    @Getter
    private String errmsg;

    public ElasticsearchException(ResultCode resultCode) {
        this(resultCode.getCode(), resultCode.getMsg());
    }

    public ElasticsearchException(String message) {
        super(message);
    }

    public ElasticsearchException(Integer errcode, String errmsg) {
        super(errmsg);
        this.errcode = errcode;
        this.errmsg = errmsg;
    }

    public ElasticsearchException(String message, Throwable cause) {
        super(message, cause);
    }

    public ElasticsearchException(Throwable cause) {
        super(cause);
```

```
    }

    public ElasticsearchException(String message, Throwable cause, boolean
enableSuppression, boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }

}
```

Person.java

实体类

```
package com.xkcoding.elasticsearch.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;
import java.util.Date;

/**
 * Person
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/9/15 23:04
 */
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Person implements Serializable {

    private static final long serialVersionUID = 8510634155374943623L;

    /**
     * 主键
     */
    private Long id;

    /**
     * 名字
     */
    private String name;

    /**
     * 国家
     */
```

```
    private String country;

    /**
     * 年龄
     */
    private Integer age;

    /**
     * 生日
     */
    private Date birthday;

    /**
     * 介绍
     */
    private String remark;

}
```

BaseElasticsearchService.java

```
package com.xkcoding.elasticsearch.service.base;

import cn.hutool.core.bean.BeanUtil;
import com.xkcoding.elasticsearch.config.ElasticsearchProperties;
import com.xkcoding.elasticsearch.exception.ElasticsearchException;
import lombok.extern.slf4j.Slf4j;
import org.elasticsearch.action.admin.indices.create.CreateIndexRequest;
import org.elasticsearch.action.admin.indices.create.CreateIndexResponse;
import org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest;
import org.elasticsearch.action.delete.DeleteRequest;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.search.SearchRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.action.update.UpdateRequest;
import org.elasticsearch.client.HttpAsyncResponseConsumerFactory;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.common.settings.Settings;
import org.elasticsearch.common.xcontent.XContentType;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.search.builder.SearchSourceBuilder;

import javax.annotation.Resource;
import java.io.IOException;

/**
 * BaseElasticsearchService
 *
 * @author fxbin
 * @version 1.0v
```

```
 * @since 2019/9/16 15:44
 */
@Slf4j
public abstract class BaseElasticsearchService {

    @Resource
    protected RestHighLevelClient client;

    @Resource
    private ElasticsearchProperties elasticsearchProperties;

    protected static final RequestOptions COMMON_OPTIONS;

    static {
        RequestOptions.Builder builder = RequestOptions.DEFAULT.toBuilder();

        // 默认缓冲限制为100MB，此处修改为30MB。
        builder.setHttpAsyncResponseConsumerFactory(new
HttpAsyncResponseConsumerFactory.HeapBufferedResponseConsumerFactory(30 * 1024 *
1024));
        COMMON_OPTIONS = builder.build();
    }

    /**
     * create elasticsearch index (同步执行)
     *
     * @param index   elasticsearch index 索引名称
     * @param type    elasticsearch type 类型名称
     * @param mapping elasticsearch mapping 类型映射字符串
     * @author fxbin
     */
    protected void createIndexRequest(String index, String type, String mapping)
{
        try {
            CreateIndexRequest request = new CreateIndexRequest(index);
            // Settings for this index
            request.settings(Settings.builder().put("index.number_of_shards",
elasticsearchProperties.getIndex().getNumberOfShards()).put("index.number_of_rep
licas", elasticsearchProperties.getIndex().getNumberOfReplicas()));

            //创建索引时创建文档类型映射
            request.mapping(type, mapping, XContentType.JSON);

            // 同步执行
            CreateIndexResponse createIndexResponse =
client.indices().create(request, COMMON_OPTIONS);

            log.info(" whether all of the nodes have acknowledged the request :
{}", createIndexResponse.isAcknowledged());
            log.info(" Indicates whether the requisite number of shard copies
were started for each shard in the index before timing out :{}",
createIndexResponse.isShardsAcknowledged());

        } catch (IOException e) {
            throw new ElasticsearchException("创建索引 {" + index + "} 失败");
        }
    }
```

```
    /**
     * delete elasticsearch index
     *
     * @param index elasticsearch index name
     * @author fxbin
     */
    protected void deleteIndexRequest(String index) {
        DeleteIndexRequest deleteIndexRequest = buildDeleteIndexRequest(index);
        try {
            client.indices().delete(deleteIndexRequest, COMMON_OPTIONS);
        } catch (IOException e) {
            throw new ElasticsearchException("删除索引 {" + index + "} 失败");
        }
    }

    /**
     * build DeleteIndexRequest
     *
     * @param index elasticsearch index name
     * @author fxbin
     */
    private static DeleteIndexRequest buildDeleteIndexRequest(String index) {
        return new DeleteIndexRequest(index);
    }

    /**
     * build IndexRequest
     *
     * @param index  elasticsearch index name
     * @param id     request object id
     * @param object request object
     * @return {@link org.elasticsearch.action.index.IndexRequest}
     * @author fxbin
     */
    protected static IndexRequest buildIndexRequest(String index, String type,
String id, Object object) {
        return new IndexRequest(index,
type).id(id).source(BeanUtil.beanToMap(object), XContentType.JSON);
    }

    /**
     * exec updateRequest
     *
     * @param index  elasticsearch index name
     * @param id     Document id
     * @param object request object
     * @author fxbin
     */
    protected void updateRequest(String index, String type, String id, Object
object) {
        try {
            UpdateRequest updateRequest = new UpdateRequest(index, type,
id).doc(BeanUtil.beanToMap(object), XContentType.JSON);
            // 同步
            client.update(updateRequest, COMMON_OPTIONS);
        } catch (IOException e) {
            throw new ElasticsearchException("更新索引 {" + index + "} 数据 {" +
object + "} 失败");
```

```
        }
    }

    /**
     * exec deleteRequest
     *
     * @param index elasticsearch index name
     * @param id    Document id
     * @author fxbin
     */
    protected void deleteRequest(String index, String type, String id) {
        try {
            DeleteRequest deleteRequest = new DeleteRequest(index, type, id);
            // 同步
            client.delete(deleteRequest, COMMON_OPTIONS);
        } catch (IOException e) {
            throw new ElasticsearchException("删除索引 {" + index + "} 数据id {" +
id + "} 失败");
        }
    }

    /**
     * search all
     *
     * @param index elasticsearch index name
     * @return {@link SearchResponse}
     * @author fxbin
     */
    protected SearchResponse search(String index) {
        SearchRequest searchRequest = new SearchRequest(index);
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        searchRequest.source(searchSourceBuilder);
        SearchResponse searchResponse = null;
        try {
            searchResponse = client.search(searchRequest, COMMON_OPTIONS);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return searchResponse;
    }
}
```

PersonService.java

```
package com.xkcoding.elasticsearch.service;

import com.xkcoding.elasticsearch.model.Person;
import org.springframework.lang.Nullable;

import java.util.List;
```

```java
/**
 * PersonService
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/9/15 23:07
 */
public interface PersonService {

    /**
     * create Index 同步
     *
     * @param index   elasticsearch index 索引名称
     * @param type    elasticsearch type 类型名称
     * @param mapping elasticsearch mapping 类型映射字符串
     * @author fxbin
     */
    void createIndex(String index, String type, String mapping);


    /**
     * delete Index
     *
     * @param index elasticsearch index name
     * @author fxbin
     */
    void deleteIndex(String index);

    /**
     * insert document source
     *
     * @param index elasticsearch index name
     * @param list  data source
     * @author fxbin
     */
    void insert(String index, List<Person> list);

    /**
     * update document source
     *
     * @param index elasticsearch index name
     * @param list  data source
     * @author fxbin
     */
    void update(String index, List<Person> list);

    /**
     * delete document source
     *
     * @param person delete data source and allow null object
     * @author fxbin
     */
    void delete(String index, @Nullable Person person);

    /**
     * search all doc records
     *
```

```
     * @param index elasticsearch index name
     * @return person list
     * @author fxbin
     */
    List<Person> searchList(String index);

}
```

PersonServiceImpl.java

service 实现类，基本CURD操作

```
package com.xkcoding.elasticsearch.service.impl;

import cn.hutool.core.bean.BeanUtil;
import com.xkcoding.elasticsearch.contants.ElasticsearchConstant;
import com.xkcoding.elasticsearch.model.Person;
import com.xkcoding.elasticsearch.service.PersonService;
import com.xkcoding.elasticsearch.service.base.BaseElasticsearchService;
import lombok.extern.slf4j.Slf4j;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.search.SearchHit;
import org.springframework.stereotype.Service;
import org.springframework.util.ObjectUtils;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Map;

/**
 * PersonServiceImpl
 *
 * @author fxbin
 * @version v1.0
 * @since 2019/9/15 23:08
 */
@Slf4j
@Service
public class PersonServiceImpl extends BaseElasticsearchService implements
PersonService {

    @Override
    public void createIndex(String index, String type, String mapping) {
        createIndexRequest(index, type, mapping);
    }

    @Override
```

```
    public void deleteIndex(String index) {
        deleteIndexRequest(index);
    }

    @Override
    public void insert(String index, List<Person> list) {

        try {
            list.forEach(person -> {
                IndexRequest request = buildIndexRequest(index,
ElasticsearchConstant.INDEX_TYPE, String.valueOf(person.getId()), person);
                try {
                    // 同步
                    client.index(request, COMMON_OPTIONS);

                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
        } finally {
            try {
                client.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void update(String index, List<Person> list) {
        list.forEach(person -> {
            updateRequest(index, ElasticsearchConstant.INDEX_TYPE,
String.valueOf(person.getId()), person);
        });
    }

    @Override
    public void delete(String index, Person person) {
        if (ObjectUtils.isEmpty(person)) {
            // 如果person 对象为空，则删除全量
            searchList(index).forEach(p -> {
                deleteRequest(index, ElasticsearchConstant.INDEX_TYPE,
String.valueOf(p.getId()));
            });
        }
        deleteRequest(index, ElasticsearchConstant.INDEX_TYPE,
String.valueOf(person.getId()));
    }

    @Override
    public List<Person> searchList(String index) {
        SearchResponse searchResponse = search(index);
        SearchHit[] hits = searchResponse.getHits().getHits();
        List<Person> personList = new ArrayList<>();
        Arrays.stream(hits).forEach(hit -> {
            Map<String, Object> sourceAsMap = hit.getSourceAsMap();
            Person person = BeanUtil.mapToBean(sourceAsMap, Person.class, true);
            personList.add(person);
```

```
        });
        return personList;
    }
}
```

ElasticsearchApplicationTests.java

主要功能测试，参见service 注释说明

```
package com.xkcoding.elasticsearch;

import com.xkcoding.elasticsearch.contants.ElasticsearchConstant;
import com.xkcoding.elasticsearch.model.Person;
import com.xkcoding.elasticsearch.service.PersonService;
import org.elasticsearch.action.get.GetRequest;
import org.elasticsearch.action.get.GetResponse;
import org.elasticsearch.action.search.SearchRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.common.Strings;
import org.elasticsearch.common.unit.TimeValue;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.fetch.subphase.FetchSourceContext;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.concurrent.TimeUnit;

@RunWith(SpringRunner.class)
@SpringBootTest
public class ElasticsearchApplicationTests {

    @Autowired
    private PersonService personService;

    /**
     * 测试删除索引
     */
    @Test
```

```
    public void deleteIndexTest() {
        personService.deleteIndex(ElasticsearchConstant.INDEX_NAME);
    }

    /**
     * 测试创建索引
     */
    @Test
    public void createIndexTest() {
        String personMapping = "{\n" +
            "\"person\": {\n" +
            "\"properties\": {\n" +
            "\"birthday\": {\n" +
            "\"type\": \"date\"\n" +
            "},\n" +
            "\"country\": {\n" +
            "\"type\": \"keyword\"\n" +
            "},\n" +
            "\"name\": {\n" +
            "\"type\": \"keyword\"\n" +
            "},\n" +
            "\"remark\": {\n" +
            "\"analyzer\": \"ik_smart\",\n" +
            "\"type\": \"text\"\n" +
            "},\n" +
            "\"age\": {\n" +
            "\"type\": \"integer\"\n" +
            "}\n" +
            "}\n" +
            "}\n" +
            "}";

        personService.createIndex(ElasticsearchConstant.INDEX_NAME,
ElasticsearchConstant.INDEX_TYPE, personMapping);
    }

    /**
     * 测试新增
     */
    @Test
    public void insertTest() {
        List<Person> list = new ArrayList<>();
        list.add(Person.builder().age(11).birthday(new
Date()).country("CN").id(1L).name("哈哈").remark("test1").build());
        list.add(Person.builder().age(22).birthday(new
Date()).country("US").id(2L).name("hiahia").remark("test2").build());
        list.add(Person.builder().age(33).birthday(new
Date()).country("ID").id(3L).name("呵呵").remark("test3").build());

        personService.insert(ElasticsearchConstant.INDEX_NAME, list);
    }

    /**
     * 测试更新
     */
    @Test
    public void updateTest() {
```

```
        Person person = Person.builder().age(33).birthday(new
Date()).country("ID_update").id(3L).name("呵呵
update").remark("test3_update").build();
        List<Person> list = new ArrayList<>();
        list.add(person);
        personService.update(ElasticsearchConstant.INDEX_NAME, list);
    }

    /**
     * 测试删除
     */
    @Test
    public void deleteTest() {
        personService.delete(ElasticsearchConstant.INDEX_NAME,
Person.builder().id(1L).build());
        personService.delete(ElasticsearchConstant.INDEX_NAME,
Person.builder().id(2L).build());
        personService.delete(ElasticsearchConstant.INDEX_NAME,
Person.builder().id(3L).build());
    }

    /**
     * 测试查询
     */
    @Test
    public void searchListTest() {
        List<Person> personList =
personService.searchList(ElasticsearchConstant.INDEX_NAME);
        System.out.println(personList);
    }

    @Resource
    private RestHighLevelClient client;

    /**
     * 根据id查询，并指定返回的字段
     *
     * @throws IOException
     */
    @Test
    public void testQuery() throws IOException {
        GetRequest getRequest = new GetRequest("person", "person",
            "1");
        // 指定返回的字段
        String[] includes = new String[]{"name", "age"};
        String[] excludes = Strings.EMPTY_ARRAY;
        FetchSourceContext fetchSourceContext = new FetchSourceContext(true,
includes, excludes);
        getRequest.fetchSourceContext(fetchSourceContext);
        GetResponse response = client.get(getRequest, RequestOptions.DEFAULT);
        System.out.println("数据 -> " + response.getSource());
    }

    /**
     * 判断是否存在
     *
     * @throws Exception
     */
```

```
    @Test
    public void testExists() throws Exception {
        GetRequest getRequest = new GetRequest("person", "person",
            "1");
        // 不返回的字段
        getRequest.fetchSourceContext(new FetchSourceContext(false));
        boolean exists = client.exists(getRequest, RequestOptions.DEFAULT);
        System.out.println("exists -> " + exists);
    }

    /**
     * 测试搜索，参考searchListTest
     *
     * @throws Exception
     */
    @Test
    public void testSearch() throws Exception {
        SearchRequest searchRequest = new SearchRequest("person");
        searchRequest.types("person");
        SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();
        sourceBuilder.query(QueryBuilders.matchQuery("name", "哈哈"));
        sourceBuilder.from(0);
        sourceBuilder.size(5);
        // 设置超时属性
        sourceBuilder.timeout(new TimeValue(60, TimeUnit.SECONDS));
        searchRequest.source(sourceBuilder);
        SearchResponse search = client.search(searchRequest,
RequestOptions.DEFAULT);
        System.out.println("搜索到 " + search.getHits().totalHits + " 条数据.");
        SearchHits hits = search.getHits();
        for (SearchHit hit : hits) {
            System.out.println(hit.getSourceAsString());
        }
    }

}
```