此 demo 主要演示了 Spring Boot 如何集成 spring-boot-starter-data-elasticsearch 完成对 ElasticSearch 的高级使用技巧，包括创建索引、配置映射、删除索引、增删改查基本操作、复杂查询、高级查询、聚合查询等。

参考：https://github.com/xkcoding/spring-boot-demo/tree/master/spring-boot-demo-elasticsearch

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

spring-data-elasticsearch 官方文档：https://spring.io/projects/spring-data-elasticsearch#learn

注意：本demo中，ElasticSearch版本为 6.5.4。

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>

    <artifactId>spring-boot-demo-elasticsearch</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>spring-boot-demo-elasticsearch</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>com.xkcoding</groupId>
        <artifactId>spring-boot-demo</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </parent>

    <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>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
        </dependency>

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

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

        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
        </dependency>

        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
        </dependency>
    </dependencies>

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

</project>
```

application.yml

```
spring:
  data:
    elasticsearch:
      cluster-name: ahhs6.5.4
      cluster-nodes: 192.168.1.128:9300,192.168.1.129:9300,192.168.1.130:9300
```

EsConsts.java

```
package com.xkcoding.elasticsearch.constants;

/**
 * <p>
 * ES常量池
 * </p>
 *
 * @package: com.xkcoding.elasticsearch.constants
 * @description: ES常量池
 * @author: yangkai.shen
 * @date: Created in 2018-12-20 17:30
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
public interface EsConsts {
    /**
     * 索引名称
     */
    String INDEX_NAME = "person";

    /**
     * 类型名称
     */
    String TYPE_NAME = "person";
}
```

Person.java

实体类:

@Document 注解主要声明索引名、类型名、分片数量和备份数量

@Field 注解主要声明字段对应ES的类型

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

import com.xkcoding.elasticsearch.constants.EsConsts;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
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.util.Date;

/**
 * <p>
```

```
 * 用户实体类
 * </p>
 *
 * @package: com.xkcoding.elasticsearch.model
 * @description: 用户实体类
 * @author: yangkai.shen
 * @date: Created in 2018-12-20 17:29
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
@Document(indexName = EsConsts.INDEX_NAME, type = EsConsts.TYPE_NAME, shards =
1, replicas = 0)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Person {
    /**
     * 主键
     */
    @Id
    private Long id;

    /**
     * 名字
     */
    @Field(type = FieldType.Keyword)
    private String name;

    /**
     * 国家
     */
    @Field(type = FieldType.Keyword)
    private String country;

    /**
     * 年龄
     */
    @Field(type = FieldType.Integer)
    private Integer age;

    /**
     * 生日
     */
    @Field(type = FieldType.Date)
    private Date birthday;

    /**
     * 介绍
     */
    @Field(type = FieldType.Text, analyzer = "ik_smart")
    private String remark;
}
```

PersonRepository.java

```
package com.xkcoding.elasticsearch.repository;

import com.xkcoding.elasticsearch.model.Person;
import
org.springframework.data.elasticsearch.repository.ElasticsearchRepository;

import java.util.List;

/**
 * <p>
 * 用户持久层
 * </p>
 *
 * @package: com.xkcoding.elasticsearch.repository
 * @description: 用户持久层
 * @author: yangkai.shen
 * @date: Created in 2018-12-20 19:00
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
public interface PersonRepository extends ElasticsearchRepository<Person, Long>
{

    /**
     * 根据年龄区间查询
     *
     * @param min 最小值
     * @param max 最大值
     * @return 满足条件的用户列表
     */
    List<Person> findByAgeBetween(Integer min, Integer max);
}
```

TemplateTest.java

主要测试创建索引、映射配置、删除索引

```
package com.xkcoding.elasticsearch.template;
```

```java
import com.xkcoding.elasticsearch.SpringBootDemoElasticsearchApplicationTests;
import com.xkcoding.elasticsearch.model.Person;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.elasticsearch.core.ElasticsearchTemplate;

/**
 * <p>
 * 测试 ElasticTemplate 的创建/删除
 * </p>
 *
 * @package: com.xkcoding.elasticsearch.template
 * @description: 测试 ElasticTemplate 的创建/删除
 * @author: yangkai.shen
 * @date: Created in 2018-12-20 17:46
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
public class TemplateTest extends SpringBootDemoElasticsearchApplicationTests {
    @Autowired
    private ElasticsearchTemplate esTemplate;

    /**
     * 测试 ElasticTemplate 创建 index
     */
    @Test
    public void testCreateIndex() {
        // 创建索引，会根据Item类的@Document注解信息来创建
        esTemplate.createIndex(Person.class);

        // 配置映射，会根据Item类中的id、Field等字段来自动完成映射
        esTemplate.putMapping(Person.class);
    }

    /**
     * 测试 ElasticTemplate 删除 index
     */
    @Test
    public void testDeleteIndex() {
        esTemplate.deleteIndex(Person.class);
    }
}
```

PersonRepositoryTest.java

主要功能，参见方法上方注释

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

```
import cn.hutool.core.date.DateUtil;
import cn.hutool.json.JSONUtil;
import com.google.common.collect.Lists;
import com.xkcoding.elasticsearch.SpringBootDemoElasticsearchApplicationTests;
import com.xkcoding.elasticsearch.model.Person;
import lombok.extern.slf4j.Slf4j;
import org.elasticsearch.index.query.MatchQueryBuilder;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.search.aggregations.AggregationBuilders;
import org.elasticsearch.search.aggregations.bucket.terms.StringTerms;
import org.elasticsearch.search.aggregations.metrics.avg.InternalAvg;
import org.elasticsearch.search.sort.SortBuilders;
import org.elasticsearch.search.sort.SortOrder;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.data.elasticsearch.core.aggregation.AggregatedPage;
import org.springframework.data.elasticsearch.core.query.FetchSourceFilter;
import
org.springframework.data.elasticsearch.core.query.NativeSearchQueryBuilder;

import java.util.List;

/**
 * <p>
 * 测试 Repository 操作ES
 * </p>
 *
 * @package: com.xkcoding.elasticsearch.repository
 * @description: 测试 Repository 操作ES
 * @author: yangkai.shen
 * @date: Created in 2018-12-20 19:03
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
@Slf4j
public class PersonRepositoryTest extends
SpringBootDemoElasticsearchApplicationTests {
    @Autowired
    private PersonRepository repo;

    /**
     * 测试新增
     */
    @Test
    public void save() {
        Person person = new Person(1L, "刘备", "蜀国", 18, DateUtil.parse("1990-
01-02 03:04:05"), "刘备（161年－223年6月10日），即汉昭烈帝（221年－223年在位），又称先
主，字玄德，东汉末年幽州涿郡涿县（今河北省涿州市）人，西汉中山靖王刘胜之后，三国时期蜀汉开国皇
帝、政治家。\n刘备少年时拜卢植为师；早年颠沛流离，备尝艰辛，投靠过多个诸侯，曾参与镇压黄巾起义。
先后率军救援北海相孔融、徐州牧陶谦等。陶谦病亡后，将徐州让与刘备。赤壁之战时，刘备与孙权联盟击败
曹操，趁势夺取荆州。而后进取益州。于章武元年（221年）在成都称帝，国号汉，史称蜀或蜀汉。《三国
志》评刘备的机权干略不及曹操，但其弘毅宽厚，知人待士，百折不挠，终成帝业。刘备也称自己做事“每与
操反，事乃成尔”。\n章武三年（223年），刘备病逝于白帝城，终年六十三岁，谥号昭烈皇帝，庙号烈祖，
葬惠陵。后世有众多文艺作品以其为主角，在成都武侯祠有昭烈庙为纪念。");
```

```
        Person save = repo.save(person);
        log.info("【save】= {}", save);
    }

    /**
     * 测试批量新增
     */
    @Test
    public void saveList() {
        List<Person> personList = Lists.newArrayList();
        personList.add(new Person(2L, "曹操", "魏国", 20, DateUtil.parse("1988-01-02 03:04:05"), "曹操（155年－220年3月15日），字孟德，一名吉利，小字阿瞒，沛国谯县（今安徽亳州）人。东汉末年杰出的政治家、军事家、文学家、书法家，三国中曹魏政权的奠基人。\n曹操曾担任东汉丞相，后加封魏王，奠定了曹魏立国的基础。去世后谥号为武王。其子曹丕称帝后，追尊为武皇帝，庙号太祖。\n东汉末年，天下大乱，曹操以汉天子的名义征讨四方，对内消灭二袁、吕布、刘表、马超、韩遂等割据势力，对外降服南匈奴、乌桓、鲜卑等，统一了中国北方，并实行一系列政策恢复经济生产和社会秩序，扩大屯田、兴修水利、奖励农桑、重视手工业、安置流亡人口、实行“租调制”，从而使中原社会渐趋稳定、经济出现转机。黄河流域在曹操统治下，政治渐见清明，经济逐步恢复，阶级压迫稍有减轻，社会风气有所好转。曹操在汉朝的名义下所采取的一些措施具有积极作用。\n曹操军事上精通兵法，重贤爱才，为此不惜一切代价将看中的潜能分子收于麾下；生活上善诗歌，抒发自己的政治抱负，并反映汉末人民的苦难生活，气魄雄伟，慷慨悲凉；散文亦清峻整洁，开启并繁荣了建安文学，给后人留下了宝贵的精神财富，鲁迅评价其为“改造文章的祖师”。同时曹操也擅长书法，唐朝张怀瓘在《书断》将曹操的章草评为“妙品”。"));
        personList.add(new Person(3L, "孙权", "吴国", 19, DateUtil.parse("1989-01-02 03:04:05"), "孙权（182年－252年5月21日），字仲谋，吴郡富春（今浙江杭州富阳区）人。三国时代孙吴的建立者（229年－252年在位）。\n孙权的父亲孙坚和兄长孙策，在东汉末年群雄割据中打下了江东基业。建安五年（200年），孙策遇刺身亡，孙权继之掌事，成为一方诸侯。建安十三年（208年），与刘备建立孙刘联盟，并于赤壁之战中击败曹操，奠定三国鼎立的基础。建安二十四年（219年），孙权派吕蒙成功袭取刘备的荆州，使领土面积大大增加。\n黄武元年（222年），孙权被魏文帝曹丕册封为吴王，建立吴国。同年，在夷陵之战中大败刘备。黄龙元年（229年），在武昌正式称帝，国号吴，不久后迁都建业。孙权称帝后，设置农官，实行屯田，设置郡县，并继续剿抚山越，促进了江南经济的发展。在此基础上，他又多次派人出海。黄龙二年（230年），孙权派卫温、诸葛直抵达夷州。\n孙权晚年在继承人问题上反复无常，引致群下党争，朝局不稳。太元元年（252年）病逝，享年七十一岁，在位二十四年，谥号大皇帝，庙号太祖，葬于蒋陵。\n孙权亦善书，唐代张怀瓘在《书估》中将其书法列为第三等。"));
        personList.add(new Person(4L, "诸葛亮", "蜀国", 16, DateUtil.parse("1992-01-02 03:04:05"), "诸葛亮（181年-234年10月8日），字孔明，号卧龙，徐州琅琊阳都（今山东临沂市沂南县）人，三国时期蜀国丞相，杰出的政治家、军事家、外交家、文学家、书法家、发明家。\n早年随叔父诸葛玄到荆州，诸葛玄死后，诸葛亮就在襄阳隆中隐居。后刘备三顾茅庐请出诸葛亮，联孙抗曹，于赤壁之战大败曹军。形成三国鼎足之势，又夺占荆州。建安十六年（211年），攻取益州。继又击败曹军，夺得汉中。蜀章武元年（221年），刘备在成都建立蜀汉政权，诸葛亮被任命为丞相，主持朝政。蜀后主刘禅继位，诸葛亮被封为武乡侯，领益州牧。勤勉谨慎，大小政事必亲自处理，赏罚严明；与东吴联盟，改善和西南各族的关系；实行屯田政策，加强战备。前后六次北伐中原，多以粮尽无功。终因积劳成疾，于蜀建兴十二年（234年）病逝于五丈原（今陕西宝鸡岐山境内），享年54岁。刘禅追封其为忠武侯，后世常以武侯尊称诸葛亮。东晋政权因其军事才能特追封他为武兴王。\n诸葛亮散文代表作有《出师表》《诫子书》等。曾发明木牛流马、孔明灯等，并改造连弩，叫做诸葛连弩，可一弩十矢俱发。诸葛亮一生“鞠躬尽瘁、死而后已”，是中国传统文化中忠臣与智者的代表人物。"));
        Iterable<Person> people = repo.saveAll(personList);
        log.info("【people】= {}", people);
    }

    /**
     * 测试更新
     */
    @Test
    public void update() {
        repo.findById(1L).ifPresent(person -> {
            person.setRemark(person.getRemark() + "\n更新更新更新更新更新");
            Person save = repo.save(person);
            log.info("【save】= {}", save);
```

```
        });
    }

    /**
     * 测试删除
     */
    @Test
    public void delete() {
        // 主键删除
        repo.deleteById(1L);

        // 对象删除
        repo.findById(2L).ifPresent(person -> repo.delete(person));

        // 批量删除
        repo.deleteAll(repo.findAll());
    }

    /**
     * 测试普通查询，按生日倒序
     */
    @Test
    public void select() {
        repo.findAll(Sort.by(Sort.Direction.DESC, "birthday"))
                .forEach(person -> log.info("{} 生日: {}", person.getName(),
DateUtil.formatDateTime(person.getBirthday())));
    }

    /**
     * 自定义查询，根据年龄范围查询
     */
    @Test
    public void customSelectRangeOfAge() {
        repo.findByAgeBetween(18, 19).forEach(person -> log.info("{} 年龄: {}",
person.getName(), person.getAge()));
    }

    /**
     * 高级查询
     */
    @Test
    public void advanceSelect() {
        // QueryBuilders 提供了很多静态方法，可以实现大部分查询条件的封装
        MatchQueryBuilder queryBuilder = QueryBuilders.matchQuery("name", "孙
权");
        log.info("【queryBuilder】= {}", queryBuilder.toString());

        repo.search(queryBuilder).forEach(person -> log.info("【person】= {}",
person));
    }

    /**
     * 自定义高级查询
     */
    @Test
    public void customAdvanceSelect() {
        // 构造查询条件
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();
```

```
        // 添加基本的分词条件
        queryBuilder.withQuery(QueryBuilders.matchQuery("remark", "东汉"));
        // 排序条件

 queryBuilder.withSort(SortBuilders.fieldSort("age").order(SortOrder.DESC));
        // 分页条件
        queryBuilder.withPageable(PageRequest.of(0, 2));
        Page<Person> people = repo.search(queryBuilder.build());
        log.info("【people】总条数 = {}", people.getTotalElements());
        log.info("【people】总页数 = {}", people.getTotalPages());
        people.forEach(person -> log.info("【person】= {}，年龄 = {}",
person.getName(), person.getAge()));
    }

    /**
     * 测试聚合，测试平均年龄
     */
    @Test
    public void agg() {
        // 构造查询条件
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();
        // 不查询任何结果
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[]{""},
null));

        // 平均年龄

 queryBuilder.addAggregation(AggregationBuilders.avg("avg").field("age"));

        log.info("【queryBuilder】= {}",
JSONUtil.toJsonStr(queryBuilder.build()));

        AggregatedPage<Person> people = (AggregatedPage<Person>)
repo.search(queryBuilder.build());
        double avgAge = ((InternalAvg) people.getAggregation("avg")).getValue();
        log.info("【avgAge】= {}", avgAge);
    }

    /**
     * 测试高级聚合查询，每个国家的人有几个，每个国家的平均年龄是多少
     */
    @Test
    public void advanceAgg() {
        // 构造查询条件
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();
        // 不查询任何结果
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[]{""},
null));

        // 1. 添加一个新的聚合，聚合类型为terms，聚合名称为country，聚合字段为country

 queryBuilder.addAggregation(AggregationBuilders.terms("country").field("country
")
                // 2. 在国家聚合桶内进行嵌套聚合，求平均年龄
                .subAggregation(AggregationBuilders.avg("avg").field("age")));

        log.info("【queryBuilder】= {}",
JSONUtil.toJsonStr(queryBuilder.build()));
```

```
        // 3. 查询
        AggregatedPage<Person> people = (AggregatedPage<Person>)
repo.search(queryBuilder.build());

        // 4. 解析
        // 4.1. 从结果中取出名为 country 的那个聚合，因为是利用String类型字段来进行的term
聚合，所以结果要强转为StringTerm类型
        StringTerms country = (StringTerms) people.getAggregation("country");
        // 4.2. 获取桶
        List<StringTerms.Bucket> buckets = country.getBuckets();
        for (StringTerms.Bucket bucket : buckets) {
            // 4.3. 获取桶中的key，即国家名称  4.4. 获取桶中的文档数量
            log.info("{} 总共有 {} 人", bucket.getKeyAsString(),
bucket.getDocCount());
            // 4.5. 获取子聚合结果:
            InternalAvg avg = (InternalAvg)
bucket.getAggregations().asMap().get("avg");
            log.info("平均年龄:{}", avg);
        }
    }

}
```