---elasticsearch实战学习笔记---

初始化项目

https://github.com/maomao124/elasticsearch_hotel

maven依赖

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

        <!--spring boot elasticsearch high-level-client-->
        <dependency>
            <groupId>org.elasticsearch.client</groupId>
            <artifactId>elasticsearch-rest-high-level-client</artifactId>
        </dependency>
        <!--spring-boot mybatis-plus依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.5.1</version>
        </dependency>
        <!--mysql依赖 spring-boot-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!--spring-boot lombok-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!--阿里巴巴的FastJson json解析-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.79</version>
        </dependency>
        <!--spring-boot 开发者工具 主要用于热部署-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <optional>true</optional>
        </dependency>
        <!--spring-boot druid连接池依赖-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.2.8</version>
        </dependency>

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

配置

设置端口号为8089

```
server:
  port: 8089
```

配置数据源

```
   # 导入jdbc.properties文件
  config:
    import: classpath:jdbc.properties
 # 配置数据源
  datasource:
    # driver-class-name: ${jdbc.driver}
    # url: ${jdbc.url}
    # username: ${jdbc.username}
    # password: ${jdbc.password}
    # 配置数据源-druid
    druid:
      driver-class-name: ${jdbc.driver}
      url: ${jdbc.url}
      username: ${jdbc.username}
      password: ${jdbc.password}
```

所有配置

```


#多环境开发---单文件
spring:
  profiles:
```

```yaml
    # 当前环境
    active: dev

#-----通用环境-------------------------------------------------------

  # 导入jdbc.properties文件
  config:
    import: classpath:jdbc.properties


  #spring-mvc配置
  mvc:
    # 视图解析器
    view:
      prefix:
      suffix: .html


server:
  port: 8089





---
#------开发环境------------------------------------------------------

spring:
  config:
    activate:
      on-profile: dev

  # 配置数据源
  datasource:
    # driver-class-name: ${jdbc.driver}
    # url: ${jdbc.url}
    # username: ${jdbc.username}
    # password: ${jdbc.password}
    # 配置数据源-druid
    druid:
      driver-class-name: ${jdbc.driver}
      url: ${jdbc.url}
      username: ${jdbc.username}
      password: ${jdbc.password}



  # 关于热部署
  devtools:
    restart:
      # 热部署开关
      enabled: true
      # 自定义不参与重启排除项
      exclude: public/**,static/**
```

```yaml
# 开启debug模式，输出调试信息，常用于检查系统运行状况
#debug: true

# 设置日志级别，root表示根节点，即整体应用日志级别
logging:
 # 日志输出到文件的文件名
  file:
     name: server.log
  # 字符集
  charset:
    file: UTF-8
  # 分文件
  logback:
    rollingpolicy:
      #最大文件大小
      max-file-size: 16KB
      # 文件格式
      file-name-pattern: server_log-%d{yyyy/MM月/dd日/}%i.log
  # 设置日志组
  group:
  # 自定义组名，设置当前组中所包含的包
    mao_pro: mao
  level:
    root: info
    # 为对应组设置日志级别
    mao_pro: debug
    # 日志输出格式
# pattern:
  # console: "%d %clr(%p) --- [%16t] %clr(%-40.40c){cyan} : %m %n"




---
#----生产环境----------------------------------------------------


spring:
  config:
    activate:
      on-profile: pro






# 开启debug模式，输出调试信息，常用于检查系统运行状况
#debug: true

# 设置日志级别，root表示根节点，即整体应用日志级别
logging:
  # 日志输出到文件的文件名
  file:
    name: server.log
  # 字符集
  charset:
    file: UTF-8
  # 分文件
  logback:
    rollingpolicy:
      #最大文件大小
      max-file-size: 4MB
      # 文件格式
      file-name-pattern: server_log-%d{yyyy/MM月/dd日/}%i.log
  # 设置日志组
  group:
    # 自定义组名，设置当前组中所包含的包
    mao_pro: mao
  level:
    root: info
    # 为对应组设置日志级别
    mao_pro: warn
    # 日志输出格式
  # pattern:
  # console: "%d %clr(%p) --- [%16t] %clr(%-40.40c){cyan} : %m %n"





---
#----测试环境-----------------------------------------------------

spring:
  config:
    activate:
      on-profile: test
```

类

ElasticSearchConfig：

```java
package mao.elasticsearch_hotel.config;

import org.apache.http.HttpHost;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.config
 * Class(类名): ElasticSearchConfig
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 15:15
 * Version(版本): 1.0
 * Description(描述)： ElasticSearch配置
 */

@Configuration
public class ElasticSearchConfig
{

    /**
     * 创建ElasticSearch客户端
     *
     * @return RestHighLevelClient
     */
    @Bean
    public RestHighLevelClient restHighLevelClient()
    {
        return new RestHighLevelClient(RestClient.builder(
                new HttpHost("localhost", 9200, "http")
        ));
    }

}
```

创建索引的json：

```
package mao.elasticsearch_hotel.constants;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.constants
 * Class(类名): HotelConstants
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 15:10
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class HotelConstants
{
    /**
     * 创建hotel索引的json文件
     */
    public static final String MAPPING_TEMPLATE = "{\n" +
            "  \"mappings\": {\n" +
            "    \"properties\": {\n" +
            "      \"id\": {\n" +
            "        \"type\": \"keyword\"\n" +
            "      },\n" +
            "      \"name\": {\n" +
            "        \"type\": \"text\",\n" +
            "        \"analyzer\": \"ik_max_word\",\n" +
            "        \"copy_to\": \"all\"\n" +
            "      },\n" +
            "      \"address\": {\n" +
            "        \"type\": \"keyword\",\n" +
            "        \"index\": false\n" +
            "      },\n" +
            "      \"price\": {\n" +
            "        \"type\": \"integer\"\n" +
            "      },\n" +
            "      \"score\": {\n" +
            "        \"type\": \"integer\"\n" +
            "      },\n" +
            "      \"brand\": {\n" +
            "        \"type\": \"keyword\",\n" +
            "        \"copy_to\": \"all\"\n" +
            "      },\n" +
            "      \"city\": {\n" +
            "        \"type\": \"keyword\"\n" +
            "      },\n" +
            "      \"starName\": {\n" +
            "        \"type\": \"keyword\"\n" +
            "      },\n" +
            "      \"business\": {\n" +
            "        \"type\": \"keyword\",\n" +
            "        \"copy_to\": \"all\"\n" +
            "      },\n" +
            "      \"pic\": {\n" +
            "        \"type\": \"keyword\",\n" +
            "        \"index\": false\n" +
            "      },\n" +
            "      \"location\": {\n" +
            "        \"type\": \"geo_point\"\n" +
            "      },\n" +
            "      \"all\": {\n" +
            "        \"type\": \"text\",\n" +
            "        \"analyzer\": \"ik_max_word\"\n" +
            "      }\n" +
            "    }\n" +
            "  }\n" +
            "}";
}
```

其它的略

创建索引

```
package mao.elasticsearch_hotel.hotel;

import org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest;
```

```java
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.CreateIndexRequest;
import org.elasticsearch.client.indices.GetIndexRequest;
import org.elasticsearch.xcontent.XContentType;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;

import static mao.elasticsearch_hotel.constants.HotelConstants.MAPPING_TEMPLATE;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.hotel
 * Class(类名): HotelIndexTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 16:11
 * Version(版本): 1.0
 * Description(描述)： 文档索引操作的测试
 */

@SpringBootTest
public class HotelIndexTest
{
    @Autowired
    private RestHighLevelClient client;

    /**
     * 创建hotel索引
     *
     * @throws IOException IOException
     */
    @Test
    void testCreateIndex() throws IOException
    {
        // 1.准备Request      PUT /hotel
        CreateIndexRequest request = new CreateIndexRequest("hotel");
        // 2.准备请求参数
        request.source(MAPPING_TEMPLATE, XContentType.JSON);
        // 3.发送请求
        client.indices().create(request, RequestOptions.DEFAULT);
    }

    /**
     * 查询索引是否存在
     *
     * @throws IOException IOException
     */
    @Test
    void testExistsIndex() throws IOException
    {
        // 1.准备Request
        GetIndexRequest request = new GetIndexRequest("hotel");
        // 3.发送请求
        boolean isExists = client.indices().exists(request,
RequestOptions.DEFAULT);

        System.out.println(isExists ? "存在" : "不存在");
    }

    /**
     * 删除索引
     *
     * @throws IOException IOException
     */
    @Test
    void testDeleteIndex() throws IOException
    {
        // 1.准备Request
        DeleteIndexRequest request = new DeleteIndexRequest("hotel");
        // 3.发送请求
        client.indices().delete(request, RequestOptions.DEFAULT);
    }
}
```

添加数据

```
package mao.elasticsearch_hotel.hotel;

import com.alibaba.fastjson.JSON;
import mao.elasticsearch_hotel.entity.Hotel;
import mao.elasticsearch_hotel.entity.HotelDoc;
import mao.elasticsearch_hotel.service.IHotelService;
import org.elasticsearch.action.bulk.BulkRequest;
import org.elasticsearch.action.delete.DeleteRequest;
import org.elasticsearch.action.get.GetRequest;
import org.elasticsearch.action.get.GetResponse;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.update.UpdateRequest;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.xcontent.XContentType;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.List;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.hotel
 * Class(类名): HotelDocumentTest
 * Author(作者）: mao
 * Author QQ：1296193245
```

```java
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 16:09
 * Version(版本): 1.0
 * Description(描述)： 文档操作的测试
 */

@SpringBootTest
public class HotelDocumentTest
{
    @Autowired
    private RestHighLevelClient client;

    @Autowired
    private IHotelService hotelService;

    /**
     * 测试添加一条数据
     *
     * @throws IOException IOException
     */
    @Test
    void testAddDocument() throws IOException
    {
        // 1.查询数据库hotel数据
        Hotel hotel = hotelService.getById(61083L);
        // 2.转换为HotelDoc
        HotelDoc hotelDoc = new HotelDoc(hotel);
        // 3.转JSON
        String json = JSON.toJSONString(hotelDoc);

        // 1.准备Request
        IndexRequest request = new
IndexRequest("hotel").id(hotelDoc.getId().toString());
        // 2.准备请求参数DSL，其实就是文档的JSON字符串
        request.source(json, XContentType.JSON);
        // 3.发送请求
        client.index(request, RequestOptions.DEFAULT);
    }

    /**
     * 获取刚才插入的数据
     *
     * @throws IOException IOException
     */
    @Test
    void testGetDocumentById() throws IOException
    {
        // 1.准备Request      // GET /hotel/_doc/{id}
        GetRequest request = new GetRequest("hotel", "61083");
        // 2.发送请求
        GetResponse response = client.get(request, RequestOptions.DEFAULT);
        // 3.解析响应结果
        String json = response.getSourceAsString();

        HotelDoc hotelDoc = JSON.parseObject(json, HotelDoc.class);
        System.out.println("hotelDoc = " + hotelDoc);
    }
    /**
     * 删除刚才插入的数据
     *
     * @throws IOException IOException
     */
    @Test
    void testDeleteDocumentById() throws IOException
    {
        // 1.准备Request      // DELETE /hotel/_doc/{id}
        DeleteRequest request = new DeleteRequest("hotel", "61083");
        // 2.发送请求
        client.delete(request, RequestOptions.DEFAULT);
    }

    /**
     * 更新文档数据
     *
     * @throws IOException IOException
     */
    @Test
    void testUpdateById() throws IOException
    {
        // 1.准备Request
        UpdateRequest request = new UpdateRequest("hotel", "61083");
        // 2.准备参数
        request.doc(
                "price", "870"
        );
        // 3.发送请求
        client.update(request, RequestOptions.DEFAULT);
    }

    /**
     * 从数据库里批量插入所有数据
     *
     * @throws IOException IOException
     */
    @Test
    void testBulkRequest() throws IOException
    {
        // 查询所有的酒店数据
        List<Hotel> list = hotelService.list();
        System.out.println("一共: " + list.size() + "条数据");
        // 1.准备Request
        BulkRequest request = new BulkRequest();
        // 2.准备参数
        for (Hotel hotel : list)
        {
            // 2.1.转为HotelDoc
            HotelDoc hotelDoc = new HotelDoc(hotel);
            // 2.2.转json
            String json = JSON.toJSONString(hotelDoc);
            // 2.3.添加请求
            request.add(new
IndexRequest("hotel").id(hotel.getId().toString()).source(json,
XContentType.JSON));
```

```
        }

        // 3.发送请求
        client.bulk(request, RequestOptions.DEFAULT);
    }


}
```

测试搜索

```
package mao.elasticsearch_hotel.hotel;

import com.alibaba.fastjson.JSON;
import mao.elasticsearch_hotel.entity.HotelDoc;
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.index.query.QueryBuilders;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.search.fetch.subphase.highlight.HighlightBuilder;
import org.elasticsearch.search.fetch.subphase.highlight.HighlightField;
import org.elasticsearch.search.sort.SortOrder;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.Map;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.hotel
 * Class(类名): HotelSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 16:13
 * Version(版本): 1.0
 * Description(描述)： 搜索测试
 */

@SpringBootTest
public class HotelSearchTest
{
    @Autowired
    private RestHighLevelClient client;

    /**
```

```
     * 搜索所有数据
     *
     * @throws IOException IOException
     */
    @Test
    void testMatchAll() throws IOException
    {
        // 1.准备request
        SearchRequest request = new SearchRequest("hotel");
        // 2.准备请求参数
        request.source().query(QueryBuilders.matchAllQuery());
        // 3.发送请求，得到响应
        SearchResponse response = client.search(request,
RequestOptions.DEFAULT);
        // 4.结果解析
        handleResponse(response);
    }

    /**
     * 测试搜索某些数据
     *
     * @throws IOException IOException
     */
    @Test
    void testMatch() throws IOException
    {
        // 1.准备request
        SearchRequest request = new SearchRequest("hotel");
        // 2.准备请求参数
        // request.source().query(QueryBuilders.matchQuery("all", "外滩如家"));
        request.source().query(QueryBuilders.multiMatchQuery("外滩如家", "name",
"brand", "city"));
        // 3.发送请求，得到响应
        SearchResponse response = client.search(request,
RequestOptions.DEFAULT);
        // 4.结果解析
        handleResponse(response);
    }

    /**
     * 测试boolQuery
     *
     * @throws IOException IOException
     */
    @Test
    void testBool() throws IOException
    {
        // 1.准备request
        SearchRequest request = new SearchRequest("hotel");
        // 2.准备请求参数
       /*
         BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
        // 2.1.must
        boolQuery.must(QueryBuilders.termQuery("city", "杭州"));
        // 2.2.filter
        boolQuery.filter(QueryBuilders.rangeQuery("price").lte(250));
        */
```

```java
        request.source().query(
                QueryBuilders.boolQuery()
                        .must(QueryBuilders.termQuery("city", "杭州"))
                        .filter(QueryBuilders.rangeQuery("price").lte(250))
        );
        // 3.发送请求，得到响应
        SearchResponse response = client.search(request,
RequestOptions.DEFAULT);
        // 4.结果解析
        handleResponse(response);
    }

    /**
     * 测试排序和分页
     *
     * @throws IOException IOException
     */
    @Test
    void testSortAndPage() throws IOException
    {
        int page = 2, size = 5;

        // 1.准备request
        SearchRequest request = new SearchRequest("hotel");
        // 2.准备请求参数
        // 2.1.query
        request.source()
                .query(QueryBuilders.matchAllQuery());
        // 2.2.排序sort
        request.source().sort("price", SortOrder.ASC);
        // 2.3.分页 from\size
        request.source().from((page - 1) * size).size(size);

        // 3.发送请求，得到响应
        SearchResponse response = client.search(request,
RequestOptions.DEFAULT);
        // 4.结果解析
        handleResponse(response);
    }

    /**
     * 测试高亮
     *
     * @throws IOException IOException
     */
    @Test
    void testHighlight() throws IOException
    {
        // 1.准备request
        SearchRequest request = new SearchRequest("hotel");
        // 2.准备请求参数
        // 2.1.query
        request.source().query(QueryBuilders.matchQuery("all", "外滩如家"));
        // 2.2.高亮
        request.source().highlighter(new
HighlightBuilder().field("name").requireFieldMatch(false));
        // 3.发送请求，得到响应
        SearchResponse response = client.search(request,
RequestOptions.DEFAULT);
        // 4.结果解析
        handleResponse(response);
    }

    /**
     * 处理响应结果
     *
     * @param response SearchResponse
     */
    private void handleResponse(SearchResponse response)
    {
        SearchHits searchHits = response.getHits();
        // 4.1.总条数
        long total = searchHits.getTotalHits().value;
        System.out.println("总条数: " + total);
        // 4.2.获取文档数组
        SearchHit[] hits = searchHits.getHits();
        // 4.3.遍历
        for (SearchHit hit : hits)
        {
            // 4.4.获取source
            String json = hit.getSourceAsString();
            // 4.5.反序列化，非高亮的
            HotelDoc hotelDoc = JSON.parseObject(json, HotelDoc.class);
            // 4.6.处理高亮结果
            // 1)获取高亮map
            Map<String, HighlightField> map = hit.getHighlightFields();
            // 2）根据字段名，获取高亮结果
            HighlightField highlightField = map.get("name");
            if (highlightField != null)
            {
                // 3）获取高亮结果字符串数组中的第1个元素
                String hName = highlightField.getFragments()[0].toString();
                // 4）把高亮结果放到HotelDoc中
                hotelDoc.setName(hName);
            }
            // 4.7.打印
            System.out.println(hotelDoc);
        }
    }
}
```

搜索和分页

- 请求方式：POST
- 请求路径：/hotel/list
- 请求参数：JSON对象，包含4个字段：
    - key：搜索关键字
    - page：页码

  - size：每页大小
  - sortBy：排序，目前暂不实现
- 返回值：分页查询，需要返回分页结果PageResult，包含两个属性：
  - `total`：总条数
  - `List<HotelDoc>`：当前页的数据

实体类

前端请求的json结构如下：

```json
{
    "key": "搜索关键字",
    "page": 1,
    "size": 3,
    "sortBy": "default"
}
```

定义一个实体类 RequestParams：

```java
package mao.elasticsearch_hotel.entity;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.entity
 * Class(类名): RequestParams
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 15:07
 * Version(版本): 1.0
 * Description(描述)： 请求参数实体类
 */

public class RequestParams
{
    private String key;
    private Integer page;
    private Integer size;
    private String sortBy;
    private String brand;
    private String city;
    private String starName;
    private Integer minPrice;
    private Integer maxPrice;
    private String location;

    /**
     * Instantiates a new Request params.
     */
    public RequestParams()
    {
    }

    /**
     * Instantiates a new Request params.
     *
     * @param key      the key
     * @param page     the page
     * @param size     the size
     * @param sortBy   the sort by
     * @param brand    the brand
     * @param city     the city
     * @param starName the star name
     * @param minPrice the min price
     * @param maxPrice the max price
     * @param location the location
     */
    public RequestParams(String key, Integer page, Integer size, String sortBy,
String brand,
                         String city, String starName, Integer minPrice, Integer
maxPrice, String location)
    {
        this.key = key;
        this.page = page;
        this.size = size;
        this.sortBy = sortBy;
        this.brand = brand;
        this.city = city;
        this.starName = starName;
        this.minPrice = minPrice;
        this.maxPrice = maxPrice;
        this.location = location;
    }

    /**
     * Gets key.
     *
     * @return the key
     */
    public String getKey()
    {
        return key;
    }

    /**
     * Sets key.
     *
     * @param key the key
     */
    public void setKey(String key)
    {
        this.key = key;
    }

    /**
     * Gets page.
     *
     * @return the page
     */
    public Integer getPage()
    {
        return page;
    }

    /**
     * Sets page.
     *
     * @param page the page
     */
    public void setPage(Integer page)
    {
        this.page = page;
    }

    /**
     * Gets size.
     *
     * @return the size
     */
    public Integer getSize()
    {
        return size;
    }

    /**
     * Sets size.
     *
     * @param size the size
     */
    public void setSize(Integer size)
    {
        this.size = size;
    }

    /**
     * Gets sort by.
     *
     * @return the sort by
     */
    public String getSortBy()
    {
        return sortBy;
    }

    /**
     * Sets sort by.
     *
     * @param sortBy the sort by
     */
    public void setSortBy(String sortBy)
    {
        this.sortBy = sortBy;
    }

    /**
```

```
     * Gets brand.
     *
     * @return the brand
     */
    public String getBrand()
    {
        return brand;
    }

    /**
     * Sets brand.
     *
     * @param brand the brand
     */
    public void setBrand(String brand)
    {
        this.brand = brand;
    }

    /**
     * Gets city.
     *
     * @return the city
     */
    public String getCity()
    {
        return city;
    }

    /**
     * Sets city.
     *
     * @param city the city
     */
    public void setCity(String city)
    {
        this.city = city;
    }

    /**
     * Gets star name.
     *
     * @return the star name
     */
    public String getStarName()
    {
        return starName;
    }

    /**
     * Sets star name.
     *
     * @param starName the star name
     */
    public void setStarName(String starName)
    {
        this.starName = starName;
    }
    /**
     * Gets min price.
     *
     * @return the min price
     */
    public Integer getMinPrice()
    {
        return minPrice;
    }

    /**
     * Sets min price.
     *
     * @param minPrice the min price
     */
    public void setMinPrice(Integer minPrice)
    {
        this.minPrice = minPrice;
    }

    /**
     * Gets max price.
     *
     * @return the max price
     */
    public Integer getMaxPrice()
    {
        return maxPrice;
    }

    /**
     * Sets max price.
     *
     * @param maxPrice the max price
     */
    public void setMaxPrice(Integer maxPrice)
    {
        this.maxPrice = maxPrice;
    }

    /**
     * Gets location.
     *
     * @return the location
     */
    public String getLocation()
    {
        return location;
    }

    /**
     * Sets location.
     *
     * @param location the location
     */
    public void setLocation(String location)
    {
        this.location = location;
    }

    @Override
    @SuppressWarnings("all")
    public String toString()
    {
        final StringBuilder stringbuilder = new StringBuilder();
        stringbuilder.append("key: ").append(key).append('\n');
        stringbuilder.append("page: ").append(page).append('\n');
        stringbuilder.append("size: ").append(size).append('\n');
        stringbuilder.append("sortBy: ").append(sortBy).append('\n');
        stringbuilder.append("brand: ").append(brand).append('\n');
        stringbuilder.append("city: ").append(city).append('\n');
        stringbuilder.append("starName: ").append(starName).append('\n');
        stringbuilder.append("minPrice: ").append(minPrice).append('\n');
        stringbuilder.append("maxPrice: ").append(maxPrice).append('\n');
        stringbuilder.append("location: ").append(location).append('\n');
        return stringbuilder.toString();
    }
}
```

返回值

分页查询，需要返回分页结果PageResult，包含两个属性：

- `total`：总条数
- `List<HotelDoc>`：当前页的数据

定义返回结果

```
package mao.elasticsearch_hotel.entity;

import java.util.List;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.entity
 * Class(类名): PageResult
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 15:06
 * Version(版本): 1.0
 * Description(描述)： 返回的数据实体类
 */


public class PageResult
{
    private Long total;
    private List<HotelDoc> hotels;

    /**
     * Instantiates a new Page result.
     */
    public PageResult()
    {
    }

    /**
     * Instantiates a new Page result.
     *
     * @param total  the total
     * @param hotels the hotels
     */
    public PageResult(Long total, List<HotelDoc> hotels)
    {
        this.total = total;
        this.hotels = hotels;
    }

    /**
     * Gets total.
     *
     * @return the total
     */
    public Long getTotal()
    {
        return total;
    }

    /**
     * Sets total.
     *
     * @param total the total
     */
    public void setTotal(Long total)
    {
        this.total = total;
    }

    /**
     * Gets hotels.
     *
     * @return the hotels
     */
    public List<HotelDoc> getHotels()
    {
        return hotels;
    }

    /**
     * Sets hotels.
     *
     * @param hotels the hotels
     */
    public void setHotels(List<HotelDoc> hotels)
    {
        this.hotels = hotels;
    }
}
```

定义controller

定义一个HotelController，声明查询接口，满足下列要求：

- 请求方式：Post
- 请求路径：/hotel/list
- 请求参数：对象，类型为RequestParam
- 返回值：PageResult，包含两个属性
    - `Long total`：总条数
    - `List<HotelDoc> hotels`：酒店数据

```java
package mao.elasticsearch_hotel.controller;

import mao.elasticsearch_hotel.entity.PageResult;
import mao.elasticsearch_hotel.entity.RequestParams;
import mao.elasticsearch_hotel.service.IHotelService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.controller
 * Class(类名): HotelController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 14:53
 * Version(版本): 1.0
 * Description(描述)： Controller
 */

@RestController
@RequestMapping("hotel")
public class HotelController
{
    @Resource
    private IHotelService hotelService;

    /**
     * 获取hotel列表
     *
     * @param params 参数
     * @return PageResult
     */
    @PostMapping("list")
    public PageResult search(@RequestBody RequestParams params)
    {
        return hotelService.search(params);
    }
}
```

实现搜索业务

```
package mao.elasticsearch_hotel.service;

import com.baomidou.mybatisplus.extension.service.IService;
import mao.elasticsearch_hotel.entity.Hotel;
import mao.elasticsearch_hotel.entity.PageResult;
import mao.elasticsearch_hotel.entity.RequestParams;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.service
 * Class(类名): IHotelService
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 14:54
 * Version(版本): 1.0
 * Description(描述)： 接口
 */


public interface IHotelService extends IService<Hotel>
{
    /**
     * 搜索
     *
     * @param params 参数
     * @return PageResult
     */
    PageResult search(RequestParams params);
}
```

```
package mao.elasticsearch_hotel.service.impl;

import com.alibaba.fastjson.JSON;
```

```java
import com.baomidou.mybatisplus.core.toolkit.StringUtils;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import lombok.extern.slf4j.Slf4j;
import mao.elasticsearch_hotel.entity.Hotel;
import mao.elasticsearch_hotel.entity.HotelDoc;
import mao.elasticsearch_hotel.entity.PageResult;
import mao.elasticsearch_hotel.entity.RequestParams;
import mao.elasticsearch_hotel.mapper.HotelMapper;
import mao.elasticsearch_hotel.service.IHotelService;
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.geo.GeoPoint;
import org.elasticsearch.common.unit.DistanceUnit;
import org.elasticsearch.index.query.BoolQueryBuilder;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.index.query.functionscore.FunctionScoreQueryBuilder;
import org.elasticsearch.index.query.functionscore.ScoreFunctionBuilders;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.search.fetch.subphase.highlight.HighlightField;
import org.elasticsearch.search.sort.SortBuilders;
import org.elasticsearch.search.sort.SortOrder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.service.impl
 * Class(类名): HotelServiceImpl
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 14:58
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@Service
@Slf4j
public class HotelServiceImpl extends ServiceImpl<HotelMapper, Hotel> implements
IHotelService
{
    @Autowired
    private RestHighLevelClient restHighLevelClient;

    @Override
    public PageResult search(RequestParams params)
    {
        try
        {
            // 1.准备Request
            SearchRequest request = new SearchRequest("hotel");
            // 2.准备请求参数
            // 2.1.query
            buildBasicQuery(params, request);
            // 2.2.分页
            int page = params.getPage();
            int size = params.getSize();
            request.source().from((page - 1) * size).size(size);
            // 2.3.距离排序
            String location = params.getLocation();
            //判断距离信息是否为空
            if (StringUtils.isNotBlank(location))
            {
                //如果距离信息不为空，按距离排序
                request.source().sort(SortBuilders
                        .geoDistanceSort("location", new GeoPoint(location))
                        //升序
                        .order(SortOrder.ASC)
                        //单位为千米
                        .unit(DistanceUnit.KILOMETERS)
                );
            }
            // 3.发送请求
            SearchResponse response = restHighLevelClient.search(request,
RequestOptions.DEFAULT);
            // 4.解析响应
            return handleResponse(response);
        }
        catch (IOException e)
        {
            throw new RuntimeException("搜索数据失败", e);
        }
    }

    /**
     * 填充查询参数的方法
     *
     * @param params  RequestParams
     * @param request SearchRequest
     */
    private void buildBasicQuery(RequestParams params, SearchRequest request)
    {
        // 1.准备Boolean查询
        BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();

        // 1.1.关键字搜索，match查询，放到must中
        String key = params.getKey();
        //判断关键字是否为空
        if (StringUtils.isNotBlank(key))
        {
            // 不为空，根据关键字查询
            boolQuery.must(QueryBuilders.matchQuery("all", key));
        }
        else
        {
            // 为空，查询所有
            boolQuery.must(QueryBuilders.matchAllQuery());
        }

        // 1.2.品牌
        String brand = params.getBrand();
        //判断品牌信息是否为空
        if (StringUtils.isNotBlank(brand))
        {
            //品牌信息不为空，过滤掉其它品牌，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("brand", brand));
        }
        // 1.3.城市
        String city = params.getCity();
        //判断城市是否为空
        if (StringUtils.isNotBlank(city))
        {
            //不为空，过滤掉其它城市，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("city", city));
        }
        // 1.4.星级
        String starName = params.getStarName();
        //判断星级信息是否为空
        if (StringUtils.isNotBlank(starName))
        {
            //不为空，过滤掉其它星级，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("starName", starName));
        }
        // 1.5.价格范围
        //最小价格
        Integer minPrice = params.getMinPrice();
        //最大价格
        Integer maxPrice = params.getMaxPrice();
        //判断是否选择了价格区间
        if (minPrice != null && maxPrice != null)
        {
            //选择了价格区间
            //是否有最大值，没有最大值，就是xxx元以上
            maxPrice = maxPrice == 0 ? Integer.MAX_VALUE : maxPrice;
            //设置价格区间

 boolQuery.filter(QueryBuilders.rangeQuery("price").gte(minPrice).lte(maxPrice))
;
        }

        // 2.算分函数查询
        FunctionScoreQueryBuilder functionScoreQuery =
QueryBuilders.functionScoreQuery(
                boolQuery, // 原始查询，boolQuery
                new FunctionScoreQueryBuilder.FilterFunctionBuilder[]
                        { // function数组
                                new
FunctionScoreQueryBuilder.FilterFunctionBuilder(
                                        //判断是否是广告，如果是，分数加10倍
                                        QueryBuilders.termQuery("isAD", true),
// 过滤条件

 ScoreFunctionBuilders.weightFactorFunction(10) // 算分函数
                                )
                        }
```

```java
        );

        // 3.设置查询条件
        request.source().query(functionScoreQuery);
    }

    /**
     * 处理响应信息的方法
     *
     * @param response SearchResponse
     * @return PageResult
     */
    private PageResult handleResponse(SearchResponse response)
    {
        SearchHits searchHits = response.getHits();
        // 4.1.总条数
        long total = searchHits.getTotalHits().value;
        // 4.2.获取文档数组
        SearchHit[] hits = searchHits.getHits();
        // 4.3.遍历
        List<HotelDoc> hotels = new ArrayList<>(hits.length);
        for (SearchHit hit : hits)
        {
            // 4.4.获取source
            String json = hit.getSourceAsString();
            // 4.5.反序列化，非高亮的
            HotelDoc hotelDoc = JSON.parseObject(json, HotelDoc.class);
            // 4.6.处理高亮结果
            // 1)获取高亮map
            Map<String, HighlightField> map = hit.getHighlightFields();
            //判断集合是否为空，避免空指针
            if (map != null && !map.isEmpty())
            {
                // 2）根据字段名，获取高亮结果
                HighlightField highlightField = map.get("name");
                //判断高亮字段是否为空，避免空指针
                if (highlightField != null)
                {
                    // 3）获取高亮结果字符串数组中的第1个元素
                    String hName = highlightField.getFragments()[0].toString();
                    // 4）把高亮结果放到HotelDoc中，覆盖原有的数据
                    hotelDoc.setName(hName);
                }
            }
            // 4.8.排序信息
            Object[] sortValues = hit.getSortValues();
            if (sortValues.length > 0)
            {
                //设置距离
                hotelDoc.setDistance(sortValues[0]);
            }

            // 4.9.放入集合
            hotels.add(hotelDoc);
        }
        //返回数据
        return new PageResult(total, hotels);
    }
```

```
}
```

结果过滤

包含的过滤条件有：

- brand：品牌值
- city：城市
- minPrice~maxPrice：价格范围
- starName：星级

在之前的业务中，只有match查询，根据关键字搜索，现在要添加条件过滤，包括：

- 品牌过滤：是keyword类型，用term查询
- 星级过滤：是keyword类型，用term查询
- 价格过滤：是数值类型，用range查询
- 城市过滤：是keyword类型，用term查询

多个查询条件组合，肯定是boolean查询来组合：

- 关键字搜索放到must中，参与算分
- 其它过滤条件放到filter中，不参与算分

我周边的酒店

- 修改RequestParams参数，接收location字段
- 修改search方法业务逻辑，如果location有值，添加根据geo_distance排序的功能

地理坐标排序：

```
GET /indexName/_search
{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "price": "asc"
    },
    {
      "_geo_distance" : {
          "FIELD" : "纬度，经度",
          "order" : "asc",
          "unit" : "km"
      }
      }
    ]
  }
```

排序距离显示

- 修改HotelDoc，添加排序距离字段，用于页面显示
- 修改HotelService类中的handleResponse方法，添加对sort值的获取

酒店竞价排名

让指定的酒店在搜索结果中排名置顶

function_score查询可以影响算分，算分高了，自然排名也就高了。而function_score包含3个要素：

- 过滤条件：哪些文档要加分
- 算分函数：如何计算function score
- 加权方式：function score 与 query score如何运算

这里的需求是：让**指定酒店**排名靠前。因此我们需要给这些酒店添加一个标记，这样在过滤条件中就可以**根据这个标记来判断，是否要提高算分。**

比如，我们给酒店添加一个字段：isAD，Boolean类型：

- true：是广告
- false：不是广告

这样function_score包含3个要素就很好确定了：

- 过滤条件：判断isAD 是否为true
- 算分函数：我们可以用最简单暴力的weight，固定加权值
- 加权方式：可以用默认的相乘，大大提高算分

因此，业务的实现步骤包括：

1. 给HotelDoc类添加isAD字段，Boolean类型
2. 挑选几个你喜欢的酒店，给它的文档数据添加isAD字段，值为true
3. 修改search方法，添加function score功能，给isAD值为true的酒店增加权重

添加广告标记

```
POST /hotel/_update/1902197537
{
    "doc": {
        "isAD": true
    }
}
POST /hotel/_update/2056126831
{
    "doc": {
        "isAD": true
    }
}
POST /hotel/_update/1989806195
{
    "doc": {
        "isAD": true
    }
}
POST /hotel/_update/2056105938
{
    "doc": {
        "isAD": true
    }
}

POST /hotel/_update/60214
{
    "doc": {
        "isAD": true
    }
}
POST /hotel/_update/60223
{
    "doc": {
        "isAD": true
    }
}
POST /hotel/_update/60398
{
    "doc": {
        "isAD": true
    }
}
POST /hotel/_update/60916
{
    "doc": {
        "isAD": true
    }
}
```

添加算分函数查询

实现酒店搜索框自动补全

1. 修改hotel索引库结构，设置自定义拼音分词器
2. 修改索引库的name、all字段，使用自定义分词器
3. 索引库添加一个新字段suggestion，类型为completion类型，使用自定义的分词器
4. 给HotelDoc类添加suggestion字段，内容包含brand、business
5. 重新导入数据到hotel库

```
PUT /hotel
{
  "settings": {
    "analysis": {
      "analyzer": {
        "text_anlyzer": {
          "tokenizer": "ik_max_word",
          "filter": "py"
        },
        "completion_analyzer": {
          "tokenizer": "keyword",
          "filter": "py"
        }
      },
      "filter": {
        "py": {
          "type": "pinyin",
          "keep_full_pinyin": false,
          "keep_joined_full_pinyin": true,
          "keep_original": true,
          "limit_first_letter_length": 16,
          "remove_duplicated_term": true,
          "none_chinese_pinyin_tokenize": false
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "id":{
        "type": "keyword"
      },
      "name":{
        "type": "text",
        "analyzer": "text_anlyzer",
        "search_analyzer": "ik_smart",
        "copy_to": "all"
      },
      "address":{
        "type": "keyword",
        "index": false
      },
      "price":{
        "type": "integer"
      },
      "score":{
        "type": "integer"
      },
      "brand":{
          "type": "keyword",
          "copy_to": "all"
        },
        "city":{
          "type": "keyword"
        },
        "starName":{
          "type": "keyword"
        },
        "business":{
          "type": "keyword",
          "copy_to": "all"
        },
        "location":{
          "type": "geo_point"
        },
        "pic":{
          "type": "keyword",
          "index": false
        },
        "all":{
          "type": "text",
          "analyzer": "text_anlyzer",
          "search_analyzer": "ik_smart"
        },
        "suggestion":{
            "type": "completion",
            "analyzer": "completion_analyzer"
        }
      }
    }
}
```

修改HotelDoc实体

HotelDoc中要添加一个字段，用来做自动补全，内容可以是酒店品牌、城市、商圈等信息。按照自动补全字段的要求，最好是这些字段的数组。

因此我们在HotelDoc中添加一个suggestion字段，类型为`List<String>`，然后将brand、city、business等信息放到里面。

重新导入

重新执行之前编写的导入数据功能

实现搜索框自动补全

```
package mao.elasticsearch_hotel.controller;
```

```java
import mao.elasticsearch_hotel.entity.PageResult;
import mao.elasticsearch_hotel.entity.RequestParams;
import mao.elasticsearch_hotel.service.IHotelService;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import java.util.List;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.controller
 * Class(类名): HotelController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 14:53
 * Version(版本): 1.0
 * Description(描述)： Controller
 */

@RestController
@RequestMapping("hotel")
public class HotelController
{
    @Resource
    private IHotelService hotelService;

    /**
     * 获取hotel列表
     *
     * @param params 参数
     * @return PageResult
     */
    @PostMapping("list")
    public PageResult search(@RequestBody RequestParams params)
    {
        return hotelService.search(params);
    }

    /**
     * 自动补全功能
     *
     * @param prefix 要自动补全的前缀或者关键字
     * @return 符合条件的列表
     */
    @GetMapping("suggestion")
    public List<String> getSuggestions(@RequestParam("key") String prefix)
    {
        return hotelService.getSuggestions(prefix);
    }
}
```

接口：

```java
package mao.elasticsearch_hotel.service;

import com.baomidou.mybatisplus.extension.service.IService;
import mao.elasticsearch_hotel.entity.Hotel;
import mao.elasticsearch_hotel.entity.PageResult;
import mao.elasticsearch_hotel.entity.RequestParams;

import java.util.List;

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.service
 * Class(类名): IHotelService
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 14:54
 * Version(版本): 1.0
 * Description(描述)： 接口
 */


public interface IHotelService extends IService<Hotel>
{
    /**
     * 搜索
     *
     * @param params 参数
     * @return PageResult
     */
    PageResult search(RequestParams params);

    /**
     * 自动补全功能
     *
     * @param prefix 要自动补全的前缀或者关键字
     * @return 符合条件的列表
     */
    List<String> getSuggestions(String prefix);
}
```

实现类：

```java
package mao.elasticsearch_hotel.service.impl;

import com.alibaba.fastjson.JSON;
import com.baomidou.mybatisplus.core.toolkit.StringUtils;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import lombok.extern.slf4j.Slf4j;
import mao.elasticsearch_hotel.entity.Hotel;
import mao.elasticsearch_hotel.entity.HotelDoc;
import mao.elasticsearch_hotel.entity.PageResult;
import mao.elasticsearch_hotel.entity.RequestParams;
import mao.elasticsearch_hotel.mapper.HotelMapper;
import mao.elasticsearch_hotel.service.IHotelService;
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.geo.GeoPoint;
import org.elasticsearch.common.unit.DistanceUnit;
import org.elasticsearch.index.query.BoolQueryBuilder;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.index.query.functionscore.FunctionScoreQueryBuilder;
import org.elasticsearch.index.query.functionscore.ScoreFunctionBuilders;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.fetch.subphase.highlight.HighlightField;
import org.elasticsearch.search.sort.SortBuilders;
import org.elasticsearch.search.sort.SortOrder;
import org.elasticsearch.search.suggest.Suggest;
import org.elasticsearch.search.suggest.SuggestBuilder;
import org.elasticsearch.search.suggest.SuggestBuilders;
import org.elasticsearch.search.suggest.completion.CompletionSuggestion;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.service.impl
 * Class(类名): HotelServiceImpl
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 14:58
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@Service
@Slf4j
public class HotelServiceImpl extends ServiceImpl<HotelMapper, Hotel> implements
IHotelService
{
    @Autowired
    private RestHighLevelClient restHighLevelClient;

    @Override
    public PageResult search(RequestParams params)
    {
        try
        {
            // 1.准备Request
            SearchRequest request = new SearchRequest("hotel");
```

```java
            // 2.准备请求参数
            // 2.1.query
            buildBasicQuery(params, request);
            // 2.2.分页
            int page = params.getPage();
            int size = params.getSize();
            request.source().from((page - 1) * size).size(size);
            // 2.3.距离排序
            String location = params.getLocation();
            //判断距离信息是否为空
            if (StringUtils.isNotBlank(location))
            {
                //如果距离信息不为空，按距离排序
                request.source().sort(SortBuilders
                        .geoDistanceSort("location", new GeoPoint(location))
                        //升序
                        .order(SortOrder.ASC)
                        //单位为千米
                        .unit(DistanceUnit.KILOMETERS)
                );
            }
            // 3.发送请求
            SearchResponse response = restHighLevelClient.search(request,
RequestOptions.DEFAULT);
            // 4.解析响应
            return handleResponse(response);
        }
        catch (IOException e)
        {
            throw new RuntimeException("搜索数据失败", e);
        }
    }

    @Override
    public List<String> getSuggestions(String prefix)
    {
        try
        {
            //构建搜索请求
            SearchRequest searchRequest = new SearchRequest("hotel");
            //构建请求体
            SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
            //自动补全
            searchSourceBuilder.suggest(new SuggestBuilder()
                    .addSuggestion("suggestions",
                            SuggestBuilders.completionSuggestion("suggestion")
                                    .prefix(prefix)
                                    .size(10)
                                    .skipDuplicates(true)));
            //放入到请求中
            searchRequest.source(searchSourceBuilder);
            //发起请求
            SearchResponse searchResponse =
restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
            //获取数据
            //获取补全部分
            Suggest suggest = searchResponse.getSuggest();
            CompletionSuggestion suggestions =
suggest.getSuggestion("suggestions");
            List<CompletionSuggestion.Entry.Option> options =
suggestions.getOptions();
            //遍历数据
            List<String> list = new ArrayList<>(options.size());
            for (CompletionSuggestion.Entry.Option option : options)
            {
                //获取文本信息
                String text = option.getText().string();
                //加入到集合中
                list.add(text);
            }
            //返回数据
            return list;
        }
        catch (Exception e)
        {
            throw new RuntimeException(e);
        }
    }

    /**
     * 填充查询参数的方法
     *
     * @param params  RequestParams
     * @param request SearchRequest
     */
    private void buildBasicQuery(RequestParams params, SearchRequest request)
    {
        // 1.准备Boolean查询
        BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();

        // 1.1.关键字搜索，match查询，放到must中
        String key = params.getKey();
        //判断关键字是否为空
        if (StringUtils.isNotBlank(key))
        {
            // 不为空，根据关键字查询
            boolQuery.must(QueryBuilders.matchQuery("all", key));
        }
        else
        {
            // 为空，查询所有
            boolQuery.must(QueryBuilders.matchAllQuery());
        }

        // 1.2.品牌
        String brand = params.getBrand();
        //判断品牌信息是否为空
        if (StringUtils.isNotBlank(brand))
        {
            //品牌信息不为空，过滤掉其它品牌，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("brand", brand));
        }
        // 1.3.城市
        String city = params.getCity();
        //判断城市是否为空
        if (StringUtils.isNotBlank(city))
        {
            //不为空，过滤掉其它城市，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("city", city));
        }
        // 1.4.星级
        String starName = params.getStarName();
        //判断星级信息是否为空
        if (StringUtils.isNotBlank(starName))
        {
            //不为空，过滤掉其它星级，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("starName", starName));
        }
        // 1.5.价格范围
        //最小价格
        Integer minPrice = params.getMinPrice();
        //最大价格
        Integer maxPrice = params.getMaxPrice();
        //判断是否选择了价格区间
        if (minPrice != null && maxPrice != null)
        {
            //选择了价格区间
            //是否有最大值，没有最大值，就是xxx元以上
            maxPrice = maxPrice == 0 ? Integer.MAX_VALUE : maxPrice;
            //设置价格区间

 boolQuery.filter(QueryBuilders.rangeQuery("price").gte(minPrice).lte(maxPrice))
;
        }

        // 2.算分函数查询
        FunctionScoreQueryBuilder functionScoreQuery =
QueryBuilders.functionScoreQuery(
                boolQuery, // 原始查询，boolQuery
                new FunctionScoreQueryBuilder.FilterFunctionBuilder[]
                        { // function数组
                                new
FunctionScoreQueryBuilder.FilterFunctionBuilder(
                                        //判断是否是广告，如果是，分数加10倍
                                        QueryBuilders.termQuery("isAD", true),
// 过滤条件

 ScoreFunctionBuilders.weightFactorFunction(10) // 算分函数
                                )
                        }
        );

        // 3.设置查询条件
        request.source().query(functionScoreQuery);
    }

    /**
     * 处理响应信息的方法
     *
     * @param response SearchResponse
     * @return PageResult
     */
    private PageResult handleResponse(SearchResponse response)
    {
        SearchHits searchHits = response.getHits();
        // 4.1.总条数
        long total = searchHits.getTotalHits().value;
        // 4.2.获取文档数组
        SearchHit[] hits = searchHits.getHits();
        // 4.3.遍历
        List<HotelDoc> hotels = new ArrayList<>(hits.length);
        for (SearchHit hit : hits)
        {
            // 4.4.获取source
            String json = hit.getSourceAsString();
            // 4.5.反序列化，非高亮的
            HotelDoc hotelDoc = JSON.parseObject(json, HotelDoc.class);
            // 4.6.处理高亮结果
            // 1)获取高亮map
            Map<String, HighlightField> map = hit.getHighlightFields();
            //判断集合是否为空，避免空指针
            if (map != null && !map.isEmpty())
            {
                // 2）根据字段名，获取高亮结果
                HighlightField highlightField = map.get("name");
                //判断高亮字段是否为空，避免空指针
                if (highlightField != null)
                {
                    // 3）获取高亮结果字符串数组中的第1个元素
                    String hName = highlightField.getFragments()[0].toString();
                    // 4）把高亮结果放到HotelDoc中，覆盖原有的数据
                    hotelDoc.setName(hName);
                }
            }
            // 4.8.排序信息
            Object[] sortValues = hit.getSortValues();
            if (sortValues.length > 0)
            {
                //设置距离
                hotelDoc.setDistance(sortValues[0]);
            }

            // 4.9.放入集合
            hotels.add(hotelDoc);
        }
        //返回数据
        return new PageResult(total, hotels);
    }
}
```

数据同步

酒店后台数据：

HotelController：

```java
package mao.elasticsearch_hotel_management.controller;

import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
import mao.elasticsearch_hotel_management.entity.Hotel;
import mao.elasticsearch_hotel_management.entity.PageResult;
import mao.elasticsearch_hotel_management.service.IHotelService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.security.InvalidParameterException;

/**
 * Project name(项目名称)：elasticsearch_hotel_management
 * Package(包名): mao.elasticsearch_hotel_management.controller
 * Class(类名): HotelController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/4
 * Time(创建时间)： 13:45
 * Version(版本): 1.0
 * Description(描述)： HotelController
 */

@RestController
@RequestMapping("hotel")
public class HotelController
{
    @Autowired
    private IHotelService hotelService;


    /**
     * 根据id查询酒店数据
     *
     * @param id id
     * @return Hotel
     */
    @GetMapping("/{id}")
    public Hotel queryById(@PathVariable("id") Long id)
    {
        return hotelService.getById(id);
    }

    /**
     * 获得酒店数据，分页
     *
     * @param page 当前页
     * @param size 页大小
     * @return PageResult
     */
    @GetMapping("/list")
    public PageResult hotelList
    (
            @RequestParam(value = "page", defaultValue = "1") Integer page,
            @RequestParam(value = "size", defaultValue = "1") Integer size
    )
    {
        Page<Hotel> result = hotelService.page(new Page<>(page, size));
        return new PageResult(result.getTotal(), result.getRecords());
    }

    /**
     * 保存或者添加一条酒店信息
     *
     * @param hotel Hotel
     */
    @PostMapping
    public void saveHotel(@RequestBody Hotel hotel)
    {
        hotelService.saveHotel(hotel);
    }

    /**
     * 更新酒店数据
     *
     * @param hotel Hotel
     */
    @PutMapping()
    public void updateById(@RequestBody Hotel hotel)
    {
        hotelService.updateHotelById(hotel);
    }

    /**
     * 删除酒店数据
     *
     * @param id 要删除的酒店id
     */
    @DeleteMapping("/{id}")
    public void deleteById(@PathVariable("id") Long id)
    {
        this.hotelService.deleteHotelById(id);
    }
}
```

接口:

```
package mao.elasticsearch_hotel_management.service;

import com.baomidou.mybatisplus.extension.service.IService;
import mao.elasticsearch_hotel_management.entity.Hotel;

/**
 * Project name(项目名称): elasticsearch_hotel_management
 * Package(包名): mao.elasticsearch_hotel_management.service
 * Interface(接口名): IHotelService
 * Author(作者）: mao
 * Author QQ: 1296193245
 * GitHub: https://github.com/maomao124/
 * Date(创建日期)： 2022/6/4
 * Time(创建时间)： 13:46
 * Version(版本): 1.0
 * Description(描述)： 接口
 */

public interface IHotelService extends IService<Hotel>
{

    /**
     * 删除酒店数据
     *
     * @param id 要删除的酒店id
     */
    void deleteHotelById(Long id);

    /**
     * 更新酒店数据
     *
     * @param hotel Hotel
     */
    void updateHotelById(Hotel hotel);

    /**
     * 保存或者添加一条酒店信息
     *
     * @param hotel Hotel
     */
    void saveHotel(Hotel hotel);
}
```

实现类：

```
package mao.elasticsearch_hotel_management.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import mao.elasticsearch_hotel_management.constants.RabbitMQConstants;
import mao.elasticsearch_hotel_management.entity.Hotel;
import mao.elasticsearch_hotel_management.mapper.HotelMapper;
import mao.elasticsearch_hotel_management.service.IHotelService;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;
import java.security.InvalidParameterException;

/**
 * Project name(项目名称)：elasticsearch_hotel_management
 * Package(包名): mao.elasticsearch_hotel_management.service.impl
 * Class(类名): HotelServiceImpl
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期):  2022/6/4
 * Time(创建时间):  13:53
 * Version(版本): 1.0
 * Description(描述):  HotelService实现类
 */

@Service
public class HotelServiceImpl extends ServiceImpl<HotelMapper, Hotel> implements
IHotelService
{

    @Resource
    private RabbitTemplate rabbitTemplate;


    @Override
    public void deleteHotelById(Long id)
    {
        this.removeById(id);
        //发送MQ消息
        rabbitTemplate.convertAndSend(RabbitMQConstants.EXCHANGE_NAME,
RabbitMQConstants.DELETE_KEY, id);
    }

    @Override
    public void updateHotelById(Hotel hotel)
    {
        if (hotel.getId() == null)
        {
            throw new InvalidParameterException("id不能为空");
        }
        boolean b = this.updateById(hotel);

        //判断是否成功
        if (b)
        {
            //发送MQ消息
            rabbitTemplate.convertAndSend(RabbitMQConstants.EXCHANGE_NAME,
RabbitMQConstants.INSERT_KEY, hotel.getId());
        }
    }

    @Override
    public void saveHotel(Hotel hotel)
    {
        // 新增酒店
        boolean save = this.save(hotel);
        if (save)
        {
            // 发送MQ消息
            rabbitTemplate.convertAndSend(RabbitMQConstants.EXCHANGE_NAME,
RabbitMQConstants.INSERT_KEY, hotel.getId());
        }
    }
}
```

其它的略

声明队列交换机名称

```java
package mao.elasticsearch_hotel_management.constants;

/**
 * Project name(项目名称)：elasticsearch_hotel_management
 * Package(包名): mao.elasticsearch_hotel_management.constants
 * Class(类名): RabbitMQConstants
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/4
 * Time(创建时间)： 13:57
 * Version(版本): 1.0
 * Description(描述)： 存放RabbitMQ常量
 */

public class RabbitMQConstants
{
    public static final String EXCHANGE_NAME = "hotel.exchange";
    public static final String INSERT_QUEUE_NAME = "hotel.insert.queue";
    public static final String DELETE_QUEUE_NAME = "hotel.delete.queue";
    public static final String INSERT_KEY = "hotel.insert";
    public static final String DELETE_KEY = "hotel.delete";
}
```

```java
package mao.elasticsearch_hotel_management.config;

import mao.elasticsearch_hotel_management.constants.RabbitMQConstants;
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.DirectExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * Project name(项目名称)：elasticsearch_hotel_management
 * Package(包名): mao.elasticsearch_hotel_management.config
 * Class(类名): RabbitMQConfig
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/4
 * Time(创建时间)： 14:00
 * Version(版本): 1.0
 * Description(描述):  RabbitMQConfig
 * 不考虑消息丢失问题，不考虑持久化，不考虑重复消费
 */

@Configuration
public class RabbitMQConfig
{
    /**
     * 声明交换机 ，直接交换机
     *
     * @return DirectExchange
     */
    @Bean
    public DirectExchange directExchange()
    {
        return new DirectExchange(RabbitMQConstants.EXCHANGE_NAME, false,
false);
    }

    /**
     * 声明队列 ，用于添加和更新酒店信息
     *
     * @return Queue
     */
    @Bean
    public Queue insertAndUpdateQueue()
    {
        return new Queue(RabbitMQConstants.INSERT_QUEUE_NAME, false, false,
false, null);
    }

    /**
     * 声明队列 ，用于删除酒店信息
     *
     * @return Queue
     */
    @Bean
    public Queue deleteQueue()
    {
        return new Queue(RabbitMQConstants.DELETE_QUEUE_NAME, false, false,
false, null);
    }

    /**
     * insertAndUpdateQueue绑定交换机
     *
     * @return Binding
     */
    @Bean
    public Binding insertAndUpdateQueue_binding_directExchange()
    {
        return
BindingBuilder.bind(insertAndUpdateQueue()).to(directExchange()).with(RabbitMQCo
nstants.INSERT_KEY);
    }

    /**
     * deleteQueue绑定交换机
```

```
      *
      * @return Binding
      */
     @Bean
     public Binding deleteQueue_binding_directExchange()
     {
         return
 BindingBuilder.bind(deleteQueue()).to(directExchange()).with(RabbitMQConstants.D
 ELETE_KEY);
     }
 }
```

接收MQ消息

接口:

```
package mao.elasticsearch_hotel.service;

import com.baomidou.mybatisplus.extension.service.IService;
import mao.elasticsearch_hotel.entity.Hotel;
import mao.elasticsearch_hotel.entity.PageResult;
import mao.elasticsearch_hotel.entity.RequestParams;

import java.util.List;
import java.util.Map;

/**
 * Project name(项目名称): elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.service
 * Class(类名): IHotelService
 * Author(作者）: mao
 * Author QQ: 1296193245
 * GitHub: https://github.com/maomao124/
 * Date(创建日期):  2022/6/2
 * Time(创建时间):  14:54
 * Version(版本): 1.0
 * Description(描述):  接口
 */


public interface IHotelService extends IService<Hotel>
{
    /**
     * 搜索
     *
     * @param params 参数
     * @return PageResult
     */
    PageResult search(RequestParams params);

    /**
     * 自动补全功能
     *
     * @param prefix 要自动补全的前缀或者关键字
     * @return 符合条件的列表
     */
    List<String> getSuggestions(String prefix);

    /**
     * 添加ElasticSearch数据
     *
     * @param id 要添加的id号，信息从数据库里根据id查
     */
    void insertElasticSearchHotelById(Long id);

    /**
     * 删除ElasticSearch里某个id的数据
     *
     * @param id id
     */
    void deleteElasticSearchHotelById(Long id);

    /**
     * 聚合
     *
     * @param params RequestParams
     * @return Map<String, List < String>>
     */
    Map<String, List<String>> getFilters(RequestParams params);
}
```

实现类:

```
package mao.elasticsearch_hotel.service.impl;

import com.alibaba.fastjson.JSON;
import com.baomidou.mybatisplus.core.toolkit.StringUtils;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import lombok.extern.slf4j.Slf4j;
import mao.elasticsearch_hotel.entity.Hotel;
import mao.elasticsearch_hotel.entity.HotelDoc;
import mao.elasticsearch_hotel.entity.PageResult;
import mao.elasticsearch_hotel.entity.RequestParams;
import mao.elasticsearch_hotel.mapper.HotelMapper;
import mao.elasticsearch_hotel.service.IHotelService;
import org.elasticsearch.action.delete.DeleteRequest;
import org.elasticsearch.action.delete.DeleteResponse;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.index.IndexResponse;
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.geo.GeoPoint;
import org.elasticsearch.common.unit.DistanceUnit;
import org.elasticsearch.index.query.BoolQueryBuilder;
```

```java
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.index.query.functionscore.FunctionScoreQueryBuilder;
import org.elasticsearch.index.query.functionscore.ScoreFunctionBuilders;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.search.aggregations.AggregationBuilders;
import org.elasticsearch.search.aggregations.Aggregations;
import org.elasticsearch.search.aggregations.bucket.terms.Terms;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.fetch.subphase.highlight.HighlightField;
import org.elasticsearch.search.sort.SortBuilders;
import org.elasticsearch.search.sort.SortOrder;
import org.elasticsearch.search.suggest.Suggest;
import org.elasticsearch.search.suggest.SuggestBuilder;
import org.elasticsearch.search.suggest.SuggestBuilders;
import org.elasticsearch.search.suggest.completion.CompletionSuggestion;
import org.elasticsearch.xcontent.XContentType;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

/**
 * Project name(项目名称)：elasticsearch_hotel
 * Package(包名): mao.elasticsearch_hotel.service.impl
 * Class(类名): HotelServiceImpl
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/2
 * Time(创建时间)： 14:58
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@Service
@Slf4j
public class HotelServiceImpl extends ServiceImpl<HotelMapper, Hotel> implements
IHotelService
{
    @Autowired
    private RestHighLevelClient restHighLevelClient;

    @Override
    public PageResult search(RequestParams params)
    {
        try
        {
            // 1.准备Request
            SearchRequest request = new SearchRequest("hotel");
            // 2.准备请求参数
            // 2.1.query
            buildBasicQuery(params, request);
            // 2.2.分页
            int page = params.getPage();
            int size = params.getSize();
            request.source().from((page - 1) * size).size(size);
            //排序
            String sortBy = params.getSortBy();
            if (sortBy != null && !sortBy.equals("default"))
            {
                request.source().sort(sortBy, SortOrder.DESC);
            }
            // 2.3.距离排序
            String location = params.getLocation();
            //判断距离信息是否为空
            if (StringUtils.isNotBlank(location))
            {
                //如果距离信息不为空，按距离排序
                request.source().sort(SortBuilders
                        .geoDistanceSort("location", new GeoPoint(location))
                        //升序
                        .order(SortOrder.ASC)
                        //单位为千米
                        .unit(DistanceUnit.KILOMETERS)
                );
            }
            // 3.发送请求
            SearchResponse response = restHighLevelClient.search(request,
RequestOptions.DEFAULT);
            // 4.解析响应
            return handleResponse(response);
        }
        catch (IOException e)
        {
            throw new RuntimeException("搜索数据失败", e);
        }
    }

    @Override
    public List<String> getSuggestions(String prefix)
    {
        try
        {
            //构建搜索请求
            SearchRequest searchRequest = new SearchRequest("hotel");
            //构建请求体
            SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
            //自动补全
            searchSourceBuilder.suggest(new SuggestBuilder()
                    .addSuggestion("suggestions",
                            SuggestBuilders.completionSuggestion("suggestion")
                                    .prefix(prefix)
                                    .size(10)
                                    .skipDuplicates(true)));
            //放入到请求中
            searchRequest.source(searchSourceBuilder);
            //发起请求
            SearchResponse searchResponse =
restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
            //获取数据
            //获取补全部分
            Suggest suggest = searchResponse.getSuggest();
            CompletionSuggestion suggestions =
suggest.getSuggestion("suggestions");
            List<CompletionSuggestion.Entry.Option> options =
suggestions.getOptions();
            //遍历数据
            List<String> list = new ArrayList<>(options.size());
            for (CompletionSuggestion.Entry.Option option : options)
            {
                //获取文本信息
                String text = option.getText().string();
                //加入到集合中
                list.add(text);
            }
            //返回数据
            return list;
        }
        catch (Exception e)
        {
            throw new RuntimeException(e);
        }
    }


    /**
     * 填充查询参数的方法
     *
     * @param params  RequestParams
     * @param request SearchRequest
     */
    private void buildBasicQuery(RequestParams params, SearchRequest request)
    {
        // 1.准备Boolean查询
        BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();

        // 1.1.关键字搜索，match查询，放到must中
        String key = params.getKey();
        //判断关键字是否为空
        if (StringUtils.isNotBlank(key))
        {
            // 不为空，根据关键字查询
            boolQuery.must(QueryBuilders.matchQuery("all", key));
        }
        else
        {
            // 为空，查询所有
            boolQuery.must(QueryBuilders.matchAllQuery());
        }

        // 1.2.品牌
        String brand = params.getBrand();
        //判断品牌信息是否为空
        if (StringUtils.isNotBlank(brand))
        {
            //品牌信息不为空，过滤掉其它品牌，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("brand", brand));
        }
        // 1.3.城市
        String city = params.getCity();
        //判断城市是否为空
        if (StringUtils.isNotBlank(city))
        {
            //不为空，过滤掉其它城市，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("city", city));
        }
        // 1.4.星级
        String starName = params.getStarName();
        //判断星级信息是否为空
        if (StringUtils.isNotBlank(starName))
        {
            //不为空，过滤掉其它星级，不参与算分
            boolQuery.filter(QueryBuilders.termQuery("starName", starName));
        }
        // 1.5.价格范围
        //最小价格
        Integer minPrice = params.getMinPrice();
        //最大价格
        Integer maxPrice = params.getMaxPrice();
        //判断是否选择了价格区间
        if (minPrice != null && maxPrice != null)
        {
            //选择了价格区间
            //是否有最大值，没有最大值，就是xxx元以上
            maxPrice = maxPrice == 0 ? Integer.MAX_VALUE : maxPrice;
            //设置价格区间

 boolQuery.filter(QueryBuilders.rangeQuery("price").gte(minPrice).lte(maxPrice))
;
        }

        // 2.算分函数查询
        FunctionScoreQueryBuilder functionScoreQuery =
QueryBuilders.functionScoreQuery(
                boolQuery, // 原始查询，boolQuery
                new FunctionScoreQueryBuilder.FilterFunctionBuilder[]
                        { // function数组
                                new
FunctionScoreQueryBuilder.FilterFunctionBuilder(
                                        //判断是否是广告，如果是，分数加10倍
                                        QueryBuilders.termQuery("isAD", true),
// 过滤条件

 ScoreFunctionBuilders.weightFactorFunction(10) // 算分函数
                                )
                        }
        );

        // 3.设置查询条件
        request.source().query(functionScoreQuery);
    }

    /**
     * 处理响应信息的方法
     *
     * @param response SearchResponse
     * @return PageResult
```

```java
     */
    private PageResult handleResponse(SearchResponse response)
    {
        SearchHits searchHits = response.getHits();
        // 4.1.总条数
        long total = searchHits.getTotalHits().value;
        // 4.2.获取文档数组
        SearchHit[] hits = searchHits.getHits();
        // 4.3.遍历
        List<HotelDoc> hotels = new ArrayList<>(hits.length);
        for (SearchHit hit : hits)
        {
            // 4.4.获取source
            String json = hit.getSourceAsString();
            // 4.5.反序列化，非高亮的
            HotelDoc hotelDoc = JSON.parseObject(json, HotelDoc.class);
            // 4.6.处理高亮结果
            // 1)获取高亮map
            Map<String, HighlightField> map = hit.getHighlightFields();
            //判断集合是否为空，避免空指针
            if (map != null && !map.isEmpty())
            {
                // 2）根据字段名，获取高亮结果
                HighlightField highlightField = map.get("name");
                //判断高亮字段是否为空，避免空指针
                if (highlightField != null)
                {
                    // 3）获取高亮结果字符串数组中的第1个元素
                    String hName = highlightField.getFragments()[0].toString();
                    // 4）把高亮结果放到HotelDoc中，覆盖原有的数据
                    hotelDoc.setName(hName);
                }
            }
            // 4.8.排序信息
            Object[] sortValues = hit.getSortValues();
            if (sortValues.length > 0)
            {
                //设置距离
                hotelDoc.setDistance(sortValues[0]);
            }

            // 4.9.放入集合
            hotels.add(hotelDoc);
        }
        //返回数据
        return new PageResult(total, hotels);
    }

    @Override
    public void insertElasticSearchHotelById(Long id)
    {
        try
        {
            log.debug("开始添加或者更新ElasticSearch文档：" + id);
            //查询数据库
            Hotel hotel = this.getById(id);
            //判断结果是否为空
            if (hotel == null || hotel.getId() == null)
            {
                //数据库里查不到，无法插入或者更新
                log.warn("数据库里id为" + id + "的数据不存在，无法添加或者更新
ElasticSearch");
                return;
            }
            //数据库里存在
            //转文档类型
            HotelDoc hotelDoc = new HotelDoc(hotel);
            //构建请求
            IndexRequest indexRequest = new IndexRequest("hotel");
            //设置id
            indexRequest.id(hotel.getId().toString());
            //转json
            String json = JSON.toJSONString(hotelDoc);
            //构建请求体
            indexRequest.source(json, XContentType.JSON);
            //发起请求
            IndexResponse indexResponse =
restHighLevelClient.index(indexRequest, RequestOptions.DEFAULT);
            log.debug("添加或者更新ElasticSearch文档成功：" + id);
        }
        catch (Exception e)
        {
            //判断是否成功
            if (e.getMessage().contains("response=HTTP/1.1 200 OK}"))
            {
                log.debug("添加或者更新ElasticSearch文档成功：" + id);
            }
            else
            {
                log.warn("添加或者更新ElasticSearch文档失败：" + id);
                //抛出
                throw new RuntimeException(e);
            }
        }


    }

    @Override
    public void deleteElasticSearchHotelById(Long id)
    {
        try
        {
            log.debug("开始删除ElasticSearch文档：" + id);
            //构建请求
            DeleteRequest deleteRequest = new DeleteRequest("hotel");
            //设置id
            deleteRequest.id(id.toString());
            //发起请求
            DeleteResponse deleteResponse =
restHighLevelClient.delete(deleteRequest, RequestOptions.DEFAULT);
            log.debug("删除ElasticSearch文档成功：" + id);
        }
        catch (Exception e)
        {
            //判断是否成功
            if (e.getMessage().contains("response=HTTP/1.1 200 OK}"))
            {
                log.debug("删除ElasticSearch文档成功: " + id);
            }
            else
            {
                log.warn("删除ElasticSearch文档失败: " + id);
                //抛出
                throw new RuntimeException(e);
            }
        }

    }

    @Override
    public Map<String, List<String>> getFilters(RequestParams params)
    {
        try
        {
            //构建请求
            SearchRequest searchRequest = new SearchRequest("hotel");
            //查询
            buildBasicQuery(params, searchRequest);
            //分页
            searchRequest.source().size(0);
            //聚合
            buildAggregations(searchRequest);
            //发起请求
            SearchResponse searchResponse =
restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
            //获取数据
            //获取aggregations部分
            Aggregations aggregations = searchResponse.getAggregations();
            Map<String, List<String>> filters = new HashMap<>(3);
            //获取品牌
            List<String> brandList = getAggregationByName(aggregations,
"brandAgg");
            filters.put("brand", brandList);
            //获取城市
            List<String> cityList = getAggregationByName(aggregations,
"cityAgg");
            filters.put("city", cityList);
            //获取星级
            List<String> starList = getAggregationByName(aggregations,
"starAgg");
            filters.put("starName", starList);
            //返回
            return filters;
        }
        catch (Exception e)
        {
            throw new RuntimeException(e);
        }

    }

    /**
     * 聚合
     *
     * @param request SearchRequest
     */
    private void buildAggregations(SearchRequest request)
    {
        //分桶

 request.source().aggregation(AggregationBuilders.terms("brandAgg").field("brand
").size(100));

 request.source().aggregation(AggregationBuilders.terms("cityAgg").field("city")
.size(100));

 request.source().aggregation(AggregationBuilders.terms("starAgg").field("starNa
me").size(100));
    }

    /**
     * 根据聚合结果获取数据
     *
     * @param aggregations Aggregations
     * @param aggName      聚合的名字
     * @return List<String>
     */
    private List<String> getAggregationByName(Aggregations aggregations, String
aggName)
    {
        //根据聚合名称，获取聚合结果
        Terms terms = aggregations.get(aggName);
        //获取buckets
        List<? extends Terms.Bucket> buckets = terms.getBuckets();
        //遍历数据
        List<String> list = new ArrayList<>(buckets.size());
        for (Terms.Bucket bucket : buckets)
        {
            String brandName = bucket.getKeyAsString();
            list.add(brandName);
        }
        //返回
        return list;
    }
}
```

Listener:

```
package mao.elasticsearch_hotel.listener;

import mao.elasticsearch_hotel.constants.RabbitMQConstants;
import mao.elasticsearch_hotel.service.IHotelService;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
```

```java
/**
 * Project name(项目名称)：elasticsearch_hotel_realize_data_synchronization
 * Package(包名): mao.elasticsearch_hotel.listener
 * Class(类名): HotelListener
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/4
 * Time(创建时间)： 19:10
 * Version(版本): 1.0
 * Description(描述)： rabbitmq的监听器，消费者
 */

@Component
public class HotelListener
{
    @Resource
    private IHotelService hotelService;

    /**
     * rabbitmq的监听器，添加或者更新文档
     *
     * @param id id
     */
    @RabbitListener(queues = {RabbitMQConstants.INSERT_QUEUE_NAME})
    public void insertAndUpdateHotelListener(Long id)
    {
        hotelService.insertElasticSearchHotelById(id);
    }

    /**
     * 同理，删除文档
     *
     * @param id id
     */
    @RabbitListener(queues = {RabbitMQConstants.DELETE_QUEUE_NAME})
    public void deleteHotelListener(Long id)
    {
        hotelService.deleteElasticSearchHotelById(id);
    }
}
```

完整代码

https://github.com/maomao124/elasticsearch_hotel_final.git

后台：

https://github.com/maomao124/elasticsearch_hotel_management.git

---

end

---

by mao

2022 06 04
