# Elasticsearch



> Elasticsearch是一个开源的分布式、RESTful 风格的搜索和数据分析引擎。
>
> 本文中的例子都在araf-mas-demo工程中。

## 添加依赖

```
                <dependency>
                        <groupId>com.acca</groupId>
                        <artifactId>araf-elasticsearch-spring-boot-starter<
/artifactId>
                </dependency>
```

## 使用示例

### 添加configuration

```
/**
 * ElasticSearchConfig.
 *
 * @version OPRA v1.0
 * @author Li MingYi, 2020-03-02
 */
@Configuration
@EnableElasticsearchRepositories(basePackages = "com.acca.opra.mas.
repository")
public class ElasticSearchConfiguration {

}
```

### 修改配置文件

```
# spring
spring:
  data:
    elasticsearch:
      cluster-name: opra-es-dev
      cluster-nodes: 10.1.17.13:9300
  elasticsearch:
    rest:
      uris: ["http://10.1.17.13:9200"]
```

开发环境：10.1.17.13

测试环境：10.1.19.223

## 创建实体

```
/**
 * Phone
 *
 * @version OPRA v1.0
 * @author Li MingYi, 2020-03-02
 */
@Data
@AllArgsConstructor
@Document(indexName = "phone-#{T(araf.utils.DateUtils).dateToString(new
java.util.Date(), 'yyyy-MM-dd')}", type = "phone")
public class Phone implements Serializable {

    private static final long serialVersionUID = -3259164202484075333L;
    @Id
    private Long id;

    /**
     *
     */
    @Field(type = FieldType.Text, analyzer = "ik_max_word")
    private String title;
    /**
     *
     */
    @Field(type = FieldType.Keyword)
    private String category;

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

    /**
     *
     */
    @Field(type = FieldType.Double)
    private Double price;

    /**
     *
     */
    @Field(index = false, type = FieldType.Keyword)
    private String images;

}
```

## service与dao

```
/**
 * PhoneService
 *
 * @author Li Mingyi, 20200303
 * @version Opra v1.0.0
 */
@Service
public class PhoneService {

    @Autowired
    private ElasticsearchTemplate elasticsearchTemplate;

    @Autowired
    private PhoneRepository phoneRepository;

    /**
     *
     */
    public void createIndex() {
        // phone@Document
        elasticsearchTemplate.createIndex(Phone.class);
        // phoneidField
        elasticsearchTemplate.putMapping(Phone.class);
    }

    /**
     *
     */
    public void deleteIndex() {
        elasticsearchTemplate.deleteIndex("phone");
    }

    /**
     *
     */
    public void add() {
        Phone phone = new Phone(1L, "7", "", "", 2999.00,
                "https://img12.360buyimg.com/n1/s450x450_jfs/t1/14081/40
/4987/124705/5c371b20E53786645/c1f49cd69e6c7e6a.jpg");
        phoneRepository.save(phone);
    }

    /**
     *
     */
    public void addBatch() {
        List<Phone> list = new ArrayList<>();
        list.add(new Phone(2L, "R1", "", "", 3999.00,
                "https://img12.360buyimg.com/n1/s450x450_jfs/t1/14081/40
/4987/124705/5c371b20E53786645/c1f49cd69e6c7e6a.jpg"));
        list.add(new Phone(3L, "META20", "", "", 4999.00,
                "https://img12.360buyimg.com/n1/s450x450_jfs/t1/14081/40
/4987/124705/5c371b20E53786645/c1f49cd69e6c7e6a.jpg"));
```

```
        list.add(new Phone(4L, "iPhone X", "", "iPhone", 5100.00,
                "https://img12.360buyimg.com/n1/s450x450_jfs/t1/14081/40
/4987/124705/5c371b20E53786645/c1f49cd69e6c7e6a.jpg"));
        list.add(new Phone(5L, "iPhone XS", "", "iPhone", 5999.00,
                "https://img12.360buyimg.com/n1/s450x450_jfs/t1/14081/40
/4987/124705/5c371b20E53786645/c1f49cd69e6c7e6a.jpg"));
        //
        phoneRepository.saveAll(list);
    }

    /**
     *
     *
     * idPUT
     */

    /**
     *
     */
    public void deleteAll() {
        phoneRepository.deleteAll();
    }

    /**
     *
     */
    public void find() {
        //
        Iterable<Phone> phones = phoneRepository.findAll(Sort.by("price").
descending());
        phones.forEach(phone -> System.out.println("phone = " + phone));
    }

    /**
     *
     */

    public void queryByPriceBetween() {
        //
        List<Phone> list = phoneRepository.findByPriceBetween(5000.00,
6000.00);
        list.forEach(phone -> System.out.println("phone = " + phone));
    }

    /**
     *
     */
    public void search() {
        //
        NativeSearchQueryBuilder queryBuilder = new
NativeSearchQueryBuilder();
        //
        queryBuilder.withQuery(QueryBuilders.matchQuery("title", ""));
```

```
        //
        Page<Phone> phones = phoneRepository.search(queryBuilder.build());
        //
        long total = phones.getTotalElements();
        System.out.println("total = " + total);
        phones.forEach(phone -> System.out.println("phone = " + phone));
    }

    /**
     *
     */

    public void searchByPage() {
        //
        NativeSearchQueryBuilder queryBuilder = new
NativeSearchQueryBuilder();
        //
        queryBuilder.withQuery(QueryBuilders.termQuery("category", ""));
        //
        int page = 0;
        int size = 2;
        queryBuilder.withPageable(PageRequest.of(page, size));
        //
        Page<Phone> phones = phoneRepository.search(queryBuilder.build());
        long total = phones.getTotalElements();
        System.out.println(" = " + total);
        System.out.println(" = " + phones.getTotalPages());
        System.out.println("" + phones.getNumber());
        System.out.println("" + phones.getSize());
        phones.forEach(phone -> System.out.println("phone = " + phone));
    }

    /**
     *
     */

    public void searchAndSort() {
        //
        NativeSearchQueryBuilder queryBuilder = new
NativeSearchQueryBuilder();
        //
        queryBuilder.withQuery(QueryBuilders.termQuery("category", ""));
        //
        queryBuilder.withSort(SortBuilders.fieldSort("price").order
(SortOrder.ASC));
        //
        Page<Phone> phones = this.phoneRepository.search(queryBuilder.
build());
        //
        long total = phones.getTotalElements();
        System.out.println(" = " + total);
        phones.forEach(phone -> System.out.println("phone = " + phone));
    }
```

```
    /**
     *
     */

    public void testAgg() {
        NativeSearchQueryBuilder queryBuilder = new
NativeSearchQueryBuilder();
        //
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[] {
"" }, null));
        // 1termsbrandsbrand
        queryBuilder.addAggregation(AggregationBuilders.terms("brands").
field("brand"));
        // 2,AggregatedPage
        AggregatedPage<Phone> aggPage = (AggregatedPage<Phone>)
phoneRepository
                .search(queryBuilder.build());
        // 3
        // 3.1brands
        // StringtermStringTerm
        StringTerms agg = (StringTerms) aggPage.getAggregation("brands");
        // 3.2
        List<StringTerms.Bucket> buckets = agg.getBuckets();
        // 3.3
        for (StringTerms.Bucket bucket : buckets) {
            // 3.4key
            System.out.println(bucket.getKeyAsString());
            // 3.5
            System.out.println(bucket.getDocCount());
        }
    }

    /**
     *
     */

    public void testSubAgg() {
        NativeSearchQueryBuilder queryBuilder = new
NativeSearchQueryBuilder();
        //
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[] {
"" }, null));
        // 1termsbrandsbrand
        queryBuilder.addAggregation(AggregationBuilders.terms("brands").
field("brand")
                .subAggregation(AggregationBuilders.avg("priceAvg").field
("price")) //
        );
        // 2,AggregatedPage
        AggregatedPage<Phone> aggPage = (AggregatedPage<Phone>) this.
phoneRepository
                .search(queryBuilder.build());
```

```
        // 3
        // 3.1brands
        // StringtermStringTerm
        StringTerms agg = (StringTerms) aggPage.getAggregation("brands");
        // 3.2
        List<StringTerms.Bucket> buckets = agg.getBuckets();
        // 3.3
        for (StringTerms.Bucket bucket : buckets) {
            // 3.4key 3.5
            System.out.println(bucket.getKeyAsString() + "" + bucket.
getDocCount() + "");

            // 3.6.
            InternalAvg avg = (InternalAvg) bucket.getAggregations().
asMap().get("priceAvg");
            System.out.println("" + avg.getValue());
        }
    }
}
```

```
/**
 * PhoneRepository
 *
 * @version OPRA v1.0
 * @author Li MingYi, 2020-03-02
 */
public interface PhoneRepository extends ElasticsearchRepository<Phone,
Long> {

    /**
     *
     *
     * @param priceStart //
     * @param priceEnd //
     * @return List<Phone>
     */
    List<Phone> findByPriceBetween(double priceStart, double priceEnd);
}
```

## 单元测试

在需要使用elasticsearch场景的单元测试中，添加

```
@ClassRule
public static ElasticsearchTestContainer elasticsearch = new
ElasticsearchTestContainer("6.8.4");
```

单元测试的配置文件不需要修改。

# 使用规范

es中可以存储大量的数据，并进行索引查询。index类似数据库中的表的概念。**注意index应该需要分表，一般会按照时间范围进行分表，参见上文中的例子，查询时可以指定所有索引，也可以指定特定的索引。分表可以灵活的运维。**

es中字段类型，

keyword，关键字在检索的时候是不会被拆分的，比如国家名，地理名等

text 类型：当一个字段是要被全文搜索的，比如Email内容、产品描述，应该使用text类型。设置text类型以后，字段内容会被分析，在生成倒排索引以前，字符串会被分析器分成一个一个词项。text类型的字段不用于排序，很少用于聚合。

**整数类型** 类型 取值范围 byte -128~127 short -32768~32767 integer -231~231-1 short -263~263-1 在满足需求的情况下，尽可能选择范围小的数据类型。比如，某个字段的取值最大值不会超过100，那么选择byte类型即可。迄今为止吉尼斯记录的人类的年龄的最大值为134岁，对于年龄字段，short足矣。字段的长度越短，索引和搜索的效率越高。

**浮点类型** 类型 取值范围 doule 64位双精度IEEE 754浮点类型 float 32位单精度IEEE 754浮点类型 half_float 16位半精度IEEE 754浮点类型 scaled_float 缩放类型的的浮点数 对于float、half_float和scaled_float,-0.0和+0.0是不同的值，使用term查询查找-0.0不会匹配+0.0，同样range查询中上边界是-0.0不会匹配+0.0，下边界是+0.0不会匹配-0.0。 其中scaled_float，比如价格只需要精确到分，price为57.34的字段缩放因子为100，存起来就是5734 优先考虑使用带缩放因子的scaled_float浮点类型。

**date类型** 日期类型表示格式可以是以下几种： （1）日期格式的字符串，比如 “2018-01-13” 或 “2018-01-13 12:10:30” （2）long类型的毫秒数( milliseconds-since-the-epoch，epoch就是指UNIX诞生的UTC时间1970年1月1日0时0分0秒） （3）integer的秒数(seconds-since-the-epoch)

> 中文的查询，需要将中文拆成单个字符，再查询

# 数据量级上亿之后优化的点

1.数据缓存，fileSystem cache 尽可能只存储关键查询字段（结合其它手段，比如hbase组合查询）

2.数据预热，设计策略定时预热热点数据

3.冷热分离，热数据与冷数据放置于不同的索引

4.模型设计，尽可能少用复杂查询，在数据插入前处理好数据

5.分页性能，尽量避免深度分页，采用scroll api查询