

# 分布式搜索引擎03

## 0.学习目标

### 1.数据聚合

**聚合 (aggregations)** 可以让我们极其方便的实现对数据的统计、分析、运算。例如：

- 什么品牌的手机最受欢迎？
- 这些手机的平均价格、最高价格、最低价格？
- 这些手机每月的销售情况如何？

实现这些统计功能的比数据库的sql要方便的多，而且查询速度非常快，可以实现近实时搜索效果。

#### 1.1.聚合的种类

聚合常见的有三类：

- **桶 (Bucket)** 聚合：用来对文档做分组
  - TermAggregation：按照文档字段值分组，例如按照品牌值分组、按照国家分组
  - Date Histogram：按照日期阶梯分组，例如一周为一组，或者一月为一组
- **度量 (Metric)** 聚合：用以计算一些值，比如：最大值、最小值、平均值等
  - Avg：求平均值
  - Max：求最大值
  - Min：求最小值
  - Stats：同时求max、min、avg、sum等
- **管道 (pipeline)** 聚合：其它聚合的结果为基础做聚合

**注意：**参加聚合的字段必须是keyword、日期、数值、布尔类型

#### 1.2.DSL实现聚合

现在，我们要统计所有数据中的酒店品牌有几种，其实就是按照品牌对数据分组。此时可以根据酒店品牌的名称做聚合，也就是Bucket聚合。

##### 1.2.1.Bucket聚合语法

语法如下：

```

1 GET /hotel/_search
2 {
3   "size": 0, // 设置size为0, 结果中不包含文档, 只包含聚合结果
4   "aggs": { // 定义聚合
5     "brandAgg": { //给聚合起个名字
6       "terms": { // 聚合的类型, 按照品牌值聚合, 所以选择term
7         "field": "brand", // 参与聚合的字段
8         "size": 20 // 希望获取的聚合结果数量
9       }
10    }
11  }
12 }

```

结果如图:

```

{
  "took" : 16,
  "timed_out" : false,
  "_shards" : {↔},
  "hits" : {↔},
  "aggregations" : {
    "brandAgg" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 41,
      "buckets" : [ 聚合的bucket数组
        {
          "key" : "7天酒店", 品牌为"7天酒店"的桶
          "doc_count" : 34
        },
        {
          "key" : "如家", 品牌为"如家"的桶
          "doc_count" : 30
        },
        {
          "key" : "速8", 品牌为"速8"的桶
          "doc_count" : 20
        },
        {
          "key" : "自在假日"

```

### 1.2.2. 聚合结果排序

默认情况下, Bucket聚合会统计Bucket内的文档数量, 记为 *count*, 并且按照 *count* 降序排序。

我们可以指定 *order* 属性, 自定义聚合的排序方式:

```

1 GET /hotel/_search
2 {
3   "size": 0,
4   "aggs": {
5     "brandAgg": {
6       "terms": {
7         "field": "brand",
8         "order": {
9           "_count": "asc" // 按照_count升序排列
10        },
11       "size": 20
12     }
13   }
14 }
15 }

```

### 1.2.3. 限定聚合范围

默认情况下，Bucket聚合是对索引库的所有文档做聚合，但真实场景下，用户会输入搜索条件，因此聚合必须是对搜索结果聚合。那么聚合必须添加限定条件。

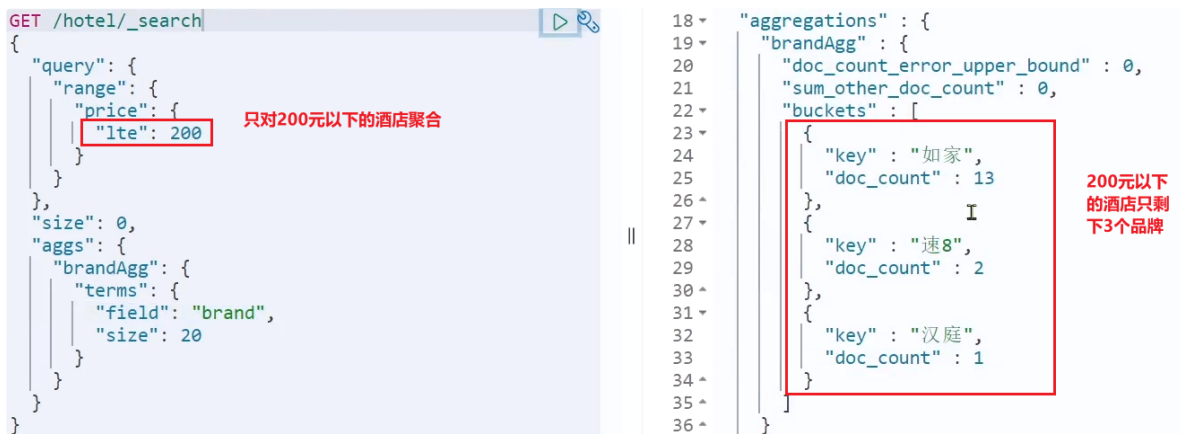
我们可以限定要聚合的文档范围，只要添加query条件即可：

```

1 GET /hotel/_search
2 {
3   "query": {
4     "range": {
5       "price": {
6         "lte": 200 // 只对200元以下的文档聚合
7       }
8     }
9   },
10  "size": 0,
11  "aggs": {
12    "brandAgg": {
13      "terms": {
14        "field": "brand",
15        "size": 20
16      }
17    }
18  }
19 }

```

这次，聚合得到的品牌明显变少了：



#### 1.2.4.Metric聚合语法

上节课，我们对酒店按照品牌分组，形成了一个桶。现在我们需要对桶内的酒店做运算，获取每个品牌的用户评分的min、max、avg等值。

这就要用到Metric聚合了，例如stat聚合：就可以获取min、max、avg等结果。

语法如下：



这次的score\_stats聚合是在brandAgg的聚合内部嵌套的子聚合。因为我们需要在每个桶分别计算。

另外，我们还可以给聚合结果做个排序，例如按照每个桶的酒店平均分做排序：

```

    }
  }
}

# 嵌套聚合metric
GET /hotel/_search
{
  "size": 0,
  "aggs": {
    "brandAgg": {
      "terms": {
        "field": "brand",
        "size": 20,
        "order": {
          "scoreAgg.avg": "desc"
        }
      },
      "aggs": {
        "scoreAgg": {
          "stats": {
            "field": "score"
          }
        }
      }
    }
  }
}

```

```

52 |         "avg" : 40.25,
53 |         "sum" : 370.0
54 |       }
55 |     },
56 |     {
57 |       "key" : "和颐",
58 |       "doc_count" : 12,
59 |       "scoreAgg" : {
60 |         "count" : 12,
61 |         "min" : 44.0,
62 |         "max" : 47.0,
63 |         "avg" : 46.083333333333336,
64 |         "sum" : 553.0
65 |       }
66 |     },
67 |     {
68 |       "key" : "喜来登",
69 |       "doc_count" : 14,
70 |       "scoreAgg" : {
71 |         "count" : 14,
72 |         "min" : 44.0,
73 |         "max" : 48.0,
74 |         "avg" : 46.07142857142857,
75 |         "sum" : 645.0

```

### 1.2.5.小结

aggs代表聚合，与query同级，此时query的作用是？

- 限定聚合的文档范围

聚合必须的三要素：

- 聚合名称
- 聚合类型
- 聚合字段

聚合可配置属性有：

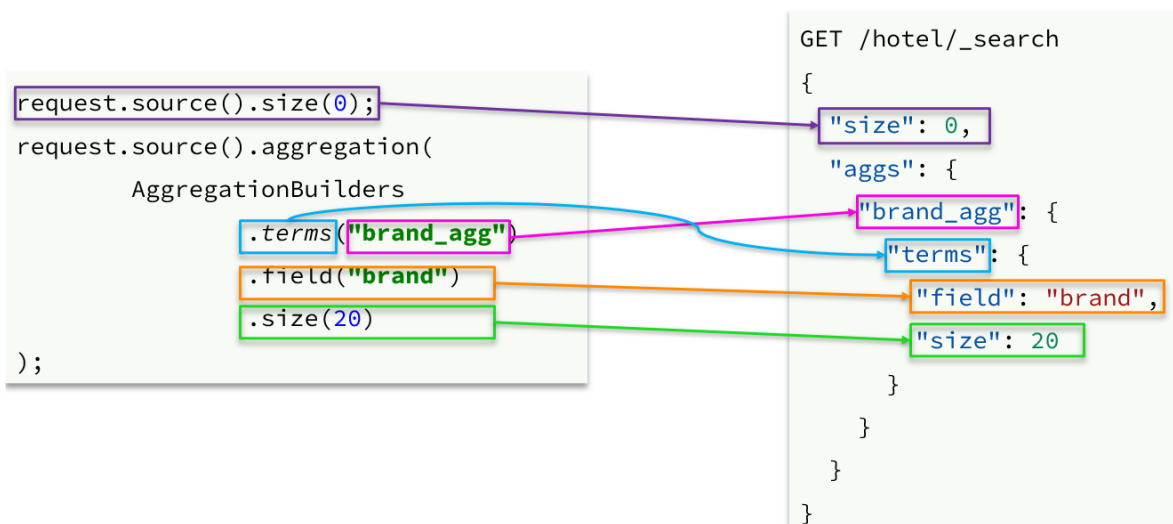
- size：指定聚合结果数量
- order：指定聚合结果排序方式
- field：指定聚合字段

## 1.3.RestAPI实现聚合

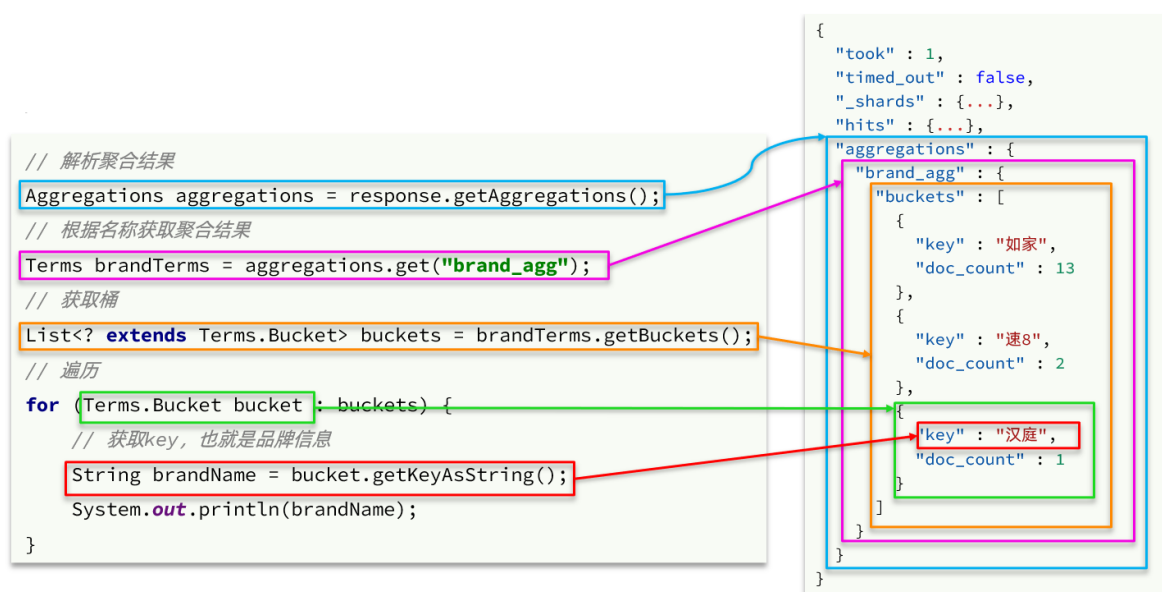
### 1.3.1.API语法

聚合条件与query条件同级别，因此需要使用request.source()来指定聚合条件。

聚合条件的语法：



聚合的结果也与查询结果不同，API也比较特殊。不过同样是JSON逐层解析：



### 1.3.2. 业务需求

需求：搜索页面的品牌、城市等信息不应该是在页面写死，而是通过聚合索引库中的酒店数据得来的：

黑马旅游  
www.itheima.com

全部结果:

城市	上海	北京	深圳	杭州					
星级	四星	五星	二钻	三钻	四钻	五钻			
品牌	7天酒店	如家	速8	皇冠假日	华美达	万怡	喜来登	万豪	和颐
价格	100元以下	100-300元	300-600元	600-1500元	1500元以上				

分析：

目前，页面的城市列表、星级列表、品牌列表都是写死的，并不会随着搜索结果的变化而变化。但是用户搜索条件改变时，搜索结果会跟着变化。

例如：用户搜索“东方明珠”，那搜索的酒店肯定是在上海东方明珠附近，因此，城市只能是上海，此时城市列表中就不应该显示北京、深圳、杭州这些信息了。

也就是说，搜索结果中包含哪些城市，页面就应该列出哪些城市；搜索结果中包含哪些品牌，页面就应该列出哪些品牌。

如何得知搜索结果中包含哪些品牌？如何得知搜索结果中包含哪些城市？

使用聚合功能，利用Bucket聚合，对搜索结果中的文档基于品牌分组、基于城市分组，就能得知包含哪些品牌、哪些城市了。

因为是对搜索结果聚合，因此聚合是**限定范围的聚合**，也就是说聚合的限定条件跟搜索文档的条件一致。

查看浏览器可以发现，前端其实已经发出了这样的一个请求：



请求**参数与搜索文档的参数完全一致**。

返回值类型就是页面要展示的最终结果：

key	value
城市 String	上海 北京 深圳 杭州 List<String>
星级	四星 五星 二钻 三钻 四钻 五钻
品牌	7天酒店 如家 速8 皇冠假日 华美达 万怡 喜来登 万豪 和
价格	100元以下 100-300元 300-600元 600-1500元 1500元以上

结果是一个Map结构：

- key是字符串，城市、星级、品牌、价格
- value是集合，例如多个城市的名称

### 1.3.3. 业务实现

在 `cn.itcast.hotel.web` 包的 `HotelController` 中添加一个方法，遵循下面的要求：

- 请求方式： `POST`
- 请求路径： `/hotel/filters`
- 请求参数： `RequestParam`，与搜索文档的参数一致
- 返回值类型： `Map<String, List<String>>`

代码：

```
1  @PostMapping("filters")
2  public Map<String, List<String>> getFilters(@RequestBody RequestParams
    params){
3      return hotelService.getFilters(params);
4  }
```

这里调用了 `IHotelService` 中的 `getFilters` 方法，尚未实现。

在 `cn.itcast.hotel.service.IHotelService` 中定义新方法：

```
1  Map<String, List<String>> filters(RequestParams params);
```

在 `cn.itcast.hotel.service.impl.HotelService` 中实现该方法：

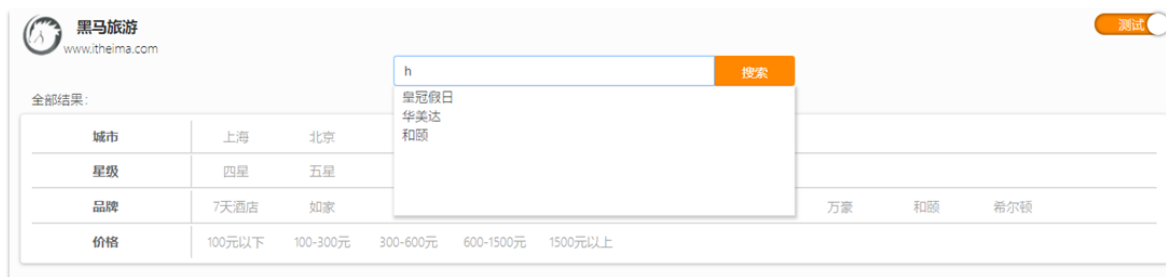
```
1  @Override
2  public Map<String, List<String>> filters(RequestParams params) {
3      try {
4          // 1.准备Request
5          SearchRequest request = new SearchRequest("hotel");
6          // 2.准备DSL
7          // 2.1.query
8          buildBasicQuery(params, request);
9          // 2.2.设置size
10         request.source().size(0);
11         // 2.3.聚合
12         buildAggregation(request);
13         // 3.发出请求
14         SearchResponse response = client.search(request,
            RequestOptions.DEFAULT);
15         // 4.解析结果
16         Map<String, List<String>> result = new HashMap<>();
17         Aggregations aggregations = response.getAggregations();
```



```
18 // 4.1.根据品牌名称, 获取品牌结果
19 List<String> brandList = getAggByName(aggregations, "brandAgg");
20 result.put("品牌", brandList);
21 // 4.2.根据品牌名称, 获取品牌结果
22 List<String> cityList = getAggByName(aggregations, "cityAgg");
23 result.put("城市", cityList);
24 // 4.3.根据品牌名称, 获取品牌结果
25 List<String> starList = getAggByName(aggregations, "starAgg");
26 result.put("星级", starList);
27
28 return result;
29 } catch (IOException e) {
30     throw new RuntimeException(e);
31 }
32 }
33
34 private void buildAggregation(SearchRequest request) {
35     request.source().aggregation(AggregationBuilders
36         .terms("brandAgg")
37         .field("brand")
38         .size(100)
39     );
40     request.source().aggregation(AggregationBuilders
41         .terms("cityAgg")
42         .field("city")
43         .size(100)
44     );
45     request.source().aggregation(AggregationBuilders
46         .terms("starAgg")
47         .field("starName")
48         .size(100)
49     );
50 }
51
52 private List<String> getAggByName(Aggregations aggregations, String
aggName) {
53     // 4.1.根据聚合名称获取聚合结果
54     Terms brandTerms = aggregations.get(aggName);
55     // 4.2.获取buckets
56     List<? extends Terms.Bucket> buckets = brandTerms.getBuckets();
57     // 4.3.遍历
58     List<String> brandList = new ArrayList<>();
59     for (Terms.Bucket bucket : buckets) {
60         // 4.4.获取key
61         String key = bucket.getKeyAsString();
62         brandList.add(key);
63     }
64     return brandList;
65 }
```

## 2. 自动补全

当用户在搜索框输入字符时，我们应该提示出与该字符有关的搜索项，如图：

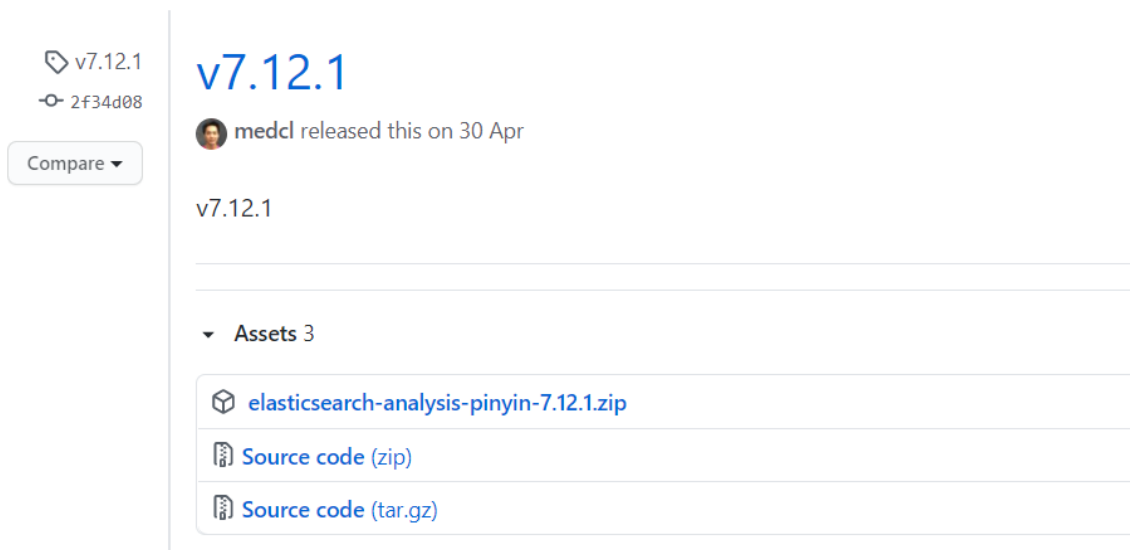


这种根据用户输入的字母，提示完整词条的功能，就是自动补全了。

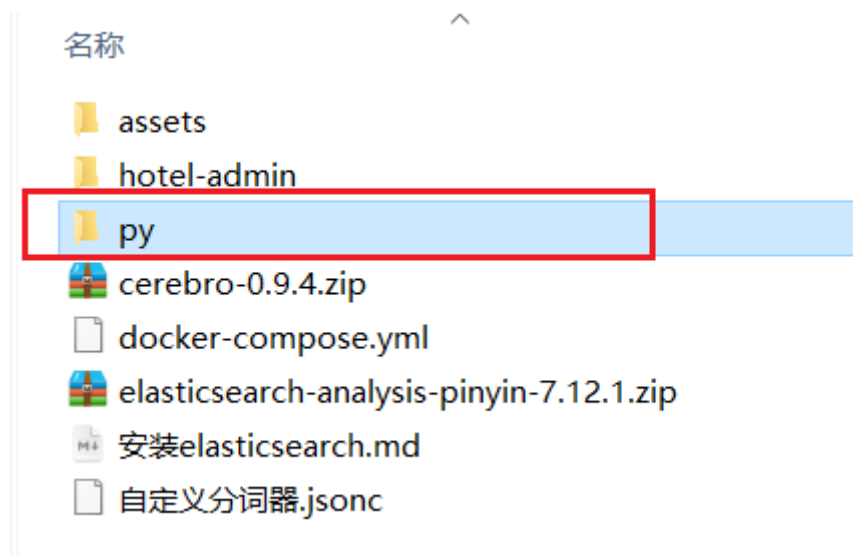
因为需要根据拼音字母来推断，因此要用到拼音分词功能。

### 2.1. 拼音分词器

要实现根据字母做补全，就必须对文档按照拼音分词。在GitHub上恰好有elasticsearch的拼音分词插件。地址：<https://github.com/medcl/elasticsearch-analysis-pinyin>



课前资料中也提供了拼音分词器的安装包：



安装方式与IK分词器一样，分三步：

- ①解压
- ②上传到虚拟机中，elasticsearch的plugin目录
- ③重启elasticsearch
- ④测试

详细安装步骤可以参考IK分词器的安装过程。

测试用法如下：

```
1 POST /_analyze
2 {
3   "text": "如家酒店还不错",
4   "analyzer": "pinyin"
5 }
```

结果：

```
{
  "tokens" : [
    {
      "token" : "ru",
      "start_offset" : 0,
      "end_offset" : 0,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "fjjdhbc",
      "start_offset" : 0,
      "end_offset" : 0,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "jia",
      "start_offset" : 0,
      "end_offset" : 0,
      "type" : "word",
      "position" : 1
    }
  ],
}
```

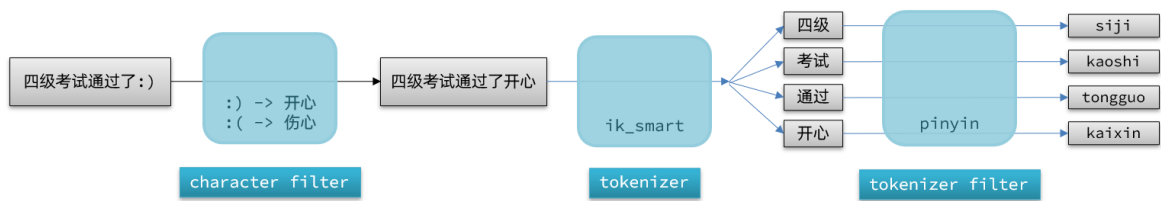
## 2.2. 自定义分词器

默认的拼音分词器会将每个汉字单独分为拼音，而我们希望的是每个词条形成一组拼音，需要对拼音分词器做个性化定制，形成自定义分词器。

elasticsearch中分词器 (analyzer) 的组成包含三部分：

- character filters: 在tokenizer之前对文本进行处理。例如删除字符、替换字符
- tokenizer: 将文本按照一定的规则切割成词条 (term)。例如keyword，就是不分词；还有ik\_smart
- tokenizer filter: 将tokenizer输出的词条做进一步处理。例如大小写转换、同义词处理、拼音处理等

文档分词时会依次由这三部分来处理文档：

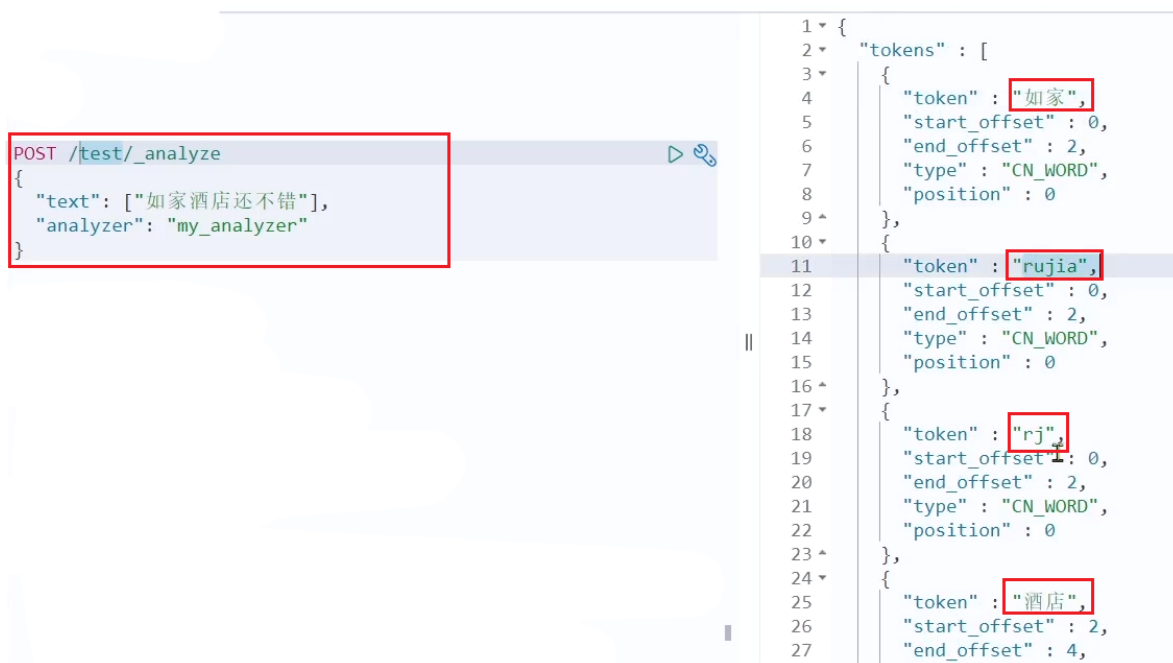


声明自定义分词器的语法如下:

```

1  PUT /test
2  {
3    "settings": {
4      "analysis": {
5        "analyzer": { // 自定义分词器
6          "my_analyzer": { // 分词器名称
7            "tokenizer": "ik_max_word",
8            "filter": "py"
9          }
10       },
11       "filter": { // 自定义tokenizer filter
12         "py": { // 过滤器名称
13           "type": "pinyin", // 过滤器类型，这里是pinyin
14           "keep_full_pinyin": false,
15           "keep_joined_full_pinyin": true,
16           "keep_original": true,
17           "limit_first_letter_length": 16,
18           "remove_duplicated_term": true,
19           "none_chinese_pinyin_tokenize": false
20         }
21       }
22     },
23     "mappings": {
24       "properties": {
25         "name": {
26           "type": "text",
27           "analyzer": "my_analyzer",
28           "search_analyzer": "ik_smart"
29         }
30       }
31     }
32   }
33 }
```

测试:



总结:

如何使用拼音分词器?

- ①下载pinyin分词器
- ②解压并放到elasticsearch的plugin目录
- ③重启即可

如何自定义分词器?

- ①创建索引库时, 在settings中配置, 可以包含三部分
- ②character filter
- ③tokenizer
- ④filter

拼音分词器注意事项?

- 为了避免搜索到同音字, 搜索时不要使用拼音分词器

## 2.3. 自动补全查询

elasticsearch提供了 [Completion Suggester](#) 查询来实现自动补全功能。这个查询会匹配以用户输入内容开头的词条并返回。为了提高补全查询的效率, 对于文档中字段的类型有一些约束:

- 参与补全查询的字段必须是completion类型。
- 字段的内容一般是用来补全的多个词条形成的数组。

比如, 一个这样的索引库:

```
1 // 创建索引库
2 PUT test
3 {
4   "mappings": {
5     "properties": {
6       "title": {
7         "type": "completion"
8       }
9     }
10  }
11 }
```

然后插入下面的数据：

```
1 // 示例数据
2 POST test/_doc
3 {
4   "title": ["Sony", "WH-1000XM3"]
5 }
6 POST test/_doc
7 {
8   "title": ["SK-II", "PITERA"]
9 }
10 POST test/_doc
11 {
12   "title": ["Nintendo", "switch"]
13 }
```

查询的DSL语句如下：

```
1 // 自动补全查询
2 GET /test/_search
3 {
4   "suggest": {
5     "title_suggest": {
6       "text": "s", // 关键字
7       "completion": {
8         "field": "title", // 补全查询的字段
9         "skip_duplicates": true, // 跳过重复的
10        "size": 10 // 获取前10条结果
11      }
12    }
13  }
14 }
```

## 2.4. 实现酒店搜索框自动补全

现在，我们的hotel索引库还没有设置拼音分词器，需要修改索引库中的配置。但是我们知道索引库是无法修改的，只能删除然后重新创建。

另外，我们需要添加一个字段，用来做自动补全，将brand、suggestion、city等都放进去，作为自动补全的提示。

因此，总结一下，我们需要做的事情包括：

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

### 2.4.1. 修改酒店映射结构

代码如下：

```
1 // 酒店数据索引库
2 PUT /hotel
3 {
4   "settings": {
5     "analysis": {
6       "analyzer": {
7         "text_analyzer": {
8           "tokenizer": "ik_max_word",
9           "filter": "py"
10        },
11        "completion_analyzer": {
12          "tokenizer": "keyword",
13          "filter": "py"
14        }
15      },
16      "filter": {
17        "py": {
18          "type": "pinyin",
19          "keep_full_pinyin": false,
20          "keep_joined_full_pinyin": true,
21          "keep_original": true,
22          "limit_first_letter_length": 16,
23          "remove_duplicated_term": true,
24          "none_chinese_pinyin_tokenize": false
```



```
25     }
26   }
27 }
28 },
29 "mappings": {
30   "properties": {
31     "id":{
32       "type": "keyword"
33     },
34     "name":{
35       "type": "text",
36       "analyzer": "text_anlyzer",
37       "search_analyzer": "ik_smart",
38       "copy_to": "all"
39     },
40     "address":{
41       "type": "keyword",
42       "index": false
43     },
44     "price":{
45       "type": "integer"
46     },
47     "score":{
48       "type": "integer"
49     },
50     "brand":{
51       "type": "keyword",
52       "copy_to": "all"
53     },
54     "city":{
55       "type": "keyword"
56     },
57     "starName":{
58       "type": "keyword"
59     },
60     "business":{
61       "type": "keyword",
62       "copy_to": "all"
63     },
64     "location":{
65       "type": "geo_point"
66     },
67     "pic":{
68       "type": "keyword",
69       "index": false
70     },
71     "all":{
72       "type": "text",
73       "analyzer": "text_anlyzer",
74       "search_analyzer": "ik_smart"
75     },
76     "suggestion":{
```

```

77         "type": "completion",
78         "analyzer": "completion_analyzer"
79     }
80 }
81 }
82 }

```

#### 2.4.2. 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。

代码如下：

```

1  package cn.itcast.hotel.pojo;
2
3  import lombok.Data;
4  import lombok.NoArgsConstructor;
5
6  import java.util.ArrayList;
7  import java.util.Arrays;
8  import java.util.Collections;
9  import java.util.List;
10
11  @Data
12  @NoArgsConstructor
13  public class HotelDoc {
14      private Long id;
15      private String name;
16      private String address;
17      private Integer price;
18      private Integer score;
19      private String brand;
20      private String city;
21      private String starName;
22      private String business;
23      private String location;
24      private String pic;
25      private Object distance;
26      private Boolean isAD;
27      private List<String> suggestion;
28
29      public HotelDoc(Hotel hotel) {
30          this.id = hotel.getId();
31          this.name = hotel.getName();
32          this.address = hotel.getAddress();
33          this.price = hotel.getPrice();

```

```

34     this.score = hotel.getScore();
35     this.brand = hotel.getBrand();
36     this.city = hotel.getCity();
37     this.starName = hotel.getStarName();
38     this.business = hotel.getBusiness();
39     this.location = hotel.getLatitude() + ", " +
hotel.getLongitude();
40     this.pic = hotel.getPic();
41     // 组装suggestion
42     if(this.business.contains("/")){
43         // business有多个值，需要切割
44         String[] arr = this.business.split("/");
45         // 添加元素
46         this.suggestion = new ArrayList<>();
47         this.suggestion.add(this.brand);
48         Collections.addAll(this.suggestion, arr);
49     }else {
50         this.suggestion = Arrays.asList(this.brand, this.business);
51     }
52 }
53 }

```

### 2.4.3. 重新导入

重新执行之前编写的导入数据功能，可以看到新的酒店数据中包含了suggestion:

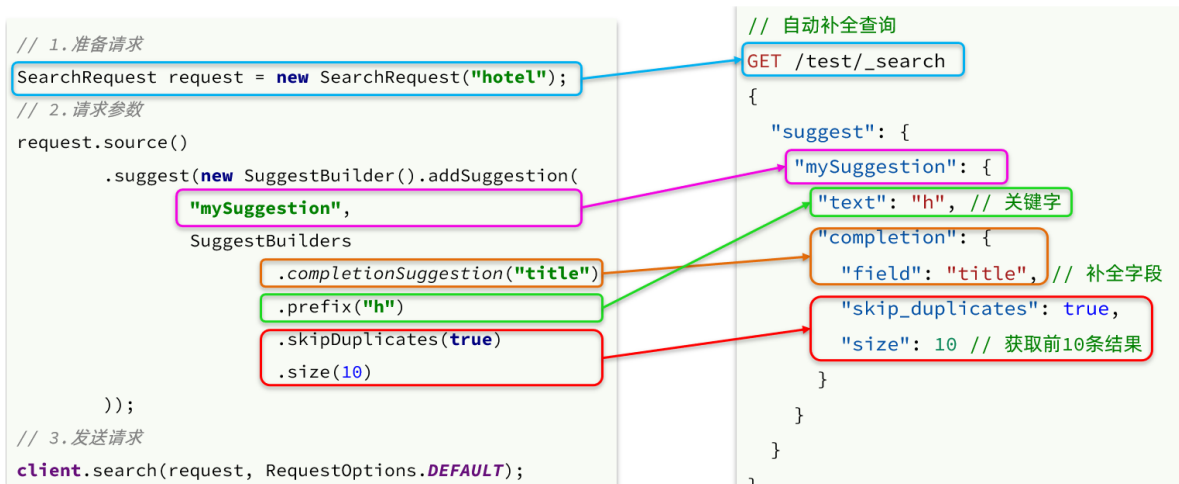
```

    "business" : "江湾/五角场商业区",
    "city" : "上海",
    "id" : 39141,
    "location" : "31.290057, 121.508804",
    "name" : "7天连锁酒店(上海五角场复旦同济大学店)",
    "pic" : "https://m.tuniucdn.com/fb2/t1/G2/M00/C7/E3/Cii-T1knFX
        -uFNAEAAKYkQPcw1IAAUIL012_w200_h200_c1_t0.jpg",
    "price" : 349,
    "score" : 38,
    "starName" : "二钻",
    "suggestion" : [
        "7天酒店",
        "江湾",
        "五角场商业区"
    ]
},
,
,

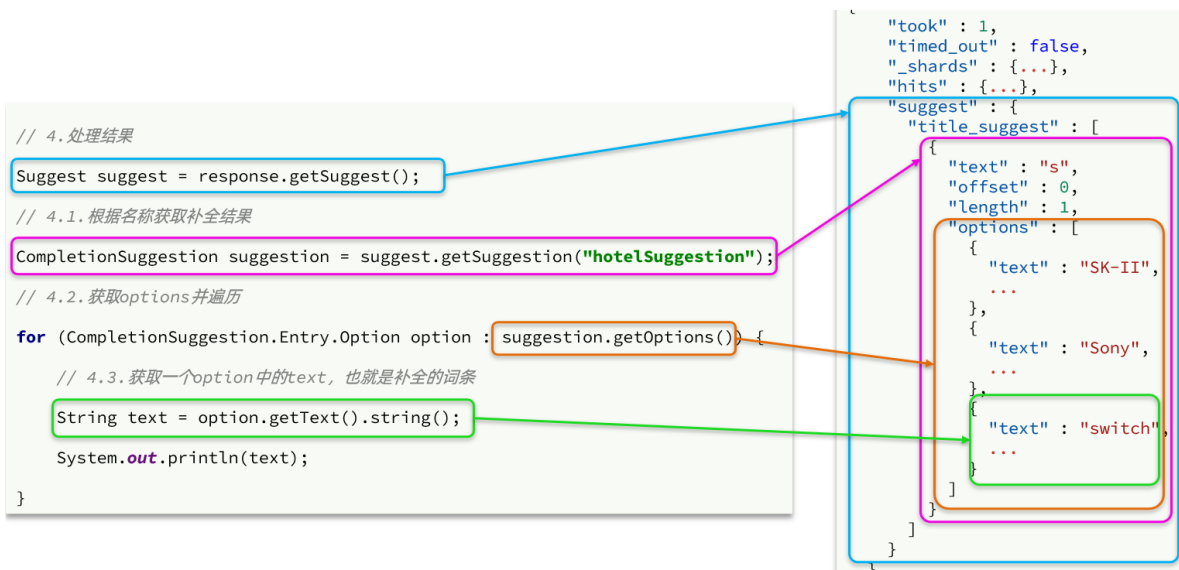
```

### 2.4.4. 自动补全查询的JavaAPI

之前我们学习了自动补全查询的DSL，而没有学习对应的JavaAPI，这里给出一个示例：



而自动补全的结果也比较特殊，解析的代码如下：



## 2.4.5. 实现搜索框自动补全

查看前端页面，可以发现当我们在输入框键入时，前端会发起ajax请求：

▼ General

Request URL: http://localhost:8089/hotel/suggestion?key=h

Request Method: GET

Status Code: 404

Remote Address: [::1]:8089

Referrer Policy: strict-origin-when-cross-origin

► Response Headers (8)

► Request Headers (13)

▼ Query String Parameters

key: h

view source

view URL-encoded

返回值是补全词条的集合，类型为 `List<String>`

- 1) 在 `cn.itcast.hotel.web` 包下的 `HotelController` 中添加新接口，接收新的请求：

```
1 @GetMapping("suggestion")
2 public List<String> getSuggestions(@RequestParam("key") String prefix) {
3     return hotelService.getSuggestions(prefix);
4 }
```

- 2) 在 `cn.itcast.hotel.service` 包下的 `IhotelService` 中添加方法：

```
1 List<String> getSuggestions(String prefix);
```

- 3) 在 `cn.itcast.hotel.service.impl.HotelService` 中实现该方法：

```
1 @Override
2 public List<String> getSuggestions(String prefix) {
3     try {
4         // 1.准备Request
5         SearchRequest request = new SearchRequest("hotel");
6         // 2.准备DSL
7         request.source().suggest(new SuggestBuilder().addSuggestion(
8             "suggestions",
9             SuggestBuilders.completionSuggestion("suggestion")
10                .prefix(prefix)
11                .skipDuplicates(true)
12                .size(10)
13        ));
14        // 3.发起请求
15        SearchResponse response = client.search(request,
16            RequestOptions.DEFAULT);
17        // 4.解析结果
18        Suggest suggest = response.getSuggest();
19        // 4.1.根据补全查询名称，获取补全结果
20        CompletionSuggestion suggestions =
21            suggest.getSuggestion("suggestions");
22        // 4.2.获取options
23        List<CompletionSuggestion.Entry.Option> options =
24            suggestions.getOptions();
25        // 4.3.遍历
26        List<String> list = new ArrayList<>(options.size());
27        for (CompletionSuggestion.Entry.Option option : options) {
28            String text = option.getText().toString();
29            list.add(text);
30        }
31    } catch (Exception e) {
32        e.printStackTrace();
33    }
34    return list;
35 }
```

```
27     }
28     return list;
29 } catch (IOException e) {
30     throw new RuntimeException(e);
31 }
32 }
```

## 3. 数据同步

elasticsearch中的酒店数据来自于mysql数据库，因此mysql数据发生改变时，elasticsearch也必须跟着改变，这个就是elasticsearch与mysql之间的**数据同步**。



在微服务中，负责酒店管理（操作mysql）的业务与负责酒店搜索（操作elasticsearch）的业务可能在两个**不同**的微服务上，数据同步该如何实现呢？

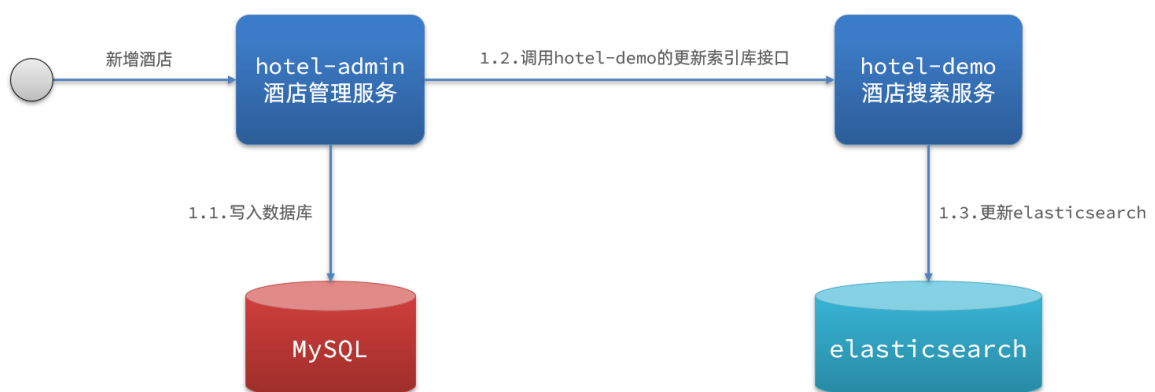
### 3.1. 思路分析

常见的数据同步方案有三种：

- 同步调用
- 异步通知
- 监听binlog

#### 3.1.1. 同步调用

方案一：同步调用

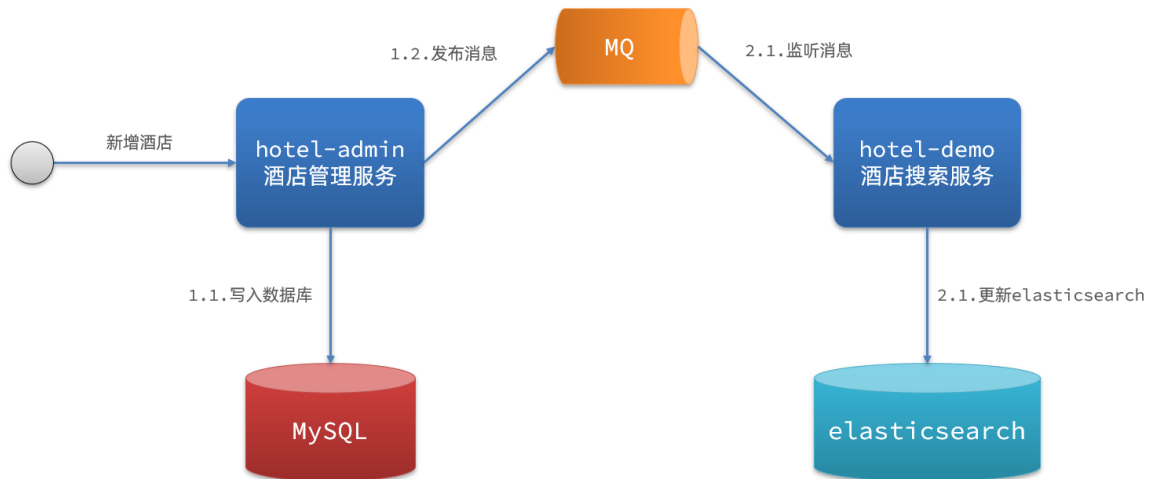


基本步骤如下：

- hotel-demo对外提供接口，用来修改elasticsearch中的数据
- 酒店管理服务在完成数据库操作后，直接调用hotel-demo提供的接口，

### 3.1.2. 异步通知

方案二：异步通知

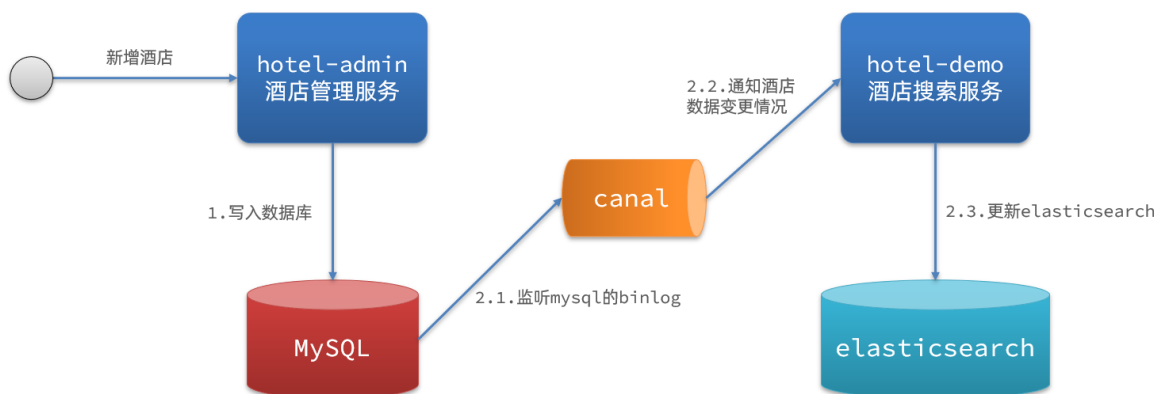


流程如下：

- hotel-admin对mysql数据库数据完成增、删、改后，发送MQ消息
- hotel-demo监听MQ，接收到消息后完成elasticsearch数据修改

### 3.1.3. 监听binlog

方案三：监听binlog



流程如下：

- 给mysql开启binlog功能
- mysql完成增、删、改操作都会记录在binlog中
- hotel-demo基于canal监听binlog变化，实时更新elasticsearch中的内容

### 3.1.4. 选择

---

方式一：同步调用

- 优点：实现简单，粗暴
- 缺点：业务耦合度高

方式二：异步通知

- 优点：低耦合，实现难度一般
- 缺点：依赖mq的可靠性

方式三：监听binlog

- 优点：完全解除服务间耦合
- 缺点：开启binlog增加数据库负担、实现复杂度高

## 3.2. 实现数据同步

---

### 3.2.1. 思路

---

利用课前资料提供的hotel-admin项目作为酒店管理的微服务。当酒店数据发生增、删、改时，要求对elasticsearch中数据也要完成相同操作。

步骤：

- 导入课前资料提供的hotel-admin项目，启动并测试酒店数据的CRUD
- 声明exchange、queue、RoutingKey
- 在hotel-admin中的增、删、改业务中完成消息发送
- 在hotel-demo中完成消息监听，并更新elasticsearch中数据
- 启动并测试数据同步功能

### 3.2.2. 导入demo

---

导入课前资料提供的hotel-admin项目：



assets	文件夹
hotel-admin	文件夹
py	文件夹
cerebro-0.9.4.zip	好压 ZIP 压缩文件
docker-compose.yml	YML 文件
elasticsearch-analysis-pinyin-7.12.1.zip	好压 ZIP 压缩文件
安装elasticsearch.md	Markdown File
自定义分词器.jsonc	JSONC 文件

运行后，访问 <http://localhost:8099>

## 酒店数据管理

新增酒店

ID	酒店名称	酒店品牌	酒店价格	所在商圈	操作
36934	7天连锁酒店(上海宝山路地铁站店)	7天酒店	338	四川北路商业区	<a href="#">编辑</a> <a href="#">删除</a>
38609	速8酒店(上海赤峰路店)	速8	247	四川北路商业区	<a href="#">编辑</a> <a href="#">删除</a>
38665	速8酒店上海中山北路兰田路店	速8	216	长风公园	<a href="#">编辑</a> <a href="#">删除</a>
38812	7天连锁酒店(上海漕溪路地铁站店)	7天酒店	298	八万人体育场	<a href="#">编辑</a> <a href="#">删除</a>
39106	7天连锁酒店 (上海莘庄地铁站店)	7天酒店	348	莘庄工业区	<a href="#">编辑</a> <a href="#">删除</a>

< 1 2 3 4 5 6 ... 46 >

其中包含了酒店的CRUD功能：

```

hotel-admin) × C HotelController.java × application.yaml × C MqConstants.java ×

@PostMapping
public void saveHotel(@RequestBody Hotel hotel){
    hotelService.save(hotel);
}

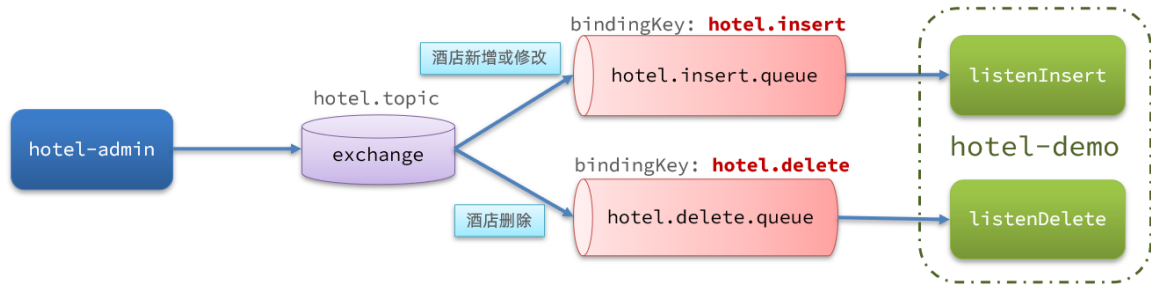
@PutMapping()
public void updateById(@RequestBody Hotel hotel){
    if (hotel.getId() == null) {
        throw new InvalidParameterException("id不能为空");
    }
    hotelService.updateById(hotel);
}

@DeleteMapping("/{id}")
public void deleteById(@PathVariable("id") Long id) {
    hotelService.removeById(id);
}

```

### 3.2.3. 声明交换机、队列

MQ结构如图:



#### 1) 引入依赖

在hotel-admin、hotel-demo中引入rabbitmq的依赖:

```
1 <!--amqp-->
2 <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-amqp</artifactId>
5 </dependency>
```

#### 2) 声明队列交换机名称

在hotel-admin和hotel-demo中的 `cn.itcast.hotel.constatnts` 包下新建一个类 `MqConstants` :

```
1 package cn.itcast.hotel.constatnts;
2
3 public class MqConstants {
4     /**
5      * 交换机
6      */
7     public final static String HOTEL_EXCHANGE = "hotel.topic";
8     /**
9      * 监听新增和修改的队列
10    */
11    public final static String HOTEL_INSERT_QUEUE = "hotel.insert.queue";
12    /**
13     * 监听删除的队列
14     */
15    public final static String HOTEL_DELETE_QUEUE = "hotel.delete.queue";
16    /**
17     * 新增或修改的RoutingKey
18     */
19 }
```

```

19     public final static String HOTEL_INSERT_KEY = "hotel.insert";
20     /**
21      * 删除的RoutingKey
22      */
23     public final static String HOTEL_DELETE_KEY = "hotel.delete";
24 }

```

### 3) 声明队列交换机

在hotel-demo中，定义配置类，声明队列、交换机：

```

1  package cn.itcast.hotel.config;
2
3  import cn.itcast.hotel.constants.MqConstants;
4  import org.springframework.amqp.core.Binding;
5  import org.springframework.amqp.core.BindingBuilder;
6  import org.springframework.amqp.core.Queue;
7  import org.springframework.amqp.core.TopicExchange;
8  import org.springframework.context.annotation.Bean;
9  import org.springframework.context.annotation.Configuration;
10
11  @Configuration
12  public class MqConfig {
13      @Bean
14      public TopicExchange topicExchange(){
15          return new TopicExchange(MqConstants.HOTEL_EXCHANGE, true,
16          false);
17      }
18
19      @Bean
20      public Queue insertQueue(){
21          return new Queue(MqConstants.HOTEL_INSERT_QUEUE, true);
22      }
23
24      @Bean
25      public Queue deleteQueue(){
26          return new Queue(MqConstants.HOTEL_DELETE_QUEUE, true);
27      }
28
29      @Bean
30      public Binding insertQueueBinding(){
31          return
32          BindingBuilder.bind(insertQueue()).to(topicExchange()).with(MqConstants.H
33          OTEL_INSERT_KEY);
34      }
35
36      @Bean
37      public Binding deleteQueueBinding(){
38          return
39          BindingBuilder.bind(deleteQueue()).to(topicExchange()).with(MqConstants.H
40          OTEL_DELETE_KEY);
41      }
42  }

```

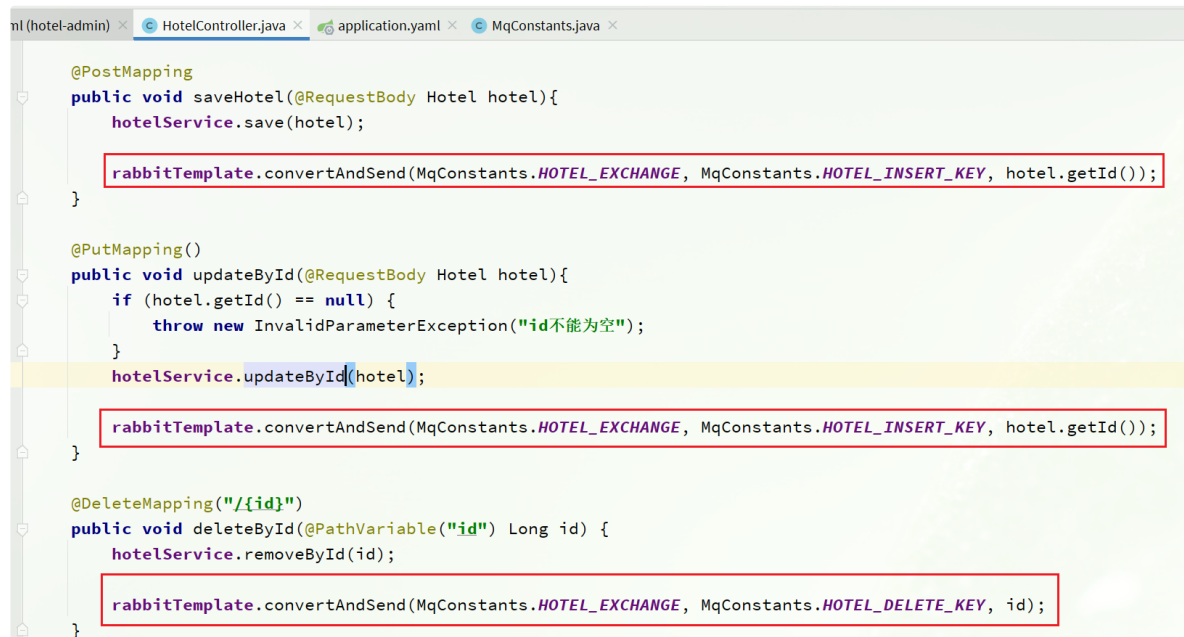
```

35         return
        BindingBuilder.bind(deleteQueue()).to(topicExchange()).with(MqConstants.H
        OTEL_DELETE_KEY);
36     }
37 }

```

### 3.2.4. 发送MQ消息

在hotel-admin中的增、删、改业务中分别发送MQ消息：



```

@PostMapping
public void saveHotel(@RequestBody Hotel hotel){
    hotelService.save(hotel);

    rabbitTemplate.convertAndSend(MqConstants.HOTEL_EXCHANGE, MqConstants.HOTEL_INSERT_KEY, hotel.getId());
}

@PutMapping()
public void updateById(@RequestBody Hotel hotel){
    if (hotel.getId() == null) {
        throw new InvalidParameterException("id不能为空");
    }
    hotelService.updateById(hotel);

    rabbitTemplate.convertAndSend(MqConstants.HOTEL_EXCHANGE, MqConstants.HOTEL_INSERT_KEY, hotel.getId());
}

@DeleteMapping("/{id}")
public void deleteById(@PathVariable("id") Long id) {
    hotelService.removeById(id);

    rabbitTemplate.convertAndSend(MqConstants.HOTEL_EXCHANGE, MqConstants.HOTEL_DELETE_KEY, id);
}

```

### 3.2.5. 接收MQ消息

hotel-demo接收到MQ消息要做的事情包括：

- 新增消息：根据传递的hotel的id查询hotel信息，然后新增一条数据到索引库
- 删除消息：根据传递的hotel的id删除索引库中的一条数据

1) 首先在hotel-demo的 `cn.itcast.hotel.service` 包下的 `IHotelService` 中新增新增、删除业务

```

1 void deleteById(Long id);
2
3 void insertById(Long id);

```

2) 给hotel-demo中的 `cn.itcast.hotel.service.impl` 包下的HotelService中实现业务：

```

1 @Override
2 public void deleteById(Long id) {

```

```

3      try {
4          // 1.准备Request
5          DeleteRequest request = new DeleteRequest("hotel",
id.toString());
6          // 2.发送请求
7          client.delete(request, RequestOptions.DEFAULT);
8      } catch (IOException e) {
9          throw new RuntimeException(e);
10     }
11 }
12
13 @Override
14 public void insertById(Long id) {
15     try {
16         // 0.根据id查询酒店数据
17         Hotel hotel = getById(id);
18         // 转换为文档类型
19         HotelDoc hotelDoc = new HotelDoc(hotel);
20
21         // 1.准备Request对象
22         IndexRequest request = new
IndexRequest("hotel").id(hotel.getId().toString());
23         // 2.准备Json文档
24         request.source(JSON.toJSONString(hotelDoc), XContentType.JSON);
25         // 3.发送请求
26         client.index(request, RequestOptions.DEFAULT);
27     } catch (IOException e) {
28         throw new RuntimeException(e);
29     }
30 }

```

### 3) 编写监听器

在hotel-demo中的 `cn.itcast.hotel.mq` 包新增一个类:

```

1 package cn.itcast.hotel.mq;
2
3 import cn.itcast.hotel.constants.MqConstants;
4 import cn.itcast.hotel.service.IHotelService;
5 import org.springframework.amqp.rabbit.annotation.RabbitListener;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Component;
8
9 @Component
10 public class HotelListener {
11
12     @Autowired
13     private IHotelService hotelService;
14

```

```

15     /**
16      * 监听酒店新增或修改的业务
17      * @param id 酒店id
18      */
19     @RabbitListener(queues = MqConstants.HOTEL_INSERT_QUEUE)
20     public void listenHotelInsertOrUpdate(Long id){
21         hotelService.insertById(id);
22     }
23
24     /**
25      * 监听酒店删除的业务
26      * @param id 酒店id
27      */
28     @RabbitListener(queues = MqConstants.HOTEL_DELETE_QUEUE)
29     public void listenHotelDelete(Long id){
30         hotelService.deleteById(id);
31     }
32 }

```

## 4. 集群

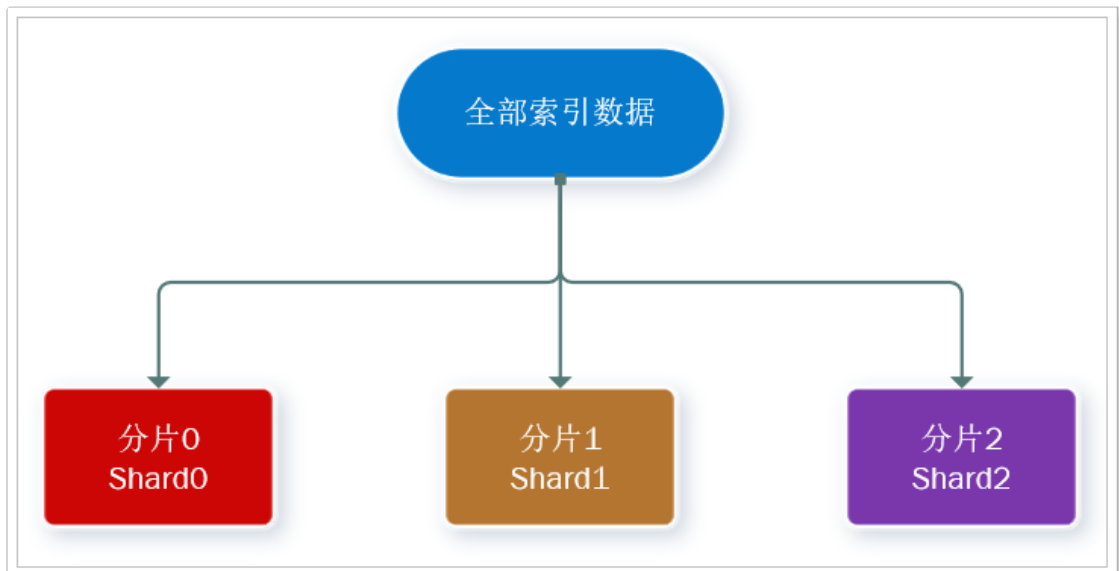
单机的elasticsearch做数据存储，必然面临两个问题：海量数据存储问题、单点故障问题。

- 海量数据存储问题：将索引库从逻辑上拆分为N个分片（shard），存储到多个节点
- 单点故障问题：将分片数据在不同节点备份（replica）

ES集群相关概念：

- 集群（cluster）：一组拥有共同的 cluster name 的节点。
- 节点（node）：集群中的一个 Elasticsearch 实例
- 分片（shard）：索引可以被拆分为不同的部分进行存储，称为分片。在集群环境下，一个索引的不同分片可以拆分到不同的节点中

解决问题：数据量太大，单点存储量有限的问题。



此处，我们把数据分成3片：shard0、shard1、shard2

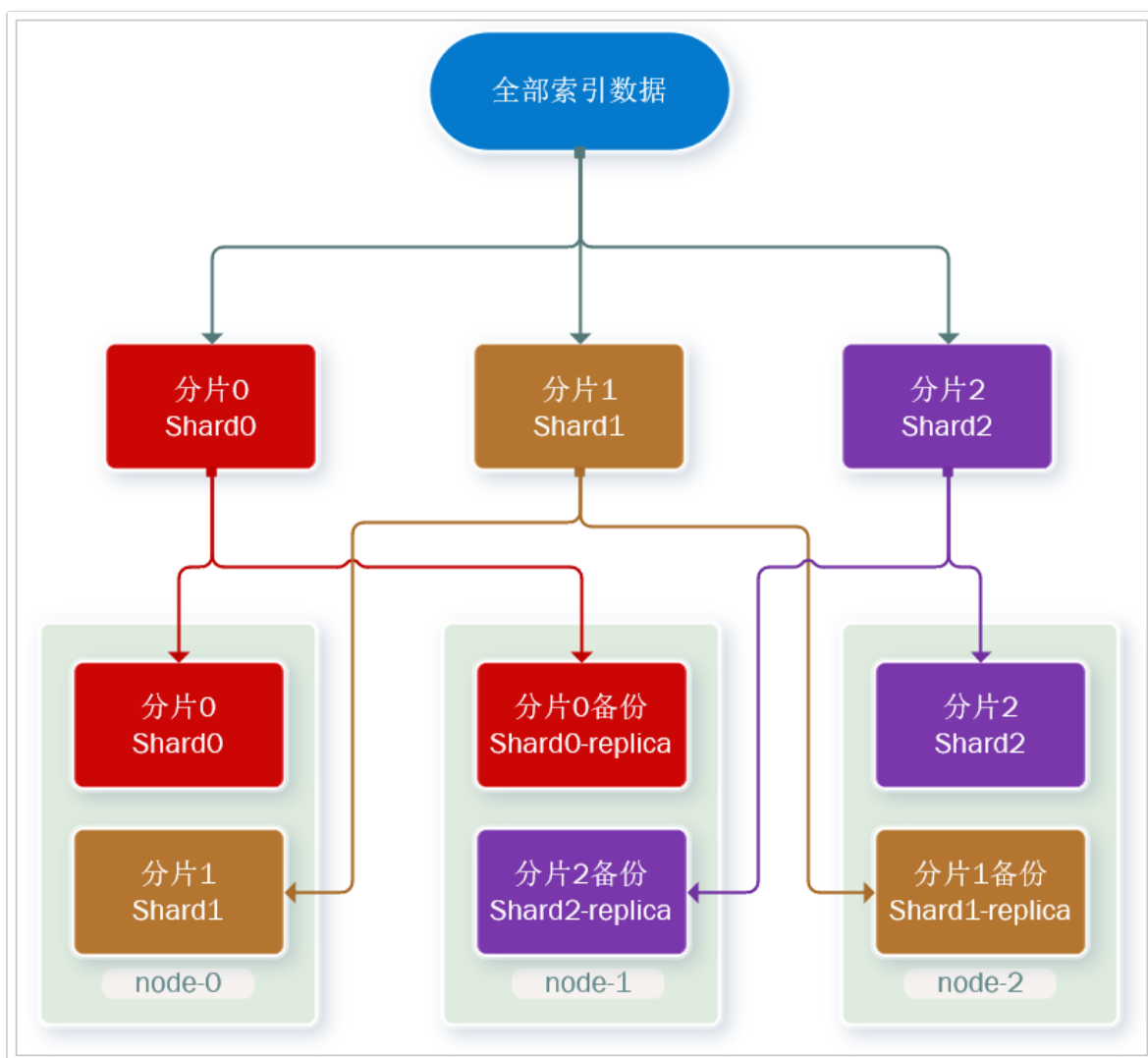
- 主分片 (Primary shard)：相对于副本分片的定义。
- 副本分片 (Replica shard) 每个主分片可以有一个或者多个副本，数据和主分片一样。

数据备份可以保证高可用，但是每个分片备份一份，所需要的节点数量就会翻一倍，成本实在是太高了！

为了在高可用和成本间寻求平衡，我们可以这样做：

- 首先对数据分片，存储到不同节点
- 然后对每个分片进行备份，放到对方节点，完成互相备份

这样可以大大减少所需要的服务节点数量，如图，我们以3分片，每个分片备份一份为例：



现在，每个分片都有1个备份，存储在3个节点：

- node0: 保存了分片0和1
- node1: 保存了分片0和2
- node2: 保存了分片1和2

参考课前资料的文档：

名称	类型
assets	文件夹
hotel-admin	文件夹
py	文件夹
cerebro-0.9.4.zip	好压 ZIP 压缩文件
docker-compose.yml	YML 文件
elasticsearch-analysis-pinyin-7.12.1.zip	好压 ZIP 压缩文件
安装elasticsearch.md	Markdown File
自定义分词器.jsonc	JSONC 文件

其中的第四章节：

## 安装elasticsearch

- > 1.部署单点es
- > 2.部署kibana
- > 3.安装IK分词器

### 4.部署es集群

- 4.1.创建es集群
- 4.2.集群状态监控
- > 4.3.创建索引库
- 4.4.查看分片效果

## 4.2. 集群脑裂问题

### 4.2.1. 集群职责划分

elasticsearch中集群节点有不同的职责划分：



节点类型	配置参数	默认值	节点职责
master eligible	node.master	true	备选主节点：主节点可以管理和记录集群状态、决定分片在哪个节点、处理创建和删除索引库的请求
data	node.data	true	数据节点：存储数据、搜索、聚合、CRUD
ingest	node.ingest	true	数据存储之前的预处理
coordinating	上面3个参数都为false 则为coordinating节点	无	路由请求到其它节点 合并其它节点处理的结果，返回给用户

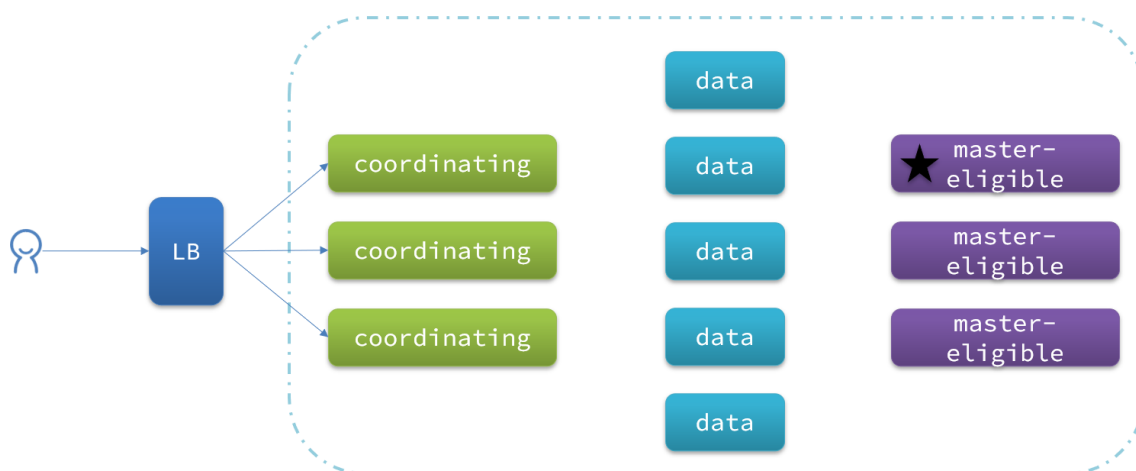
默认情况下，集群中的任何一个节点都同时具备上述四种角色。

但是真实的集群一定要将集群职责分离：

- master节点：对CPU要求高，但是内存要求第
- data节点：对CPU和内存要求都高
- coordinating节点：对网络带宽、CPU要求高

职责分离可以让我们根据不同节点的需求分配不同的硬件去部署。而且避免业务之间的互相干扰。

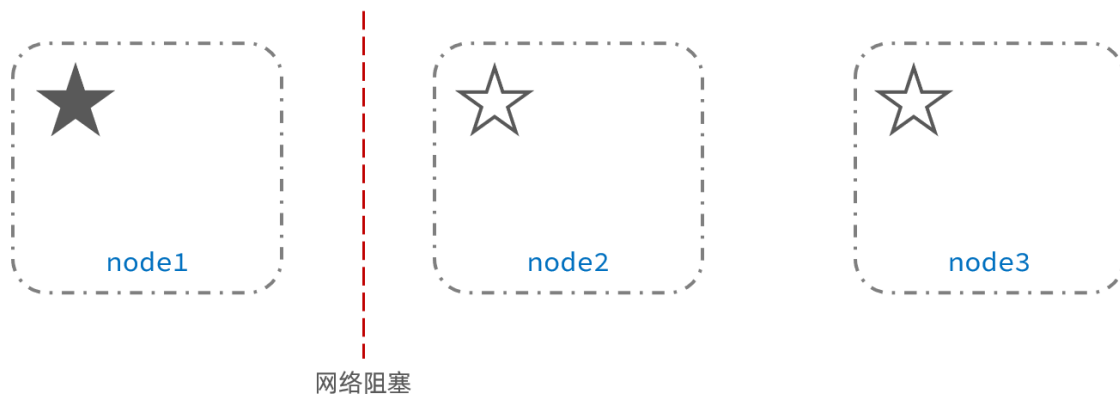
一个典型的es集群职责划分如图：



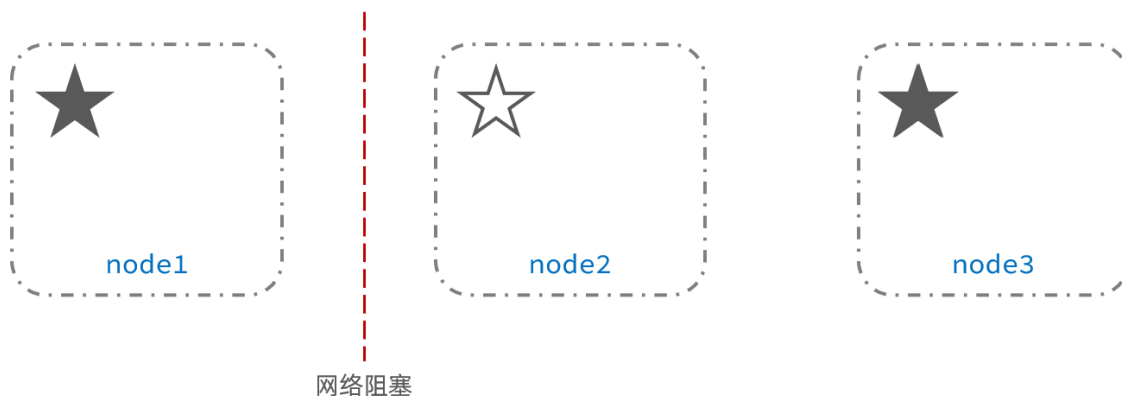
#### 4.2.2. 脑裂问题

脑裂是因为集群中的节点失联导致的。

例如一个集群中，主节点与其它节点失联：



此时，node2和node3认为node1宕机，就会重新选主：



当node3当选后，集群继续对外提供服务，node2和node3自成集群，node1自成集群，两个集群数据不同步，出现数据差异。

当网络恢复后，因为集群中有两个master节点，集群状态的不一致，出现脑裂的情况：



解决脑裂的方案是，要求选票超过  $(\text{eligible节点数量} + 1) / 2$  才能当选为主，因此eligible节点数量最好是奇数。对应配置项是discovery.zen.minimum\_master\_nodes，在es7.0以后，已经成为默认配置，因此一般不会发生脑裂问题

例如：3个节点形成的集群，选票必须超过  $(3 + 1) / 2$ ，也就是2票。node3得到node2和node3的选票，当选为主。node1只有自己1票，没有当选。集群中依然只有1个主节点，没有出现脑裂。

#### 4.2.3.小结

master eligible节点的作用是什么？

- 参与集群选主
- 主节点可以管理集群状态、管理分片信息、处理创建和删除索引库的请求

data节点的作用是什么？

- 数据的CRUD

coordinator节点的作用是什么？

- 路由请求到其它节点
- 合并查询到的结果，返回给用户

## 4.3. 集群分布式存储

当新增文档时，应该保存到不同分片，保证数据均衡，那么coordinating node如何确定数据该存储到哪个分片呢？

### 4.3.1. 分片存储测试

插入三条数据：

The image shows two screenshots of a REST client interface, likely Postman, used to test inserting documents into a MongoDB collection. Both screenshots show a POST request to the endpoint `http://192.168.150.101:9200/itcast/_doc/1` and `http://192.168.150.101:9200/itcast/_doc/3` respectively. The request body is set to JSON format. The first screenshot shows the JSON body `{ "title": "我试着插入一条, id = 1" }`. The second screenshot shows the JSON body `{ "title": "我试着插入一条, id = 3" }`. The interface includes tabs for Params, Authorization, Headers (9), Body, Pre-request Script, Tests, and Settings. The Body tab is selected, and the format is set to JSON.

```
1 {  
2   "title": "我试着插入一条, id = 1"  
3 }
```

```
1 {  
2   "title": "我试着插入一条, id = 3"  
3 }
```

POST

▼

http://192.168.150.101:9200/itcast/\_doc/5

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▼

1

{

2

... "title": "我试着插入一条, id = 5"

3

}

测试可以看到，三条数据分别在不同分片：

GET

▼

http://192.168.150.101:9200/itcast/\_search

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▼

1

{

2

... "explain": true,

3

... "query": {

4

... "match\_all": {}

5

... }

6

}

结果：

```

{
  "_shard": "[itcast][1]",
  "_node": "HhY3NGsLRii5CcImypsXJQ",
  "_index": "itcast",
  "_type": "_doc",
  "_id": "3",
  "_score": 1.0,
  "_source": {
    "title": "试着插入一条 id = 3"
  },
  "_explanation": {
    "value": 1.0,
    "description": "*:~*",
    "details": []
  }
},
{
  "_shard": "[itcast][2]",
  "_node": "APLhlP8qSoKs5gh9pmnC_A",
  "_index": "itcast",
  "_type": "_doc",
  "_id": "1",
  "_score": 1.0,
  "_source": {
    "title": "试着插入一条 id = 1"
  },
}

```

#### 4.3.2. 分片存储原理

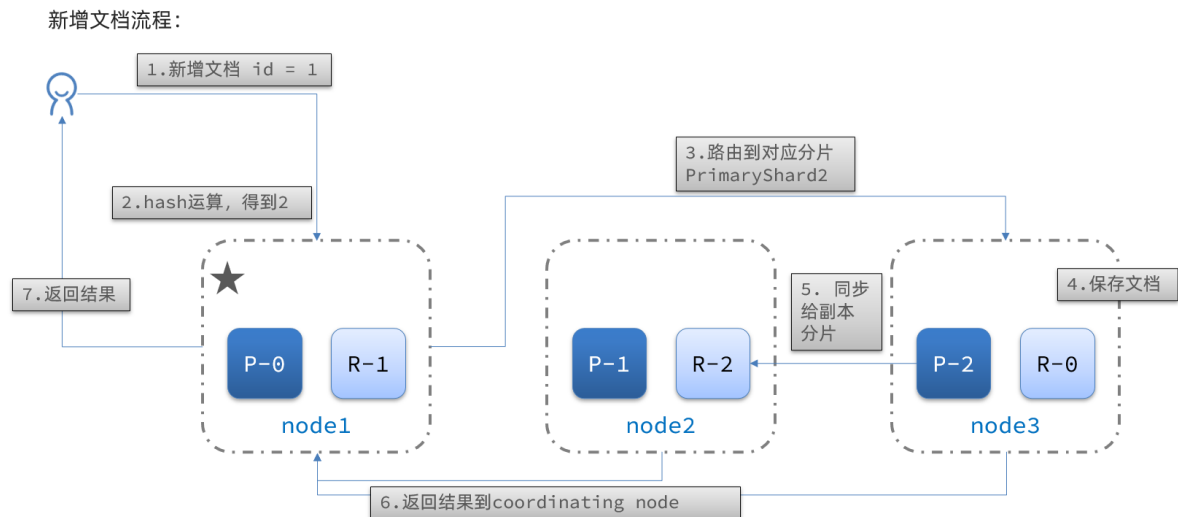
elasticsearch会通过hash算法来计算文档应该存储到哪个分片：

```
shard = hash(_routing) % number_of_shards
```

说明：

- `_routing`默认是文档的id
- 算法与分片数量有关，因此索引库一旦创建，分片数量不能修改！

新增文档的流程如下：



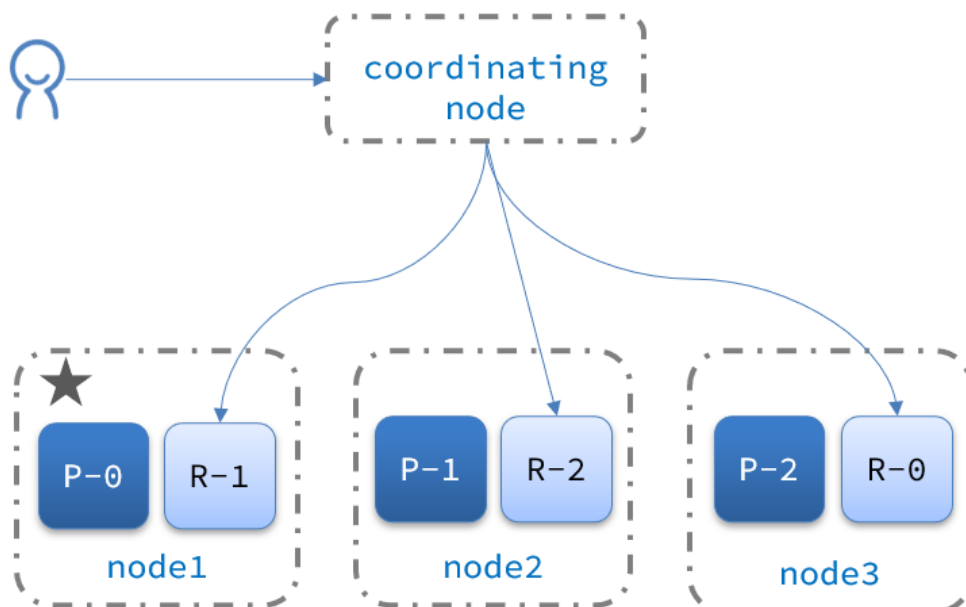
解读：

- 1) 新增一个id=1的文档
- 2) 对id做hash运算，假如得到的是2，则应该存储到shard-2
- 3) shard-2的主分片在node3节点，将数据路由到node3
- 4) 保存文档
- 5) 同步给shard-2的副本replica-2，在node2节点
- 6) 返回结果给coordinating-node节点

## 4.4. 集群分布式查询

elasticsearch的查询分成两个阶段：

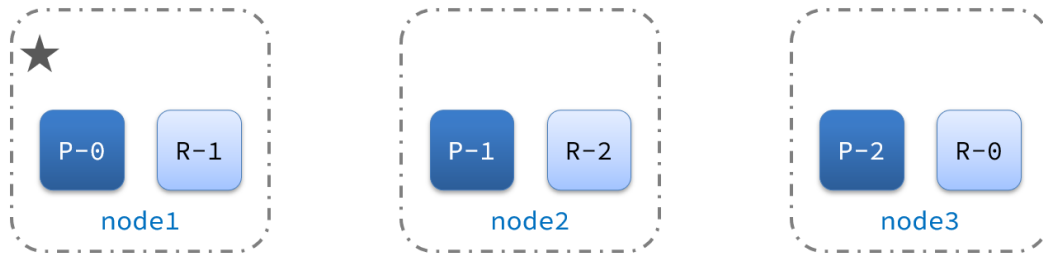
- scatter phase: 分散阶段，coordinating node会把请求分发到每一个分片
- gather phase: 聚集阶段，coordinating node汇总data node的搜索结果，并处理为最终结果集返回给用户



## 4.5. 集群故障转移

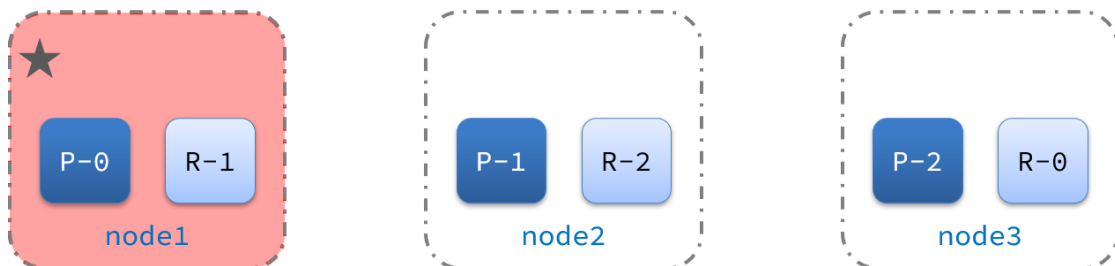
集群的master节点会监控集群中的节点状态，如果发现有节点宕机，会立即将宕机节点的分片数据迁移到其它节点，确保数据安全，这个叫做故障转移。

1) 例如一个集群结构如图：

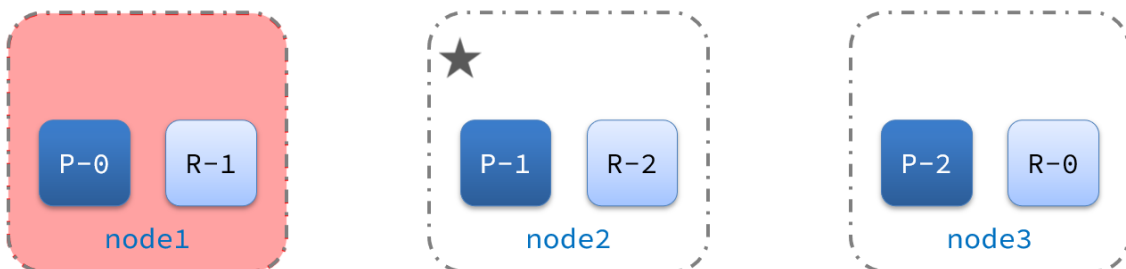


现在，node1是主节点，其它两个节点是从节点。

2) 突然，node1发生了故障：



宕机后的第一件事，需要重新选主，例如选中了node2：



node2成为主节点后，会检测集群监控状态，发现：shard-1、shard-0没有副本节点。因此需要将node1上的数据迁移到node2、node3：

