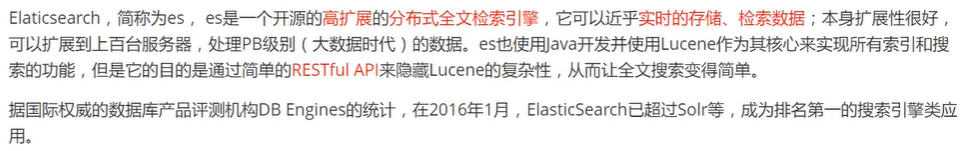
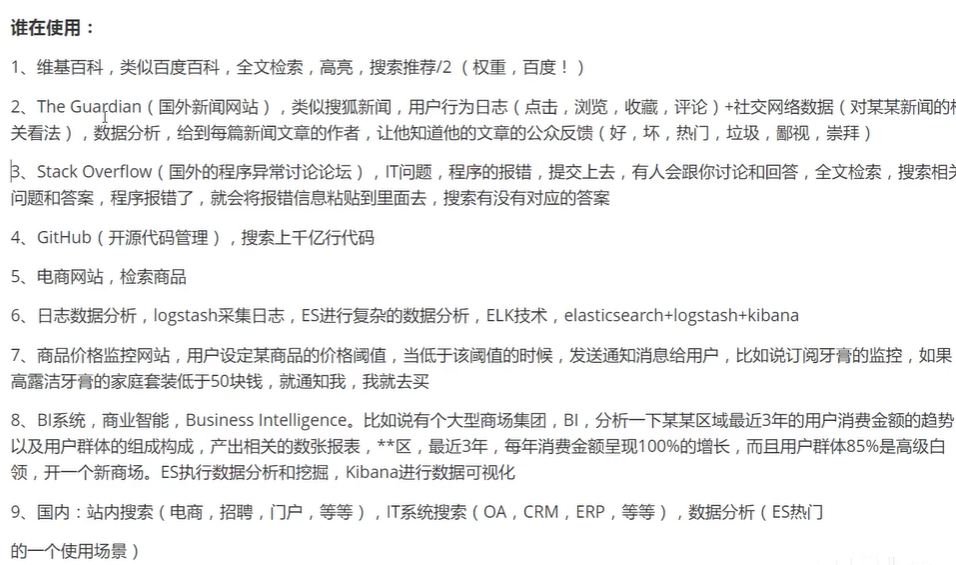
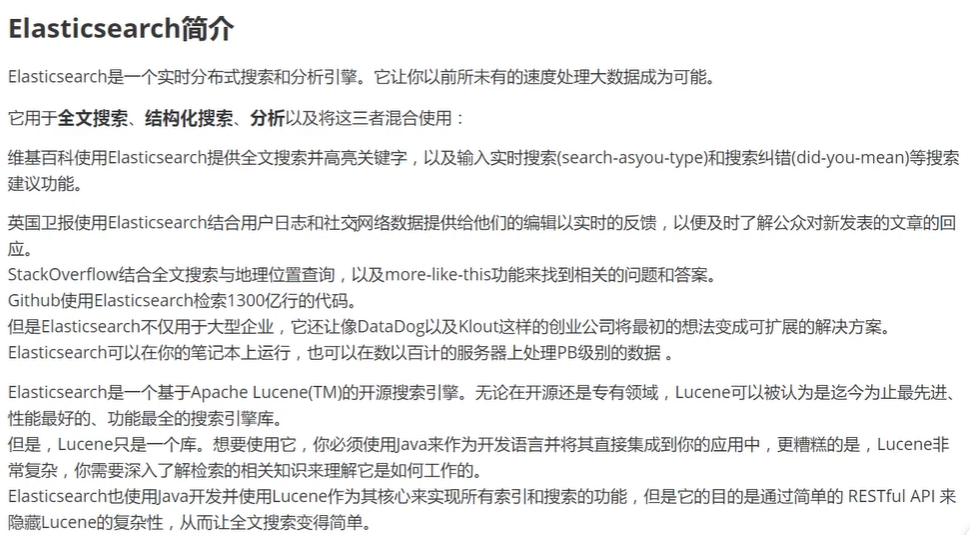
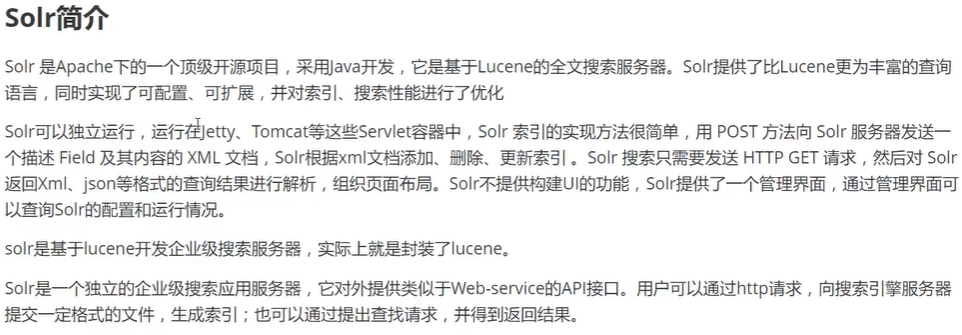
# 概括

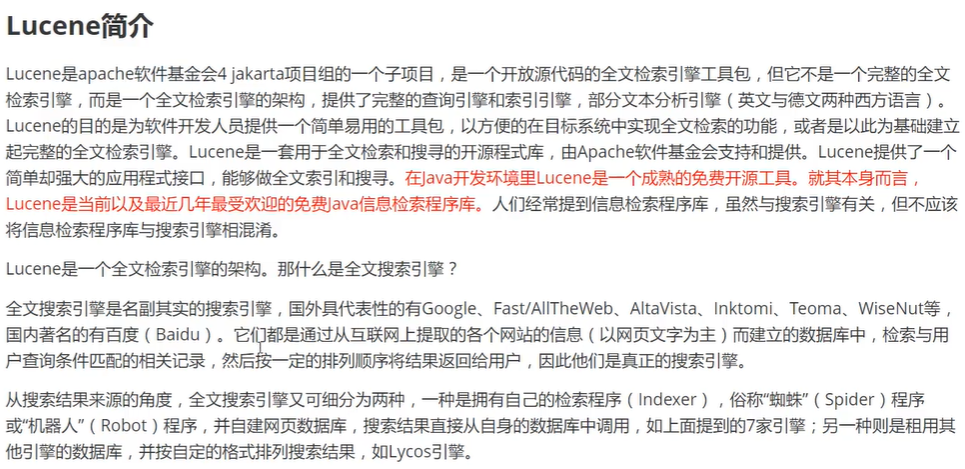




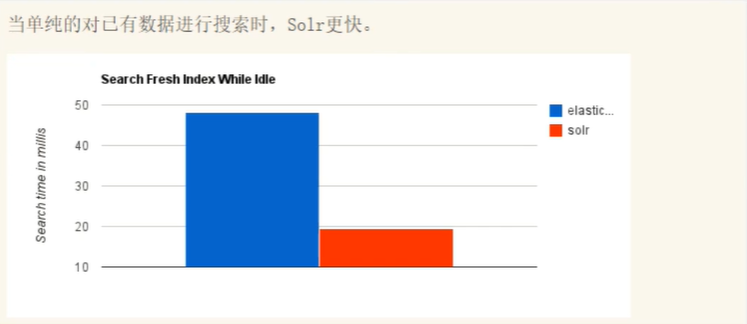
# 货比三家

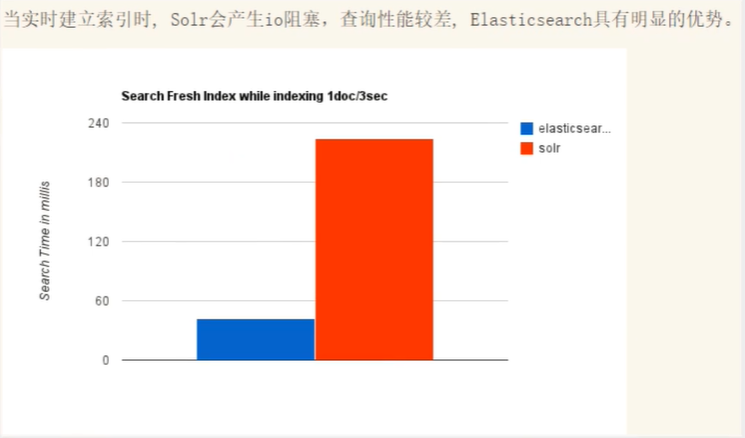


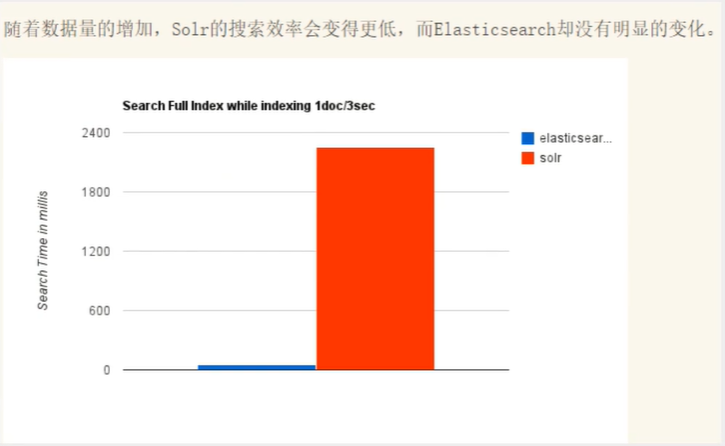




# Es与Solr区别











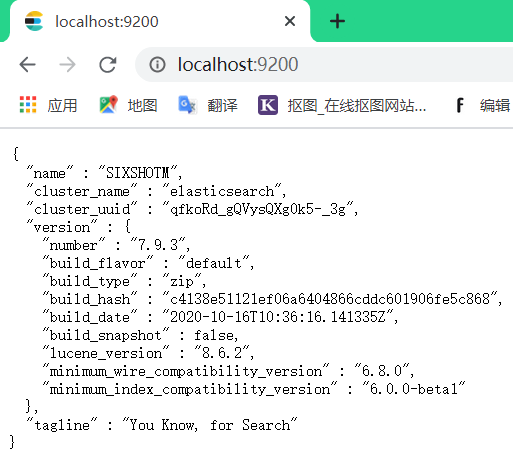
# 下载

华为云

ElasticSearch: https://mirrors.huaweicloud.com/elasticsearch/?C=N&O=D  
logstash: https://mirrors.huaweicloud.com/logstash/?C=N&O=D  
kibana: https://mirrors.huaweicloud.com/kibana/?C=N&O=D

## ES启动

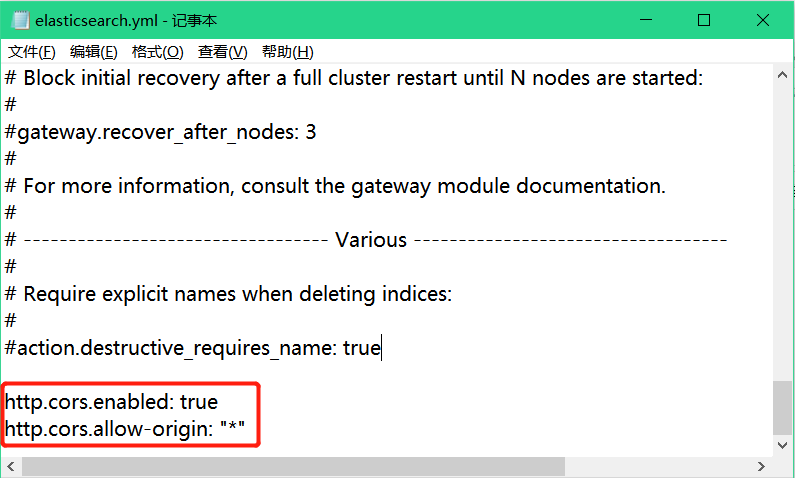
启动ElasticSearch：bin目录的elasticsearch.bat（默认端口9200）



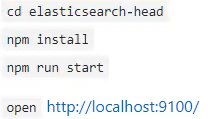
## Head可视化

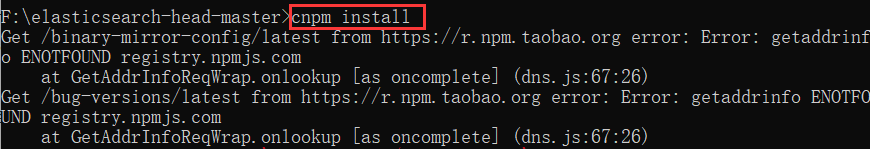
https://github.com/mobz/elasticsearch-head

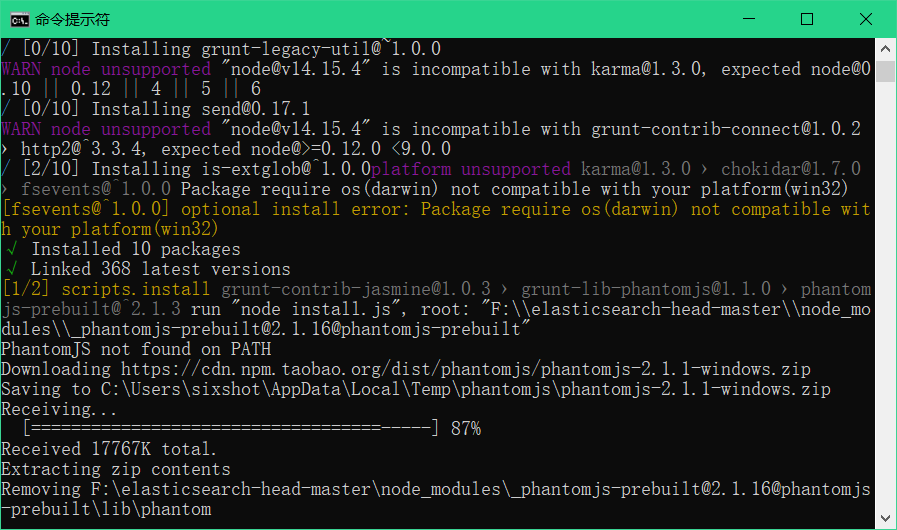
取消跨域访问拦截

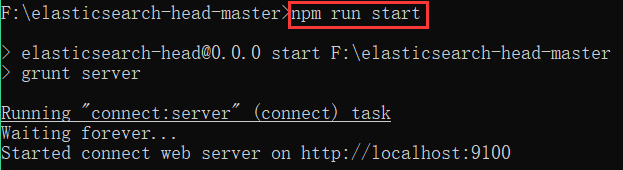


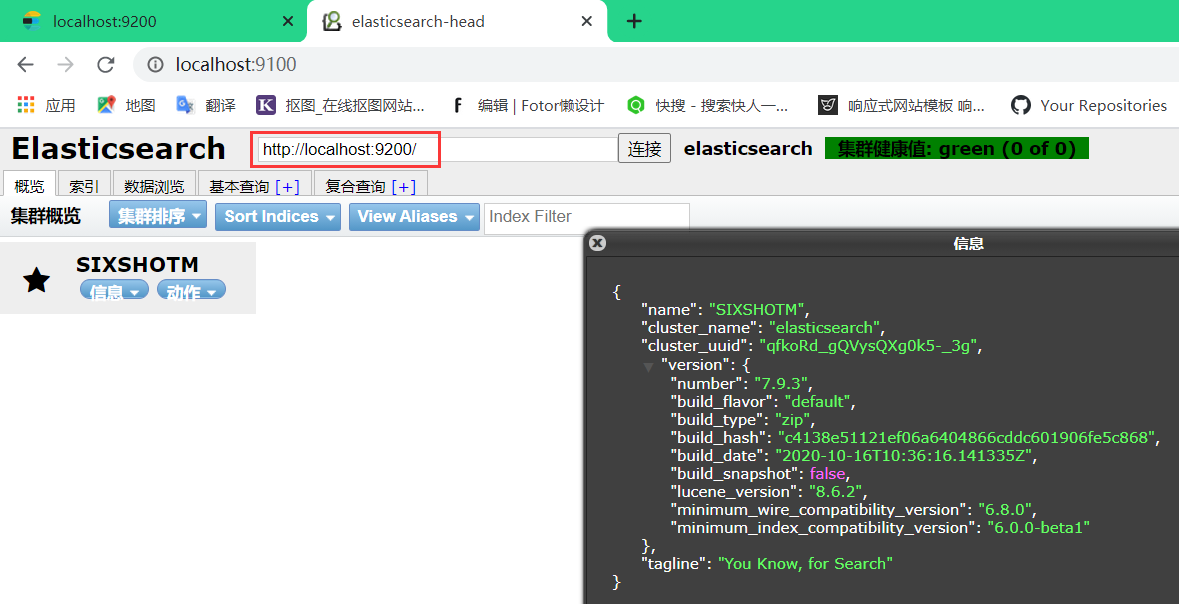
启动（需要npm）



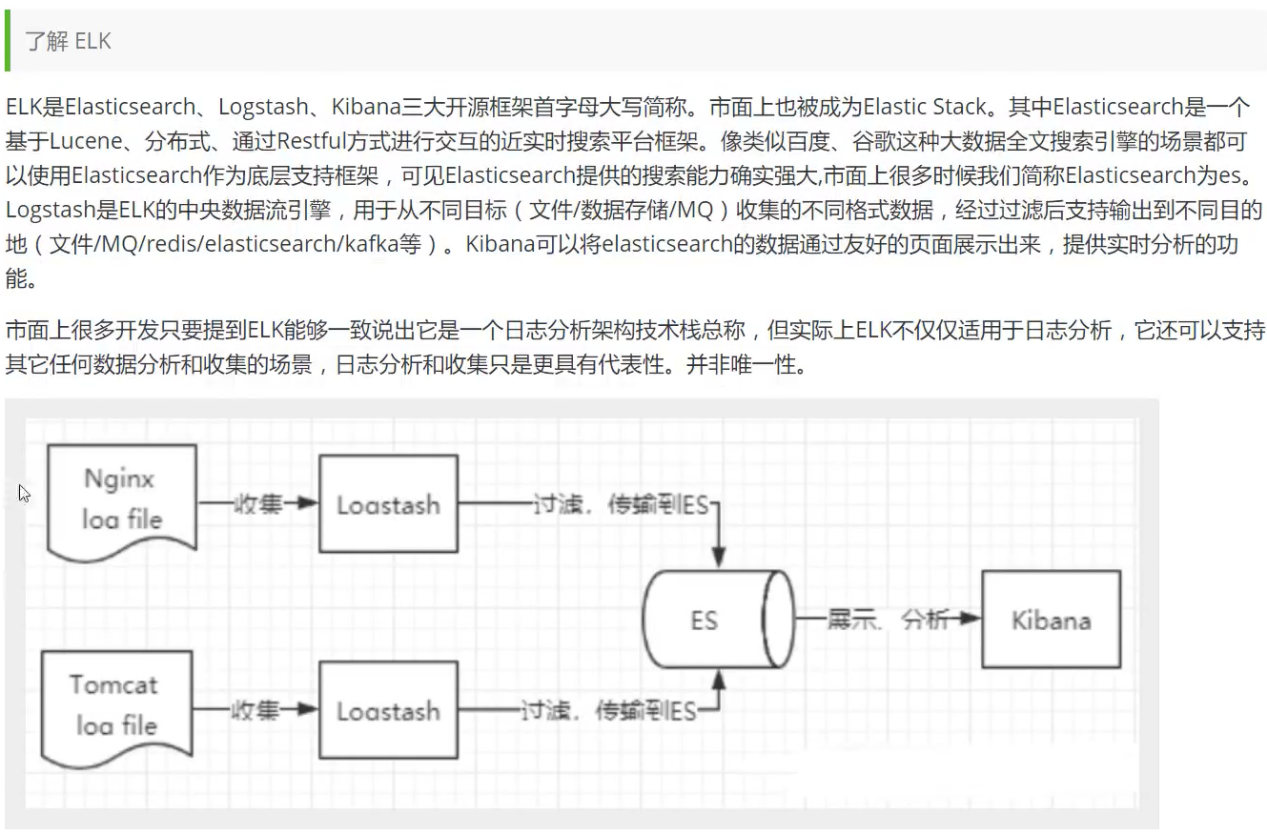




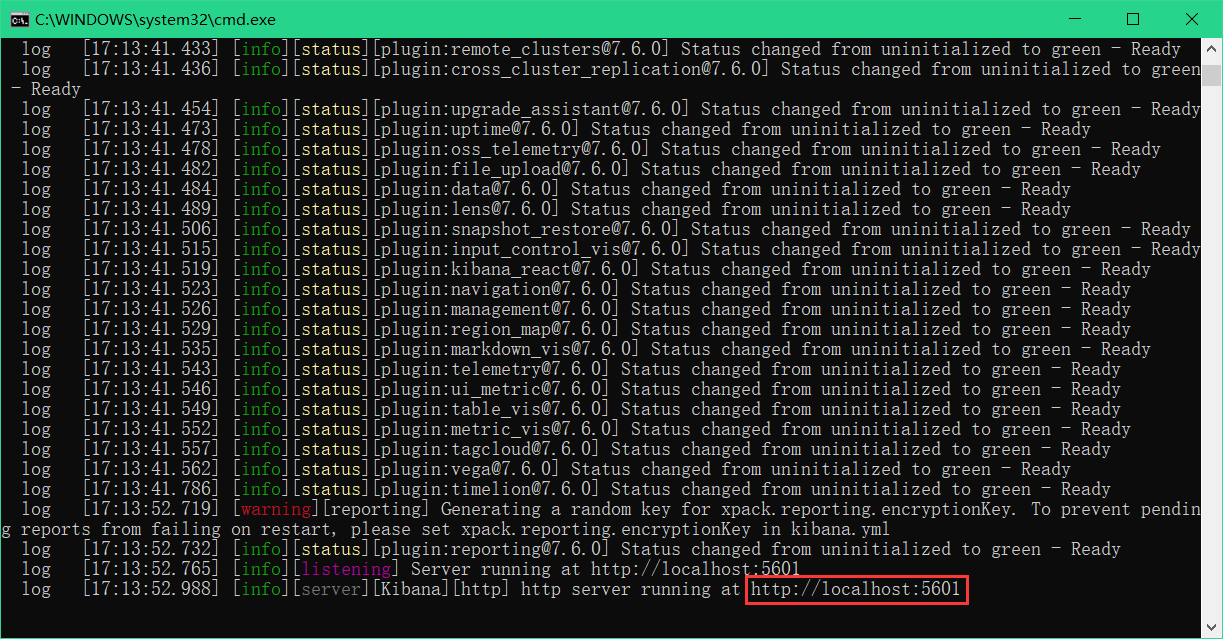


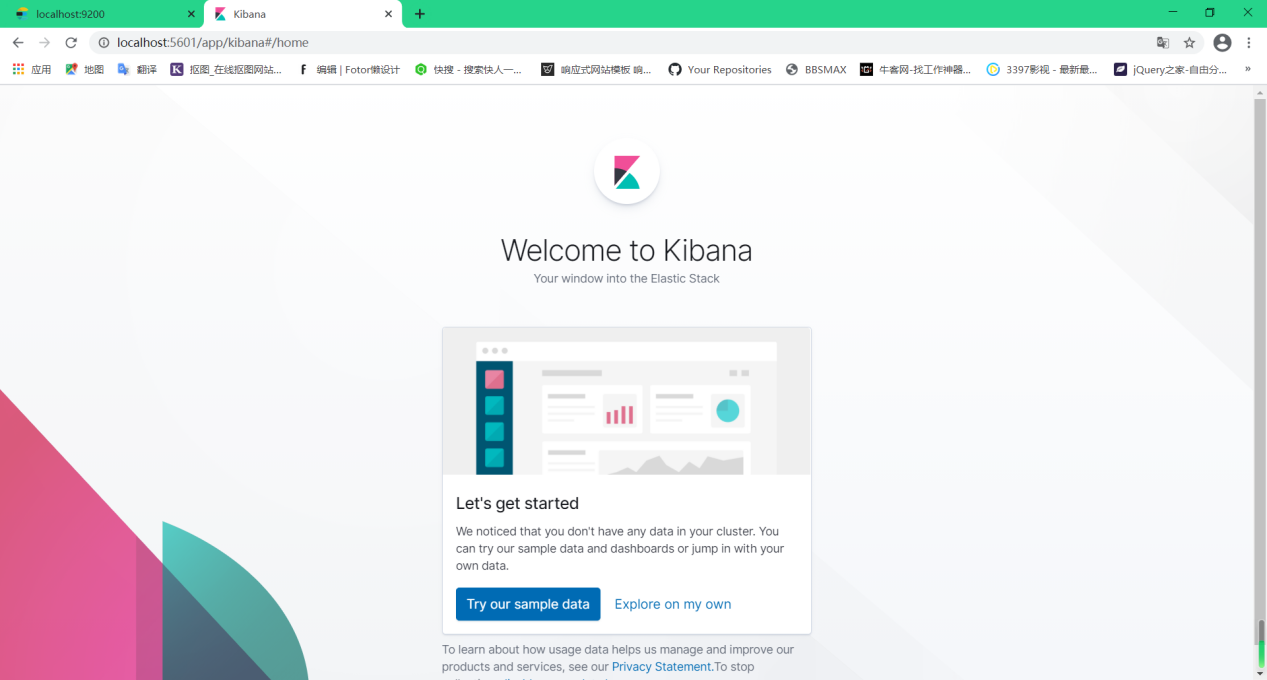


## Kibana



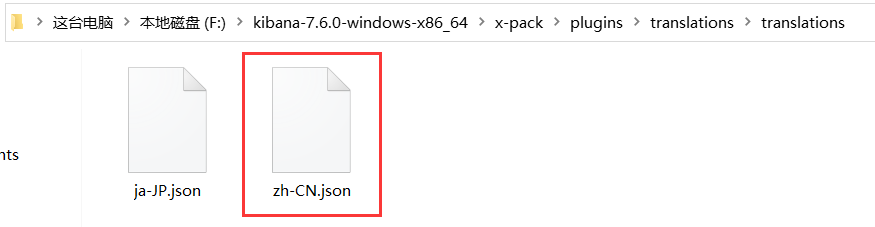
**启动Kibana：**直接运行kibana.bat（默认5601端口）



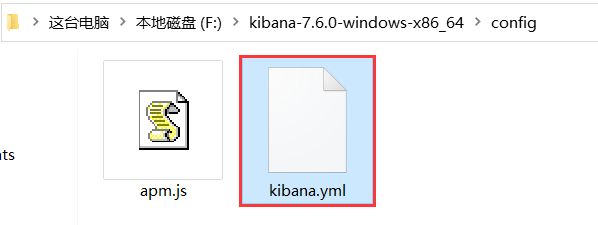


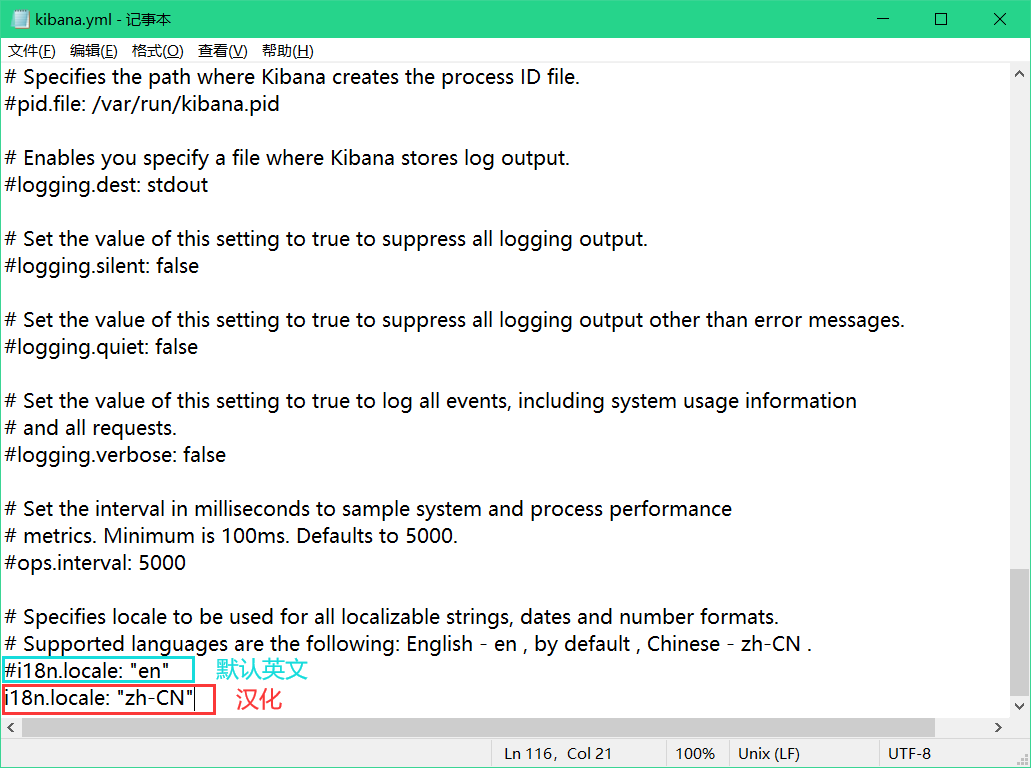
**Kibana汉化：**

目标汉化文件：

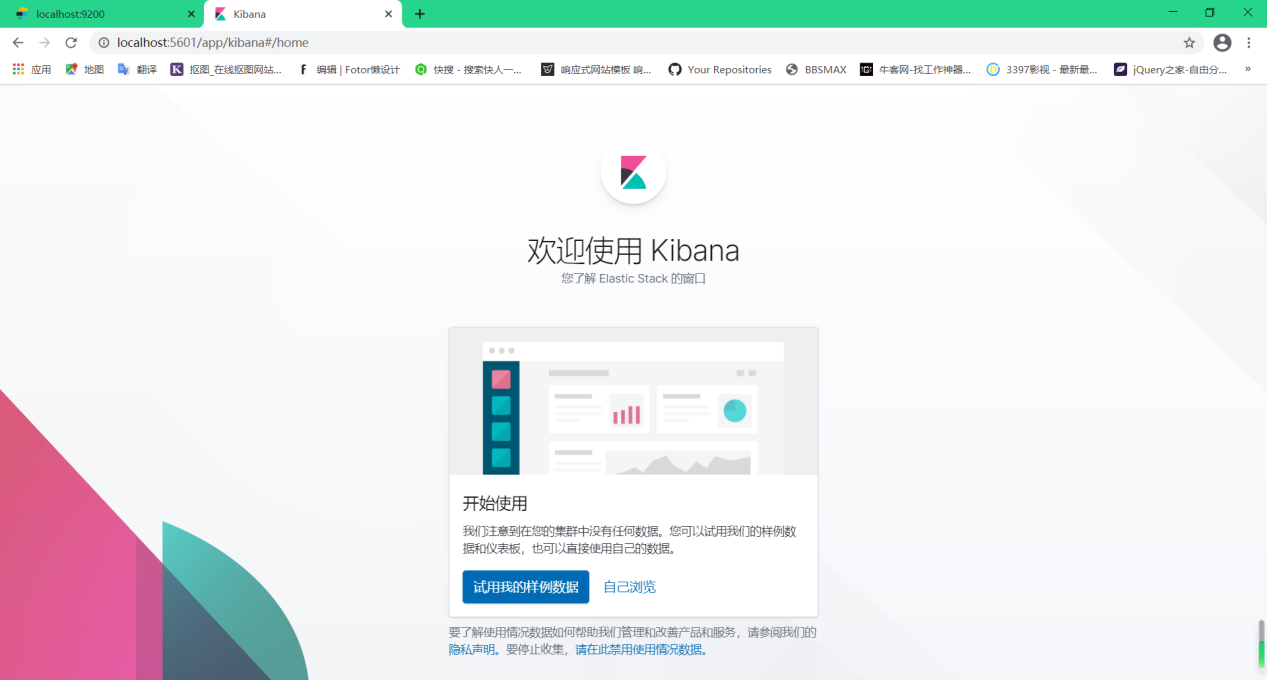


修改配置：





重启kibana

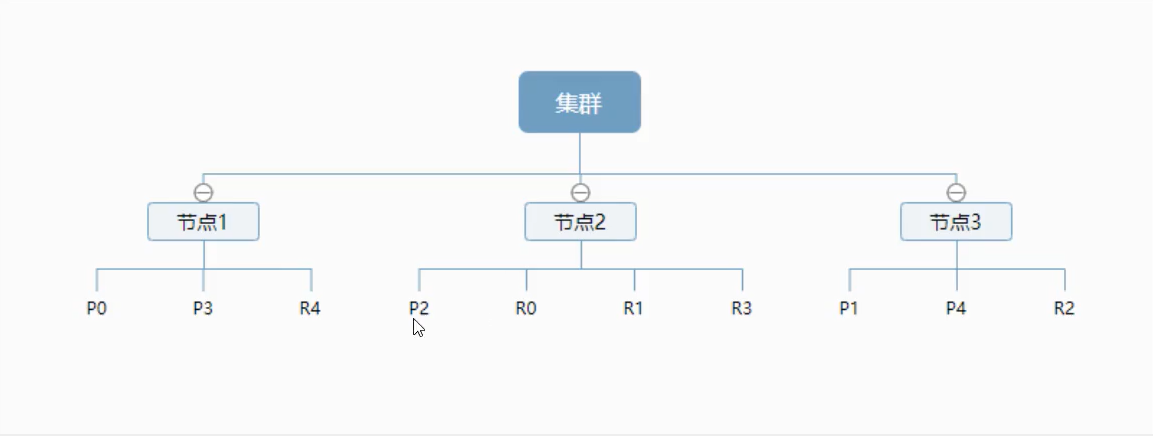


# ES核心概念



默认的集群名称就是ElasticSearch





**索引**

索引就是映射类型的容器，ES的索引是文档的集合，索引储存了映射类型字段和其它设置。然后他们被存储在各个分片上。ElasticSearch的索引是由Lucene索引组成的

**倒排索引**

ES使用倒排索引的结构，用Lucene作为底层。这种结构适合快速全文搜索，一个索引由文档中不同的列表构成，对于每一次都有一个包含他的文档列表。下面引例：

**例子1**



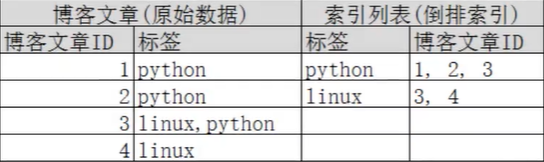


搜索to forever关键词：



**例子2**

没有索引不会找，重复的索引没必要找，从找4次变成2次，效率更高

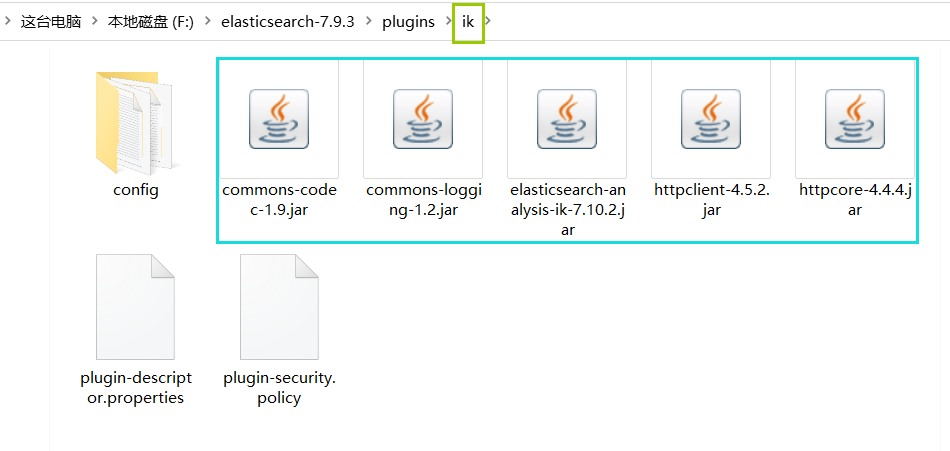


# IK分词器

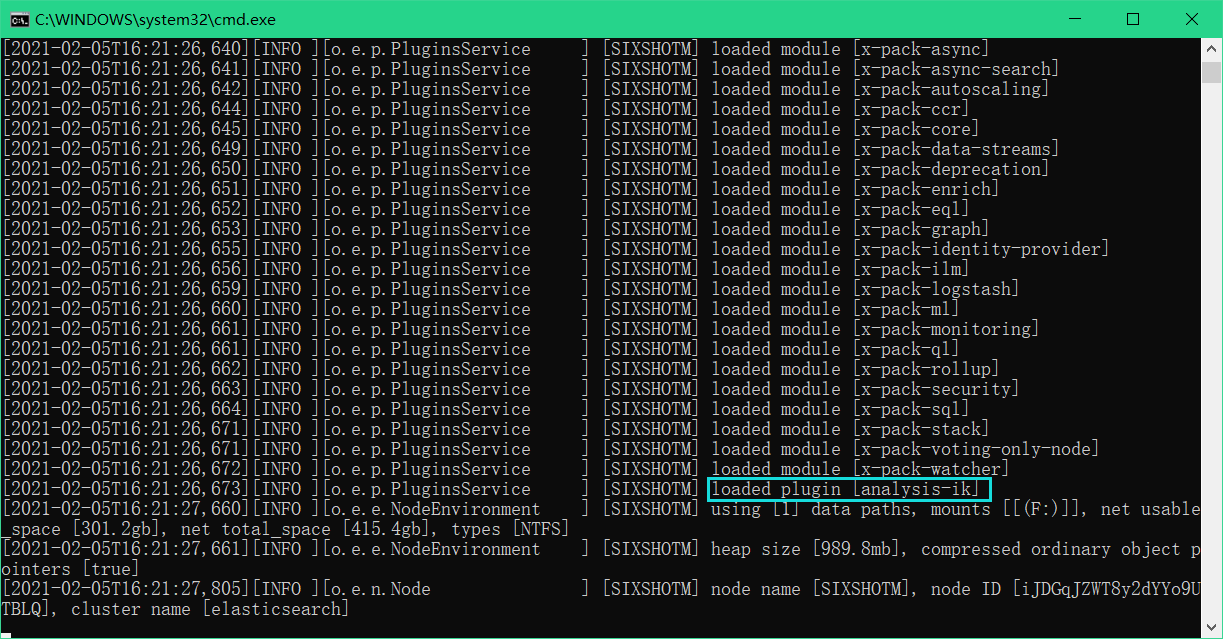
IK Analysis插件将Lucene IK分析器集成到elasticsearch中，支持自定义词典。

下载地址：https://github.com/medcl/elasticsearch-analysis-ik/releases

解压到es的插件目录下



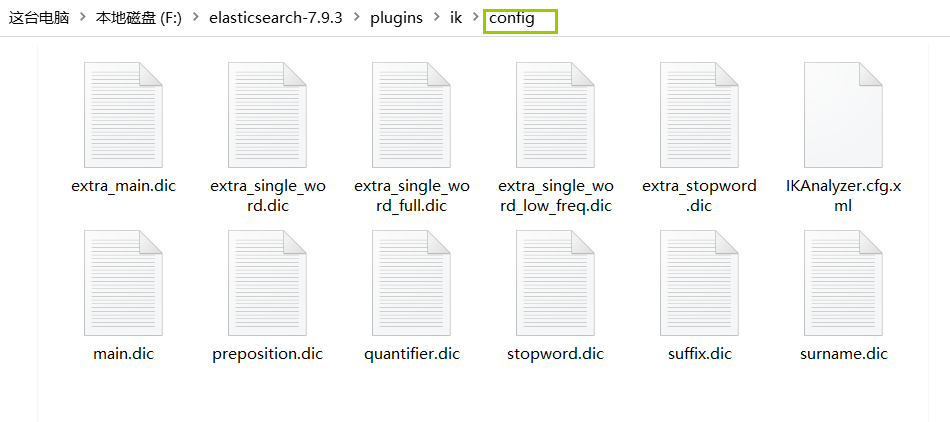
修改ik的properties的ES版本号



**修改词典**

在ik的config放置自定义的dic文件

格式与其它dic类似，都是直接把想要的词组直接写进去，多个换行



最后修改xml配置文件



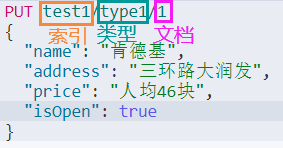
# 增删改查

Rest风格



## 创建索引

创建一个指定类型并且带数据的索引



创建一个空索引，并指定各个字段的类型

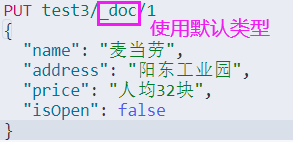


得到索引库信息





创建一个默认类型并且带数据的索引

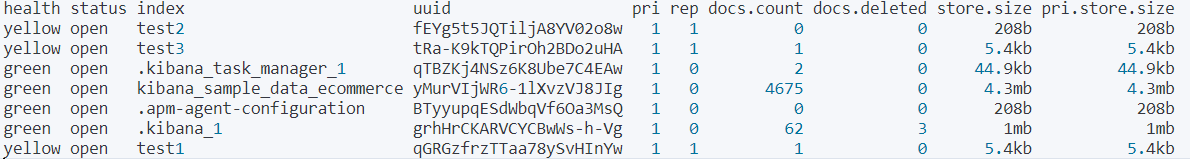


查看该索引库信息



查看所有库信息





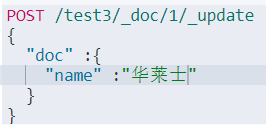
## 修改索引

1. 覆盖型修改





1. 指定字段型修改





## 删除索引





## 花式查询

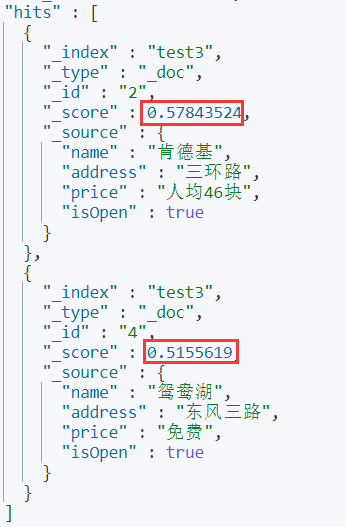
**以文档的名字简单查询：**



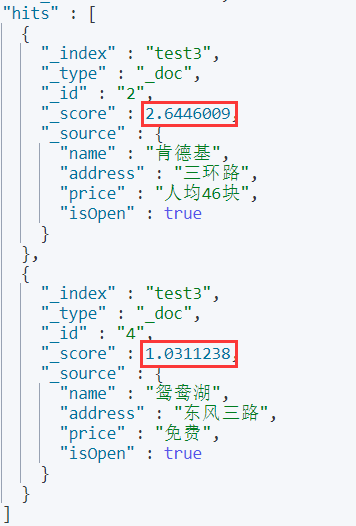


**绑定权重与关键词有关的索引全查询：**





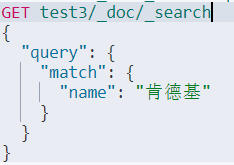




**条件的匹配：**

match是使用了分词器

分词在数组里面都有效





**结果的过滤：**



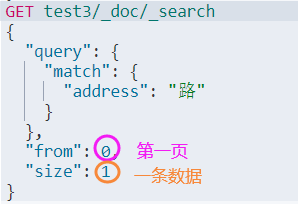


**降序排序：**





**分页查询：**

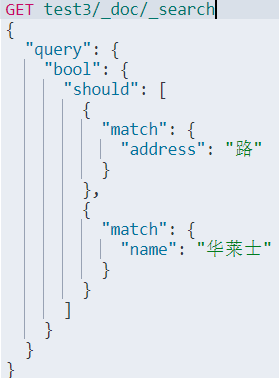


**必须包含查询：**



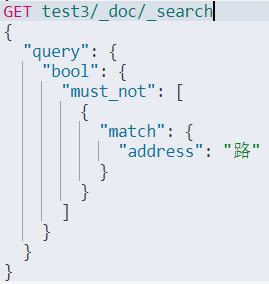


**只要满足指定匹配的全查询：**





**必须不包含查询：**



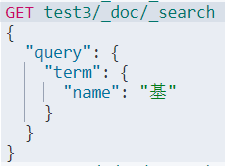


**过滤器：**



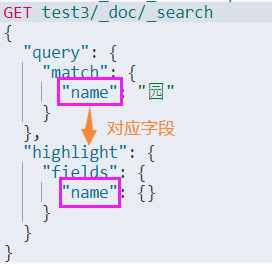


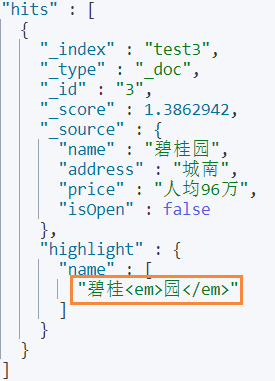
**不分词查询：**





**高亮查询：**





**自定义高亮查询：**





# SpringBoot

## 基本配置

依赖：

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-elasticsearch</artifactId>  
</dependency>

配置类：

@Configuration  
public class ElasticSearchConfig {  
 @Bean  
 public RestHighLevelClient restHighLevelClient(){  
 return new RestHighLevelClient(RestClient.*builder*(  
 new HttpHost("127.0.0.1",9200,"http")));  
 }  
}

## 基础代码

@Resource  
@Qualifier("restHighLevelClient")  
RestHighLevelClient client;  
  
@Test /\* 创建一个空索引 \*/  
void testCreateIndex() throws IOException {  
 //创建索引请求  
 CreateIndexRequest request = new CreateIndexRequest("test1");  
 //获得索引响应  
 CreateIndexResponse response = client.indices().create(request, RequestOptions.*DEFAULT*);  
 //输出结果  
 System.*out*.println(response);  
}  
  
@Test /\* 索引是否存在 \*/  
void testIndexExist() throws IOException {  
 //获得索引响应  
 GetIndexRequest request = new GetIndexRequest("test1");  
 boolean exists = client.indices().exists(request, RequestOptions.*DEFAULT*);  
 //输出结果  
 System.*out*.println(exists);  
}  
  
@Test /\* 删除索引 \*/  
void testDeleteIndex() throws IOException {  
 //获得索引响应  
 DeleteIndexRequest request = new DeleteIndexRequest("test1");  
 AcknowledgedResponse deleted = client.indices().delete(request, RequestOptions.*DEFAULT*);  
 //输出删除结果  
 System.*out*.println(deleted.isAcknowledged());  
}  
  
@Test /\* 插入文档数据 \*/  
void testInsert() throws IOException {  
 //请求目标索引  
 IndexRequest request = new IndexRequest("test1");  
 //设置连接超时时限  
 request.timeout(TimeValue.*timeValueSeconds*(1));  
 //创建文档名（默认是20位由英文数字组成的名）  
 request.id("1");  
 Users users = new Users("普洛意",28,"女");  
 //将数据对象放入这个请求当中的文档去  
 request.source(JSON.*toJSONString*(users), XContentType.*JSON*);  
 //获取响应结果  
 IndexResponse response = client.index(request, RequestOptions.*DEFAULT*);  
 System.*out*.println(response.toString());  
 //请求结束状态  
 System.*out*.println(response.status());  
  
}  
  
@Test /\* 文档是否存在 \*/  
void testDocumentExist() throws IOException {  
 //获取文档响应  
 GetRequest request = new GetRequest("test1","1");  
 //不获取\_source返回的上下文  
 request.fetchSourceContext(new FetchSourceContext(false));  
 //显式指定将返回的存储字段（\_source中的字段）  
 request.storedFields("\_none\_");  
 boolean exists = client.exists(request,RequestOptions.*DEFAULT*);  
 System.*out*.println(exists);  
}  
  
@Test /\* 查询索引 \*/  
void testGetDocument() throws IOException {  
 GetRequest request = new GetRequest("test1","1");  
 GetResponse documentFields = client.get(request, RequestOptions.*DEFAULT*);  
 //获取数据的json字符串  
 System.*out*.println(documentFields.getSourceAsString());  
 //获取包括\_source所有值的json字符串  
 System.*out*.println(documentFields);  
}  
  
@Test /\* 更新数据 \*/  
void testUpdate() throws IOException {  
 UpdateRequest updateRequest = new UpdateRequest("test1","1");  
 updateRequest.timeout("1s");  
 Users users = new Users("小意",27,"女");  
 updateRequest.doc(JSON.*toJSONString*(users),XContentType.*JSON*);  
 UpdateResponse update = client.update(updateRequest, RequestOptions.*DEFAULT*);  
 System.*out*.println(update.status());  
}  
  
@Test /\* 删除文档 \*/  
void testDelete() throws IOException {  
 DeleteRequest deleteRequest = new DeleteRequest("test1","1");  
 deleteRequest.timeout("1s");  
 DeleteResponse deleted = client.delete(deleteRequest, RequestOptions.*DEFAULT*);  
 System.*out*.println(deleted.status());  
}  
  
@Test /\* 批量插入文档 \*/  
void testBulkInsert() throws IOException {  
 BulkRequest bulkRequest = new BulkRequest();  
 bulkRequest.timeout("10s");  
  
 ArrayList<Users> usersList = new ArrayList<>();  
 usersList.add(new Users("卖萌",23,"男"));  
 usersList.add(new Users("达记",20,"男"));  
 usersList.add(new Users("鸭哥",24,"男"));  
 usersList.add(new Users("狗博",21,"男"));  
 usersList.add(new Users("小意",28,"女"));  
  
 for (int i = 0; i < usersList.size(); i++) {  
 bulkRequest.add(new IndexRequest("test1")  
 .id(i+1+"") //文档是字符串  
 .source(JSON.*toJSONString*(usersList.get(i)),XContentType.*JSON*));  
 }  
 BulkResponse bulkResponse = client.bulk(bulkRequest, RequestOptions.*DEFAULT*);  
 System.*out*.println(bulkResponse.hasFailures());  
}

## 高级代码

@Test /\* 复杂查询 \*/  
void testQuerySelect() throws IOException {  
 SearchRequest searchRequest = new SearchRequest("test1");  
 //复杂条件搜索器  
 SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();  
 /\*  
 QueryBuilders：复杂条件构建器  
 term：精确查询 ; match：分词查询  
 \*/  
 MatchQueryBuilder matchQueryBuilder = QueryBuilders.*matchQuery*("name", "意");  
 //把构建好的复杂条件放到这个搜索器内  
 sourceBuilder.query(matchQueryBuilder);  
 /\* 附加高亮 \*/  
 HighlightBuilder highlightBuilder = new HighlightBuilder(); //高亮构建器  
 highlightBuilder.field("name");  
 highlightBuilder.requireFieldMatch(false); //显示多个高亮  
 highlightBuilder.preTags("<p style='color:red'>");  
 highlightBuilder.postTags("</p>");  
 //将构建好的高亮词放置搜索器内  
 sourceBuilder.highlighter(highlightBuilder);  
 /\*  
 附加分页  
 第一页：sourceBuilder.from(1);  
 两条数据：sourceBuilder.size(2);  
 \*/  
 sourceBuilder.timeout(new TimeValue(30, TimeUnit.*SECONDS*));  
 //把构建好的搜索条件放在请求里  
 searchRequest.source(sourceBuilder);  
 //响应结果  
 SearchResponse searchResponse = client.search(searchRequest, RequestOptions.*DEFAULT*);  
 //输出Json字符串数据结果  
 System.*out*.println(JSON.*toJSONString*(searchResponse.getHits()));  
 //拆分hits中的数据至数组  
 SearchHit[] hits = searchResponse.getHits().getHits();  
 //递归每一条文档以Map格式输出（附带高亮显示）  
 for (SearchHit hit : hits) {  
 //获取高亮集合  
 Map<String, HighlightField> highlightFields = hit.getHighlightFields();  
 //在高亮集合中获取字段为name的高亮词  
 HighlightField name = highlightFields.get("name");  
 //获取原集合  
 Map<String, Object> sourceAsMap = hit.getSourceAsMap();  
 //如果高亮词存在  
 if(name!=null){  
 //得到高亮词碎片  
 Text[] fragments = name.fragments();  
 System.*out*.println(fragments.length);  
 String highLightName = "";  
 for (Text fragment : fragments) {  
 //组建高亮词碎片  
 highLightName += fragment;  
 }  
 //更新到原集合对应的词  
 sourceAsMap.put("name", highLightName);  
 } System.*out*.println(sourceAsMap);  
 }  
}

# Jsoup爬虫

<dependency>  
 <groupId>org.jsoup</groupId>  
 <artifactId>jsoup</artifactId>  
 <version>1.13.1</version>  
</dependency>

public class HtmlParse {  
 public static void main(String[] args) throws IOException {  
 String url = "https://search.jd.com/Search?keyword=csdn&enc=utf-8";  
 Document parse = Jsoup.*parse*(new URL(url), 10000);  
 Element element = parse.getElementById("J\_goodsList");  
 Elements lis = element.getElementsByTag("li");  
 for (Element li : lis) {  
 String src = li.getElementsByTag("img").eq(0).attr("data-lazy-img");  
 String price = li.getElementsByClass("p-price").eq(0).text();  
 String context = li.getElementsByClass("p-name").eq(0).text();  
 System.*out*.println("<------------------------------------------------->");  
 System.*out*.println(src);  
 System.*out*.println(price);  
 System.*out*.println(context);  
 }  
 }  
}