安装

下载ElasticSearch压缩包，bin目录启动

安装可视化界面

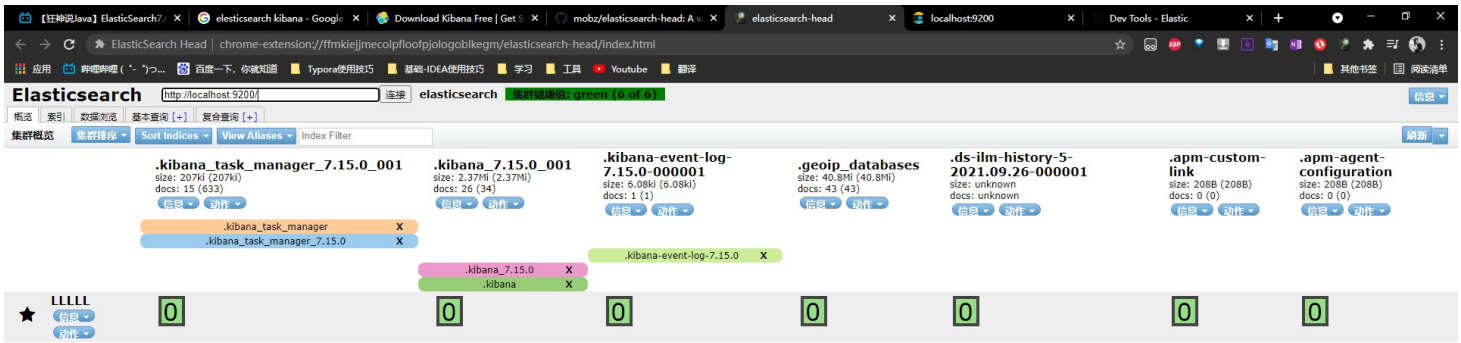
- 下载elasticsearch-head压缩包
- 在cmd环境下安装依赖

```
npm install
npm run start
```

- 存在跨域问题,配置ES

```
http.cors.enabled: true
http.cors.allow-origin: "*"
```

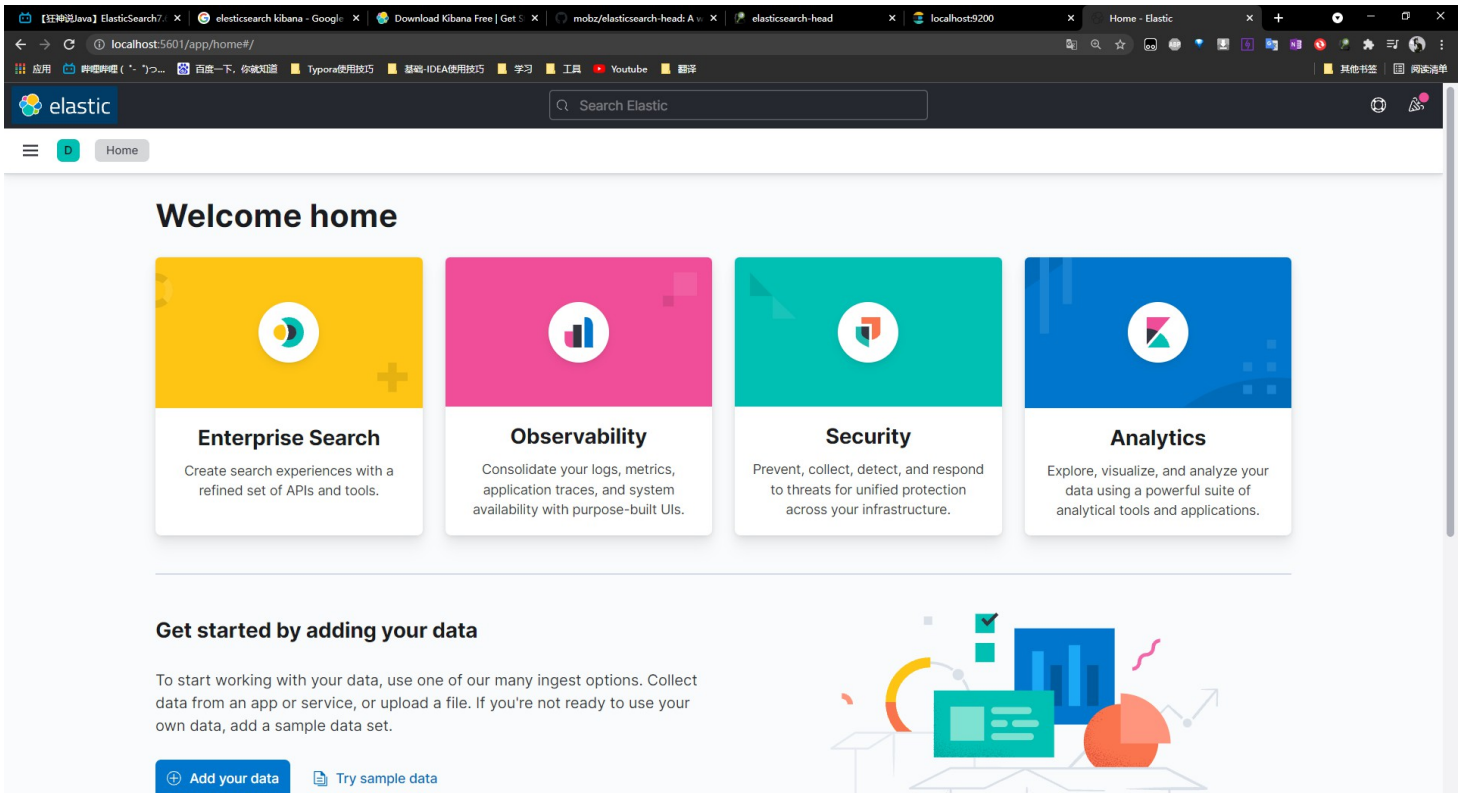
或者直接在chrome浏览器搜索ElasticSearch-head插件，也能够实现可视化界面



安装Kibana

Kibana是一个针对ElasticSearch的开源分析及可视化平台，用来搜索，查看交互存储在ElasticSearch索引的数据。

- 下载压缩包，解压
- 启动进入界面



ES核心概念

ElasticSearch面向文档

Relational DB	ElasticSearch
数据库(database)	索引(indices)
表(table)	types
行(rows)	documents
字段(columns)	fields

elasticSearch(集群)中可以包含多个索引（数据库），每个索引中可以包含多个类型（表），每个类型下又包含多个文档（行），每个文档中又包含多个字段（列）。

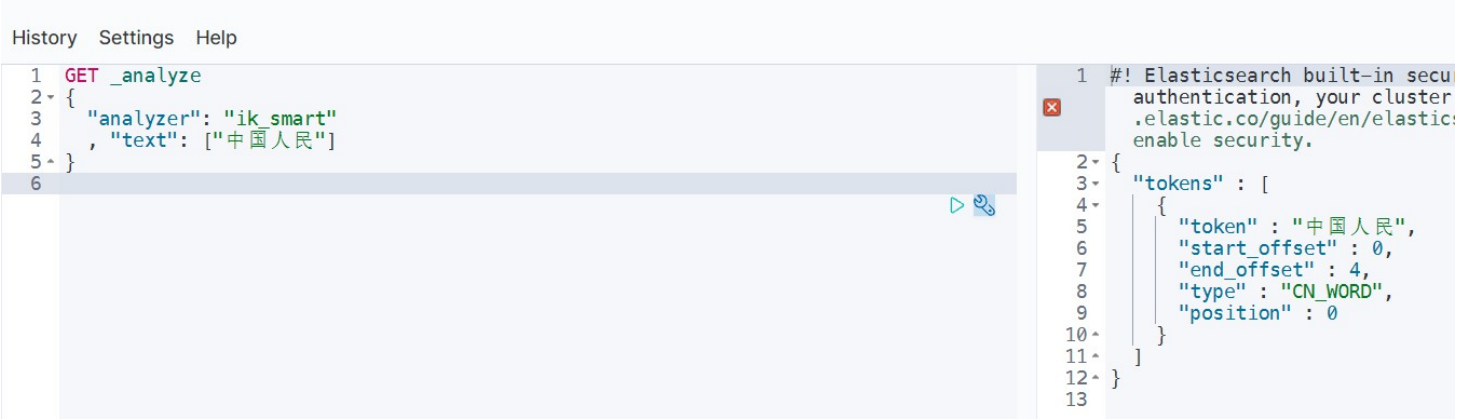
> 倒排索引

elasticsearch使用的是一种称为倒排索引的结构，采用Lucene倒排索引作为底层。这种结构适用于快速的全文搜索，一个索引由文档中所有不重复的列表构成，对于每个词，都有一个包含它的文档列表

IK分词器

分词：即把一段中文或者别的文字划分为一个个关键字，在搜索时把自己的信息进行分词，然后进行匹配操作
ik提供了两种分词算法：ik_smart（最少切分）和ik_max_word（最细粒度划分）

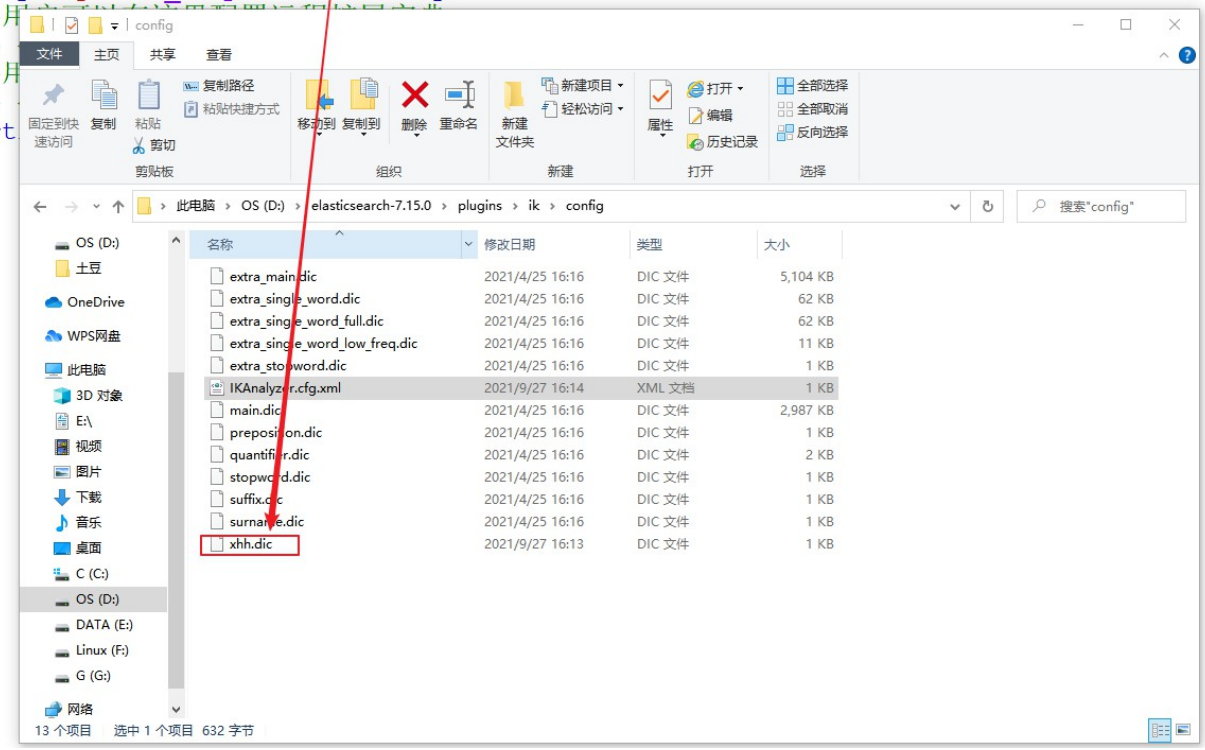
- 下载压缩包解压到 `\elasticsearch-7.15.0\plugins\ik` 目录中
- 重启elasticsearch
    - ik_smart



- ik_max-word

> 配置自己的分词

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>IK Analyzer 扩展配置</comment>
    <!--用户可以在这里配置自己的扩展字典 -->
    <entry key="ext_dict">xhh.dic</entry>
     <!--用户可以在这里配置自己的扩展停止词字典-->
    <entry key="ext_stopwords"></entry>
    <!--用
    <!--
    <!--用
    <!--
</propert
```



Rest风格说明

基本rest命令说明:

method	url地址	描述
PUT	localhost:9200/索引名称/类型名称/文档id	创建文档（指定文档id）
POST	localhost:9200/索引名称/类型名称	创建文档（随机文档id）
POST	localhost:9200/索引名称/类型名称/文档id/_update	修改文档
DELETE	localhost:9200/索引名称/类型名称/文档id	删除文档
GET	localhost:9200/索引名称/类型名称/文档id	查询文档（通过文档id）
POST	localhost:9200/索引名称/类型名称/_search	查询所有数据

> 基础测试

- 创建一个索引

PUT /索引名称/类型名称/文档id
{请求体}

- 创建索引，指定类型

```
PUT /test2
{
      "mappings" : {
      "properties" : {
        "age" : {
          "type" : "long"
        },
        "birthday" : {
          "type" : "date"
        },
        "name" : {
          "type" : "text"
        }
      }
    }
}
```

文档的基本操作

- 添加数据

```
PUT /xhh2/user/1
{
 "name":"李四",
 "age":10,
 "desc":"你记得吗"
}
```

- 更新数据

```
POST /xhh2/user/1/_update
{
  "doc":{
    "name":"陈平安"
  }
}

PUT /xhh2/user/1/
{
  "doc":{
    "name":"张三"
  }
}
```

PUT和POST在更新上的区别:

- put如果不传值就会被覆盖
- post灵活性更好，可以只修改一处的值

kibana推荐写法： `POST /{index}/_update/{id}`

- 查询数据

```
GET /xhh2/user/1
或者使用
GET /xhh2/user/_search?q=desc:你
```

- 高级查询

```
GET /xhh2/user/_search
{
  "query":{
    "match": {
      "name": "李四"
    }
  },
  "_source": ["name","desc","age"],
  "sort":[
    {
      "age":{
        "order":"asc"
      }
    }
  ],
  "from": 0,
  "size": 2
}
```

```
GET /xhh2/user/_search
{
  "query":{
    "match": {
      "name": "李四"
    }
  },
  "_source": ["name","desc","age"],
  "sort":[
    {
      "age":{
        "order":"asc"
      }
    }
  ],
  "from": 0,
  "size": 2
}
```

match:匹配；匹配查询条件

只显示_source中的字段

sort排序，按照age asc排序

分页，from代表位置，size代表显示个数

- 多重条件查询(bool)

must相当于and,should相当于or,must_not相当于not

```
GET /xhh2/user/_search
{
  "query":{
    "bool": {
      "must": [
        {
          "match": {
            "name": "李四"
          }
        },
        {
          "match": {
            "age": "25"
          }
        }
      ]
    }
  }
}

//或者使用
      [
        {
          "match": {
            "name": "李四 陈"
          }
        }
      ]
//多个条件使用空格隔开，只要满足一个条件就可以查询
```

- filter过滤

```
      "filter": [
        {
          "range": {
            "age": {
              "gte": 10,
              "lte": 80
            }
          }
        }
      ]
```

- 精确查询
  - term不会对查询条件进行分词
  - keyword不会对存储的数据进行分词
  - match进行模糊查询

    - match+keyword不会进行模糊查询
- 高亮

```
GET /testdb/_doc/_search
{
  "query":{
    "match":{
      "name":"张三说Java1"
    }
  },
  "highlight":{
  "pre_tags":"<p class='key' style='color:red'>",
  "post_tags":"</p>",
  "fields":{
    "name":{}
    }
  }
}
```

集成Springboot

> maven依赖

```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
    <version>7.15.0</version>
</dependency>
```

- 编写配置类

```
@Configuration
public class ElaticSearchConfig {

    @Bean
    public RestHighLevelClient restHighLevelClient(){
        RestHighLevelClient client =  new RestHighLevelClient(
                RestClient.builder(
                        new HttpHost("localhost", 9200, "http")));
        return client;
    }
}
```

- 测试

```java
package com.xhh;

import com.alibaba.fastjson.JSON;
import com.sun.org.apache.bcel.internal.generic.NEW;
import com.xhh.pojo.User;
import org.apache.lucene.util.QueryBuilder;
import org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest;
import org.elasticsearch.action.bulk.BulkRequest;
import org.elasticsearch.action.bulk.BulkResponse;
import org.elasticsearch.action.get.GetRequest;
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.action.support.master.AcknowledgedResponse;
import org.elasticsearch.action.update.UpdateRequest;
import org.elasticsearch.action.update.UpdateResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.core.GetSourceRequest;
import org.elasticsearch.client.core.GetSourceResponse;
import org.elasticsearch.client.indices.CreateIndexRequest;
import org.elasticsearch.client.indices.CreateIndexResponse;
import org.elasticsearch.client.indices.GetIndexRequest;
import org.elasticsearch.common.xcontent.XContentType;
import org.elasticsearch.core.TimeValue;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.index.query.TermQueryBuilder;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.fetch.subphase.FetchSourceContext;
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.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.TimeUnit;

@SpringBootTest
class ElasticsearchApiApplicationTests {

    @Autowired
    private RestHighLevelClient restHighLevelClient;

    //测试索引的创建
    @Test
    void testCreateIndex() throws IOException {
        //创建索引
        CreateIndexRequest request = new CreateIndexRequest("xhh_index");
```

```
    //客户端执行请求 获得响应
    CreateIndexResponse createIndexResponse = restHighLevelClient.indices().create(request,
    System.out.println(createIndexResponse);
}

//测试获取索引，判断是否存在
@Test
void testGetIndex() throws IOException {
    GetIndexRequest getIndexRequest = new GetIndexRequest("xhh_index");
    boolean exists = restHighLevelClient.indices().exists(getIndexRequest, RequestOptions.DE
    System.out.println(exists);
}

//测试删除索引
@Test
void testDeleteIndex() throws IOException {
    DeleteIndexRequest request = new DeleteIndexRequest("xhh_index");
    AcknowledgedResponse delete = restHighLevelClient.indices().delete(request,RequestOptior
    System.out.println(delete.isAcknowledged());
}

//添加文档
@Test
void addSources() throws IOException {
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
    //创建对象
    User user = new User("Lucy", 26, simpleDateFormat.format(new Date()));
    //创建请求
    IndexRequest request = new IndexRequest("xhh_index");
    //规则 put /xhh_index/_doc/id
    request.id("6");
    request.timeout("1s");

    //将数据放入到json请求中
    request.source(JSON.toJSONString(user), XContentType.JSON);

    //客户端发送数据,获取响应结果
    IndexResponse indexResponse = restHighLevelClient.index(request, RequestOptions.DEFAULT)
    System.out.println(indexResponse);
}

//判断文档是否存在
@Test
void testIsExists() throws IOException {
    GetRequest getRequest = new GetRequest(
            "xhh_index",
            "1");
    //不获取返回得_source的上下文
    getRequest.fetchSourceContext(new FetchSourceContext(false));
    getRequest.storedFields("_none_");
```

```
    boolean exists = restHighLevelClient.exists(getRequest, RequestOptions.DEFAULT);
    System.out.println(exists);
}

//获取文档的信息
@Test
void testGetSource() throws IOException {
    GetSourceRequest request = new GetSourceRequest("xhh_index", "1");
    GetSourceResponse response =
            restHighLevelClient.getSource(request, RequestOptions.DEFAULT);
    System.out.println(response.getSource());
    System.out.println(response);
}

//更新文档信息
@Test
void updateSource() throws IOException {
    UpdateRequest update = new UpdateRequest("xhh_index", "1")
            .doc("name","李四");
    UpdateResponse updateResponse = restHighLevelClient.update(update, RequestOptions.DEFAUL
    System.out.println(updateResponse);
}

//批量操作
@Test
void testBulkSource() throws IOException {
    BulkRequest bulkRequest = new BulkRequest();
    ArrayList<Object> arrayList = new ArrayList<>();
    arrayList.add(new User("张三",23,"1999-02-10"));
    arrayList.add(new User("王五",23,"1999-03-10"));
    arrayList.add(new User("赵六",23,"1999-04-10"));
    arrayList.add(new User("老七",23,"1999-05-10"));

    for(int i=0;i<arrayList.size();i++){
        bulkRequest.add(new IndexRequest("xhh_index").id(i+2+"")
        .source(JSON.toJSONString(arrayList.get(i)),XContentType.JSON));
    }
    BulkResponse bulkResponse = restHighLevelClient.bulk(bulkRequest, RequestOptions.DEFAULT
    System.out.println(bulkResponse);
}

//条件查询
@Test
void testSearch() throws IOException {
    SearchRequest searchRequest = new SearchRequest("xhh_index");
    //构建搜索条件
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();

    //查询条件，可以使用queryBuilders工具来实现
    //termQuery精确查询
    TermQueryBuilder termQueryBuilder = QueryBuilders.termQuery("name.keyword", "Lucy");
```

```
        searchSourceBuilder.query(termQueryBuilder);
        searchSourceBuilder.timeout(new TimeValue(60, TimeUnit.SECONDS));
        searchRequest.source(searchSourceBuilder);
        SearchResponse search = restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT
        System.out.println(JSON.toJSONString(search.getHits()));
    }
}
```

实战