# 定义

ES：是一个开源的高扩展的分布式全文检索引擎。它可以近乎实时的存储、检索数据，本身扩展性很好，可以扩展到上百台服务器，处理PB级别的数据。也可以使用lucene作为你核心来实现所有索引和搜索的工功能，但它的目的是通过简单的RESTfulAPI来隐藏Lucene的复杂性，从而让全文搜索变得简单。

ELK: 是指elasticsearch、Logstash(中央数据流引擎，收集数据后通过过滤输出到不同的目的地[文件、redis\es])、Kibana(将es数据友好展示，提供实时分析功能)。

网址：<https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/nest-getting-started.html>

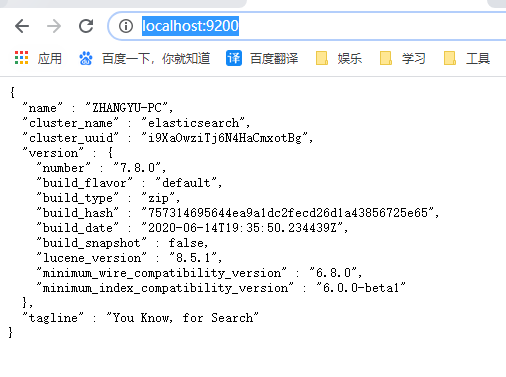
# 安装

1. 在config中elasticsearch.yml 中，最后添加下面的代码，允许跨域访问。

http.cors.enabled: true

http.cors.allow-origin: "\*"

1. 在bin中双击 elasticsearch.bat启动elasticsearch，浏览器输入 <http://localhost:9200/> ，显示下图，表示成功



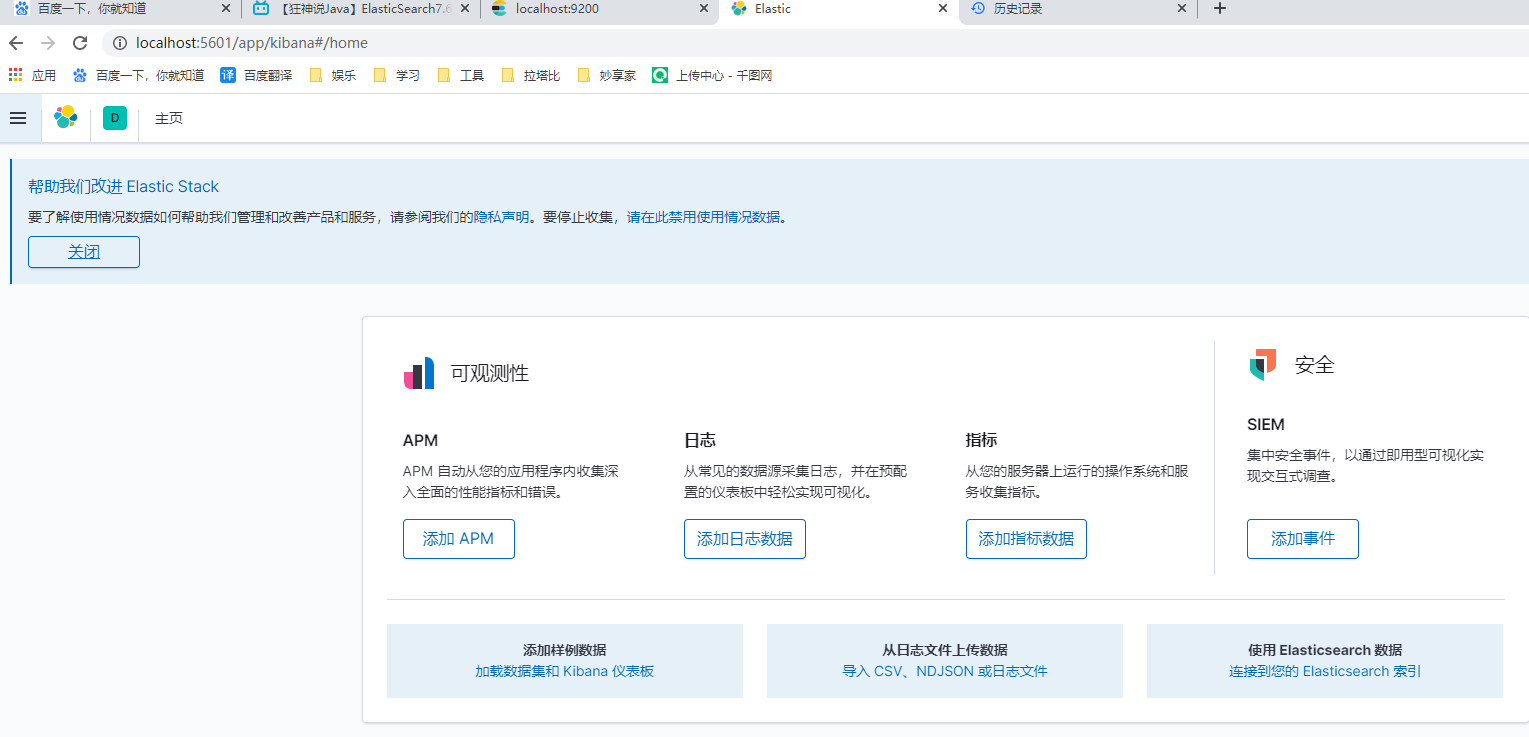
1. (过时)运行elasticsearch-head-master(es管理界面)，需要先安装node.js ,然后安装grunt

npm install –g grunt-cli

npm install

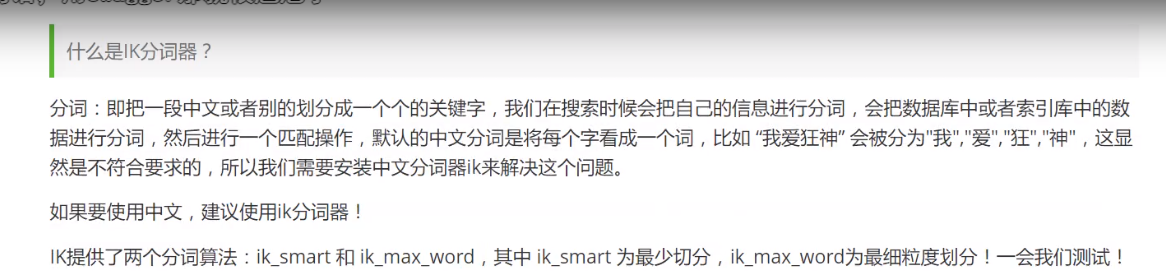
grunt server (启动管端界面, 浏览器输入 <http://localhost:9100/>)

1. 下载kibana，在config中的kibana.yml，最后一句添加 i18n.locale: "zh-CN"，界面汉化,浏览器输入<http://localhost:5601/app/kibana#/home>，打开首页

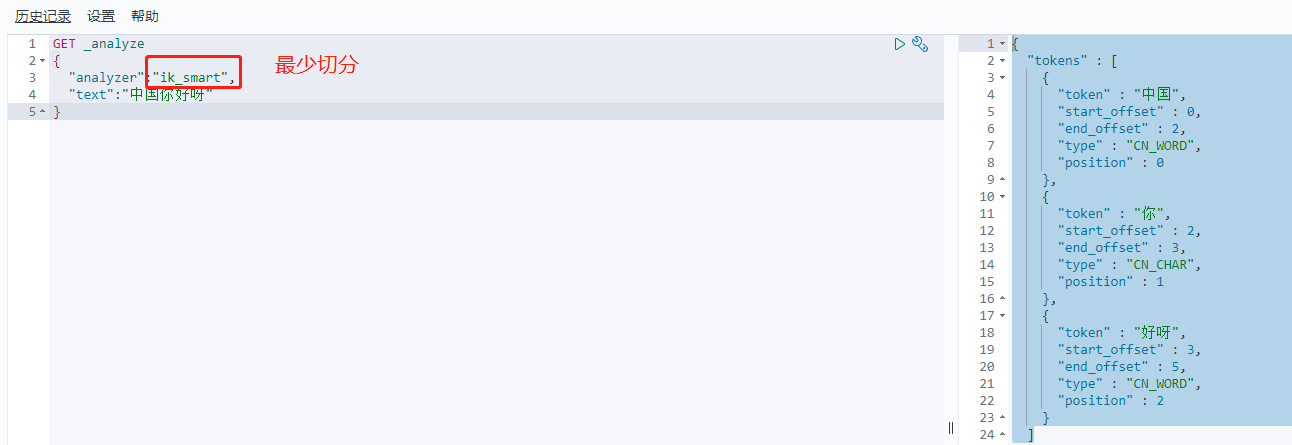


1. 下载分词器ik <https://github.com/medcl/elasticsearch-analysis-ik/releases>,然后在elasticsearch\elasticsearch-7.8.0\plugins 中新建ik文件夹，把代码解压到这里

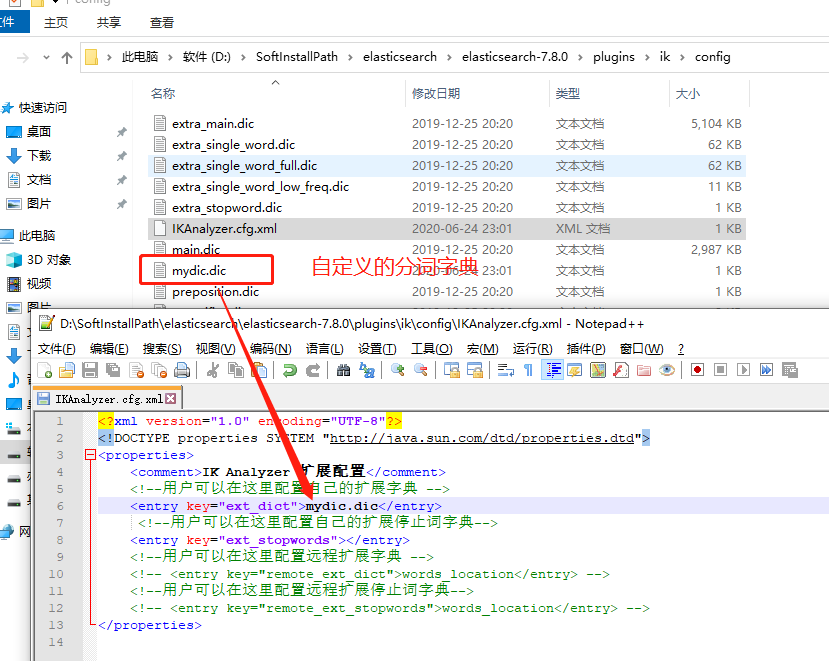
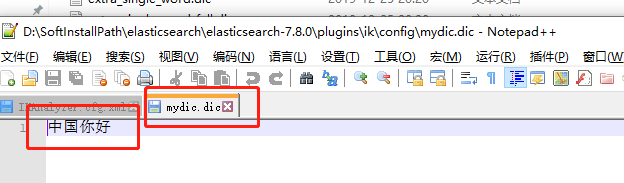
可以在elasticsearch 的bin的 cmd中，使用elasticsearch-plugin list查看是否安装成功

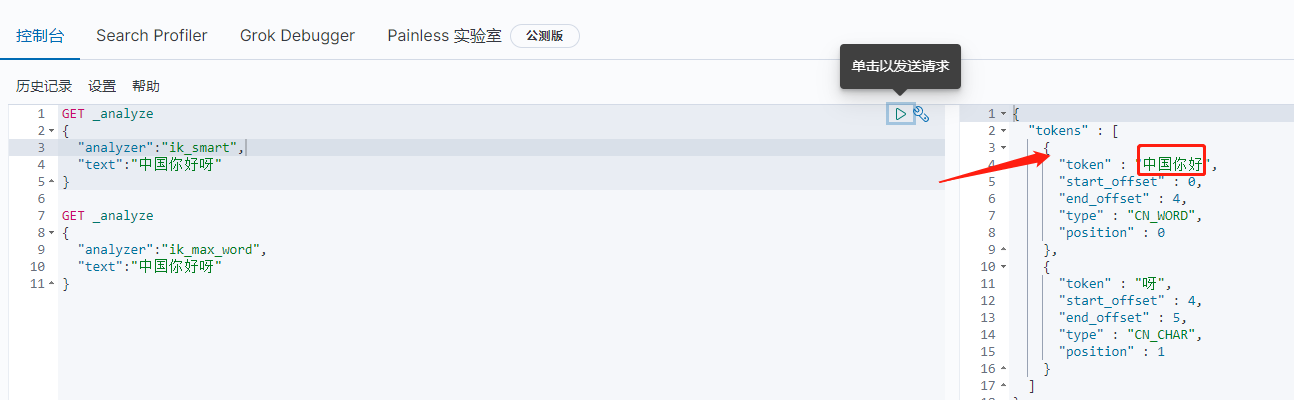


在kibana上查看分词结果(REST风格)：



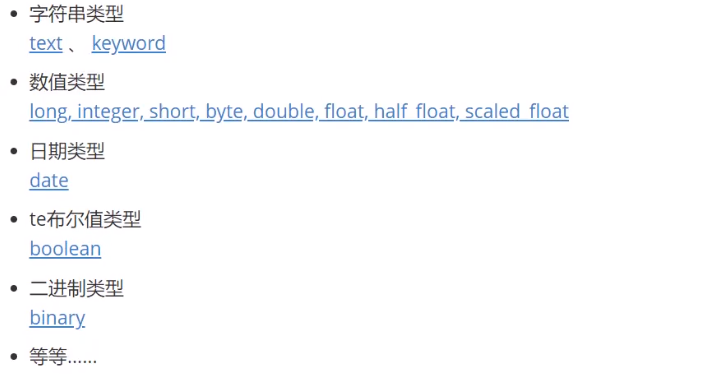
ik分词器上添加字典：elasticsearch\elasticsearch-7.8.0\plugins\ik\config中配置，添加自定义字典mydic.dic，然后在IKAnalyzer.cfg.xml中配置就行了

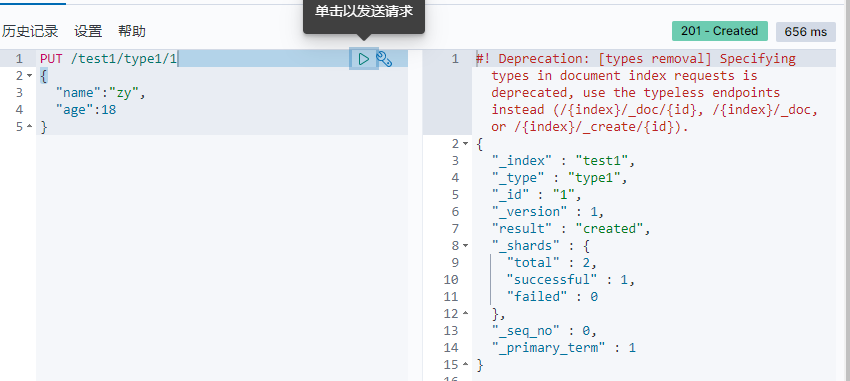
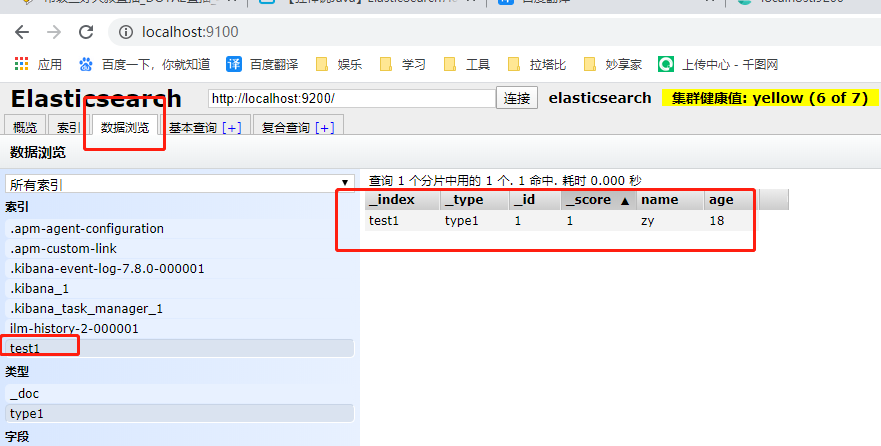


# 基本操作

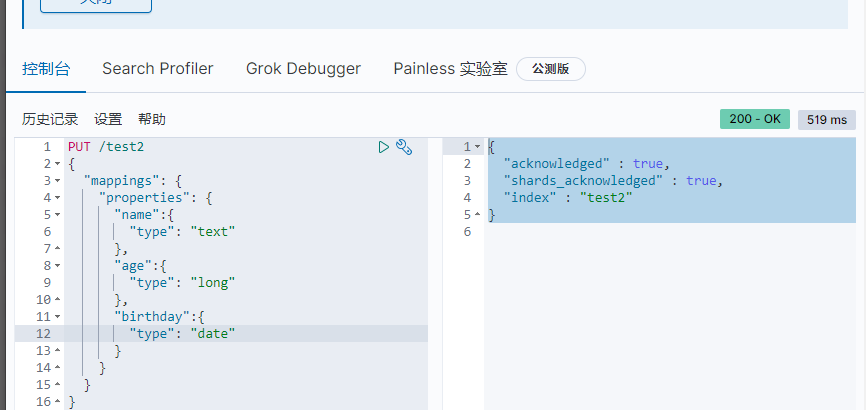
REST风格：

创建索引(put/post都行):，如果指定数据已存在，就覆盖，但是如果字段漏掉了，该字段数据就会丢失，所以修改使用Post。

创建索引+字段类型



Get \_cat/命令，可以查看很多es信息，类似于 GET \_cat/health

修改：

POST /test3/\_doc/1/\_update

{

"doc":{

"name":"222"

}

}

查询 q是搜索的意思

GET /test3/\_doc/\_search?q=name:222

复杂查询:

GET test3/\_doc/\_search

{

"query":{

"match":{

"name":"222"

}

},

"\_source":["name"] //只查询这个字段

, "sort":[

{

"age":{

"order":"desc"

}

}

],

"from":0, //分页，从第几条开始

"size":10/ //分页，每页显示多少条 类似limit

"highlight":{ //自定义高亮 把match.name 查询的关键字高亮

"pre\_tags": "<p class=’key’ style='color:red'>", //查询是高亮的自定义标签

"post\_tags": "</p>",

"fields": {

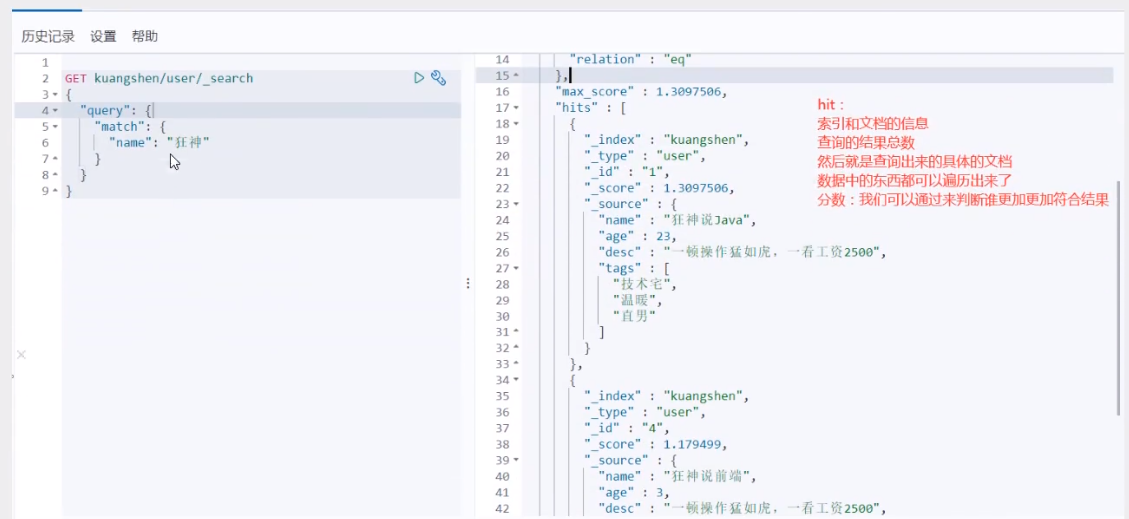
"name": {

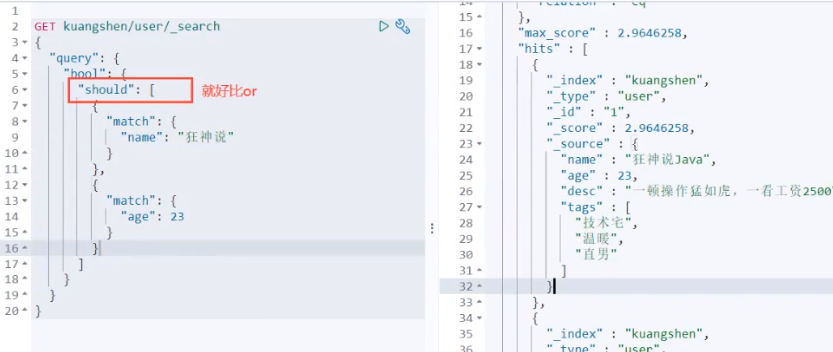
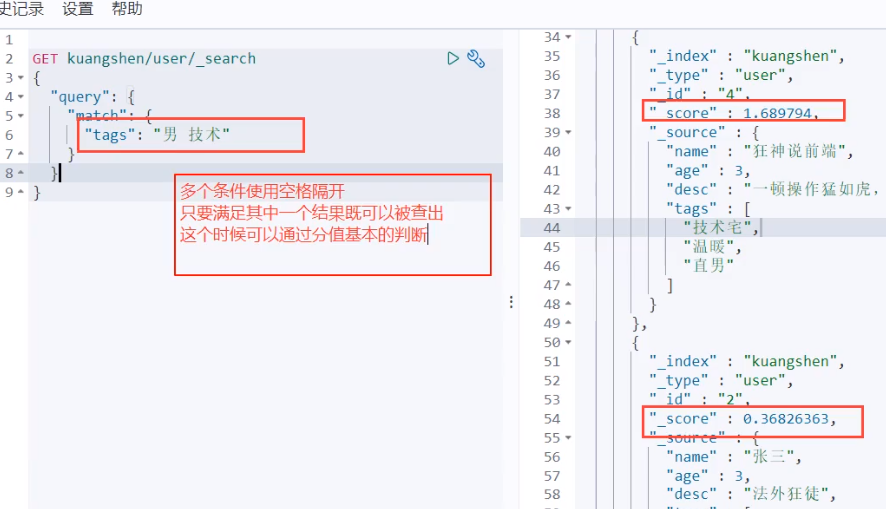
}

}

}

}



# 使用

1. 搜索Elasticsearch.Net，但是安装NEST 包
2. 实现类

//索引是否存在

var isexists = elasticSearchHelper.IsExistsByElasticClientIndex("zhaoxi");

//添加文档

user user = new user();

user.Account = "111";

user.Age = 19;

user.Name = "z2";

var re = await elasticSearchExtend.InsertEntityAsync(user);

//查找单个

var re = await elasticSearchExtend.FindOne("5mix73IBv-C3Ucz42PR-");

//根据条件查找 两种方法都行

//.Query(q => q.Match(m => m.Field(f => f.Name).Query("其他笔记本6"))) //单字段全文关键字检索 只要Name中包含值即可，且自带分词

//.Query(q => q.MultiMatch(m => m.Fields(fd=>fd.Fields(f=>f.Name,f=>f.OtherInfo)).Query("1神23456789"))) //多字段全文关键字检索 Name或OtherInfo包含该值即可，且自带分词

//.Analyzer("") // 该分词方法可不需要，因为上面的查询自带分词

//.Query(q => q.Bool(b=>b.Must(m=>m.Term(p=>p.Field(f=>f.Id).Value(4))))) //条件必须符合，无分词，有一些数据类型可能查询失败

//.Query(q => q.Range(c => c.Field(f => f.Id).LessThanOrEquals(5).GreaterThanOrEquals(3))) //范围查询

//.Sort(t => t.Ascending(p=>p.Id)) //id升序

//.From(0) //分页 第几条开始展示

//.Size(3) //分页，每页显示多少条

//多个条件一起搜索，例子如下

//var matchQuery = new List<Func<QueryContainerDescriptor<Computer>, QueryContainer>>

//{

// must => must.Bool(b => b.Must(m => m.Term(p => p.Field(f => f.Id).Value(5)),

// m => m.Term(p => p.Field(f => f.Name).Value("神州笔记本1"))

// )

// ),

// range => range.Range(c => c.Field(p => p.Id).LessThanOrEquals(5).GreaterThanOrEquals(3))

//};

//var tr = es.Search<Computer>(x=>x.Index("realyuseit").Query(q=>q.Bool(b=>b.Must(matchQuery))))

//1.

//SearchDescriptor<user> searchDescriptor = new SearchDescriptor<user>();

//searchDescriptor.From(0);

//searchDescriptor.Size(10);

//searchDescriptor.Query(q => q.Match(m => m.Field(f => f.Name).Query("mm")));

//var list = await elasticSearchExtend.FindWhere(searchDescriptor);

//2.

var searchRequest = new SearchRequest<user>(Nest.Indices.All)

{

From = 0,

Size = 10,

Query = new MatchQuery

{

Field = Infer.Field<user>(f => f.Name),

Query = "mm"

}

};

var list = await elasticSearchExtend.FindWhere(searchRequest);

1. 帮助类

namespace Zhaoxi.Helper

{

public class ElasticSearchHelper

{

private string Conn { get { return HcCrm.Util.Config.GetAppsettingsValue("ConnectionString", "Elasticsearch"); } }

private string DBName { get { return HcCrm.Util.Config.GetAppsettingsValue("Database", "Elasticsearch"); } }

public ElasticClient client;

public ElasticSearchHelper()

{

//创建客户端

var uris = new[] { new Uri(Conn) };

var connectionPool = new SniffingConnectionPool(uris);

var settings = new ConnectionSettings(connectionPool).DefaultIndex(DBName);

client = new ElasticClient(settings);

}

public ElasticSearchHelper(string conn)

{

//创建客户端

var uris = new[] { new Uri(conn) };

var connectionPool = new SniffingConnectionPool(uris);

var settings = new ConnectionSettings(connectionPool);

client = new ElasticClient(settings);

}

public ElasticSearchHelper(string conn, string dbname)

{

//创建客户端

var uris = new[] { new Uri(conn) };

var connectionPool = new SniffingConnectionPool(uris);

var settings = new ConnectionSettings(connectionPool).DefaultIndex(dbname);

client = new ElasticClient(settings);

}

#region 索引操作

/// <summary>

///索引是否存在

/// </summary>

/// <returns></returns>

public bool IsExistsByElasticClientIndex()

{

var existsResponse = client.Indices.Exists(DBName);

var isexists = existsResponse.Exists;

return isexists;

}

/// <summary>

///索引是否存在

/// </summary>

/// <returns></returns>

public bool IsExistsByElasticClientIndex(string dbname)

{

var existsResponse = client.Indices.Exists(dbname);

var isexists = existsResponse.Exists;

return isexists;

}

/// <summary>

/// 创建索引

/// </summary>

/// <returns></returns>

public bool CreateElasticClientIndex()

{

CreateIndexResponse createIndexResponse = client.Indices.Create(DBName);

var iscreate = createIndexResponse.Acknowledged;

return iscreate;

}

/// <summary>

/// 创建索引

/// </summary>

/// <returns></returns>

public bool CreateElasticClientIndex(string dbname)

{

CreateIndexResponse createIndexResponse = client.Indices.Create(dbname);

var iscreate = createIndexResponse.Acknowledged;

return iscreate;

}

/// <summary>

/// 删除索引

/// </summary>

/// <returns></returns>

public bool DeleteElasticClientIndex()

{

DeleteIndexResponse deleteIndexResponse = client.Indices.Delete(DBName);

var isdelete = deleteIndexResponse.Acknowledged;

return isdelete;

}

/// <summary>

/// 删除索引

/// </summary>

/// <returns></returns>

public bool DeleteElasticClientIndex(string dbname)

{

DeleteIndexResponse deleteIndexResponse = client.Indices.Delete(dbname);

var isdelete = deleteIndexResponse.Acknowledged;

return isdelete;

}

#endregion

}

public class ElasticSearchExtendHelper<T> where T : class, new()

{

public ElasticClient client

{

get

{

return new ElasticSearchHelper("http://localhost:9200", "zhaoxi").client;

}

}

/// <summary>

/// 查找全部

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public async Task<List<T>> FindAll()

{

var list = await client.SearchAsync<T>();

return list.Documents.ToList();

}

/// <summary>

/// 根据条件搜索查找全部

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public async Task<List<T>> FindWhere(SearchDescriptor<T> searchwhere)

{

var list = await client.SearchAsync<T>(searchwhere);

return list.Documents.ToList();

}

/// <summary>

/// 根据条件搜索查找全部

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public async Task<List<T>> FindWhere(SearchRequest<T> searchwhere)

{

var list = await client.SearchAsync<T>(searchwhere);

return list.Documents.ToList();

}

/// <summary>

/// 查找一个

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public async Task<T> FindOne(string id)

{

var one = await client.GetAsync<T>(id);

return one.Source;

}

/// <summary>

/// 插入

/// </summary>

/// <param name="t"></param>

/// <returns></returns>

public async Task<bool> InsertManyEntityAsync(List<T> t)

{

try

{

var task = await client.IndexManyAsync(t);

return task.IsValid;

}

catch (Exception ex)

{

return false;

}

}

/// <summary>

/// 插入

/// </summary>

/// <param name="t"></param>

/// <returns></returns>

public async Task<bool> InsertEntityAsync(T t)

{

try

{

var task = await client.IndexDocumentAsync(t);

return task.IsValid;

}

catch (Exception ex)

{

return false;

}

}

/// <summary>

/// 更新

/// </summary>

/// <param name="t"></param>

/// <returns></returns>

public async Task<bool> UpdateEntityAsync(string id, T t)

{

try

{

var task = await client.UpdateAsync<T>(id, x => x.Doc(t));

return task.IsValid;

}

catch (Exception ex)

{

return false;

}

}

/// <summary>

///删除

/// </summary>

/// <param name="t"></param>

/// <returns></returns>

public async Task<bool> DeleteEntityAsync(string id)

{

try

{

var task = await client.DeleteAsync<T>(id);

return task.IsValid;

}

catch (Exception ex)

{

return false;

}

}

}

}