ES开发手册

**文档修订摘要**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **日期** | **版本号** | **修订章节** | **描述** | **作者** | **评审者** | **评审日期** |
| 2017-7-17 | V0.1 |  | 初稿 | 何丹丹 |  |  |
| 2017-9-5 | V0.2 |  |  | 何丹丹 |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

# 重要概念理解

ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。

**索引(Index)**

ES将数据存储于一个或多个索引中，索引是具有类似特性的文档的集合。类比传统的关系型数据库来说，索引相当于SQL中的一个数据库。索引由其名称（必须为全小写字符）进行标识，并通过引用此名称完成文档的创建、搜索、更新及删除。

**类型(Type)**

类型是索引内部的逻辑分区，其意义完全取决于用户需求。因此，一个索引内部可定义一个或多个类型(type).

例如，在索引中，可以定义一个用户存储用户数据的类型，一个存储日志数据的类型，以及一个存储评论数据的类型。类比传统关系型数据库来说，类型相当于”表”。

**文档(Document)**

文档是Lucene索引和搜索的原子单位，它是包含了一个或多个域的容器，基于JSON格式进行表示。

文档由一个或多个域组成，每个域拥有一个名字及一个或多个值，有多个值的域通常称为“多值域”。每个文档可以存储不同的域集。

**映射(Mapping)**

ES中，所有的文档在存储之前都要首先进行分析。Mapping非常类似于静态语言中的数据类型。同语言的数据类型相比，mapping还有一些其他含义，mapping不仅告诉es一个field中是什么类型的值，它还告诉ES如何索引数据以及数据能否被搜索到。Mapping的作用就是约束。

# 2、客户端配置

ES集群环境统一配置在common-tools中作为默认的ES环境，客户端可以获取默认的ES服务，也可以定制自己的ES环境。

### 引入Jar包

若是gradle编译，添加以下依赖：

**注：com.google.guava:guava版本必须是19.0， 否则调用会失败**

compile group: **'org.elasticsearch'**, name: **'elasticsearch'**, version:**'5.4.3'**

**compile 'org.elasticsearch.client:transport:5.4.3'**compile **"com.aii.ipaas.search:ipaas-ses:** **1.0.0-SNAPSHOT"**

compile **"com.ai:ipaas-ccs:0.3"**compile **"com.aii.ipaas:ipaas-common-tool:** **1.0.0-SNAPSHOT"**

compile '**org.apache.logging.log4j:log4j-slf4j-impl:2.7**'

compile **'org.apache.logging.log4j:log4j-api:2.7**'

compile '**org.apache.logging.log4j:log4j-core:2.7**'

**compile 'com.google.guava:guava:19.0'**

compile 'org.apache.curator:curator-framework:2.9.0'

若是maven编译，添加以下依赖：

**注：com.google.guava:guava版本必须是19.0， 否则调用会失败**

<**dependency**>  
 <**groupId**>org.elasticsearch</**groupId**>  
 <**artifactId**>elasticsearch</**artifactId**>  
 <**version**>5.4.3</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**>org.elasticsearch.client</**groupId**>  
 <**artifactId**>transport</**artifactId**>  
 <**version**>5.4.3</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**>com.aii.ipaas </**groupId**>  
 <**artifactId**>ipaas-ses</**artifactId**>  
 <**version**>1.0.0-SNAPSHOT</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**>com.aii.ipaas</**groupId**>  
 <**artifactId**>ipaas-common-tool</**artifactId**>  
 <**version**>1.0.0-SNAPSHOT</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**>**com.ai**</**groupId**>  
 <**artifactId**>**ipaas-ccs**</**artifactId**>  
 <**version**>**0.3**</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**> **org.apache.logging.log4j** </**groupId**>  
 <**artifactId**> **log4j-slf4j-impl** </**artifactId**>  
 <**version**>**2.7**</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**> **org.apache.logging.log4j** </**groupId**>  
 <**artifactId**> **log4j-api** </**artifactId**>  
 <**version**>**2.7**</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**> **org.apache.logging.log4j** </**groupId**>  
 <**artifactId**> **log4j-core** </**artifactId**>  
 <**version**>**2.7**</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**> **com.google.guava** </**groupId**>  
 <**artifactId**> **guava** </**artifactId**>  
 <**version**>**19.0**</**version**>  
</**dependency**>

### 配置ES服务地址

ES服务地址配置在文件ses.properties中，ses.properties配置项如下：

**ccs.appname=crm-so-ins**

**ccs.zk\_address=10.1.239.45:42181**

**ses.appname=crm-so**

**ses.address=10.1.239.44:9300**

**ses.mode=SDK**

说明：

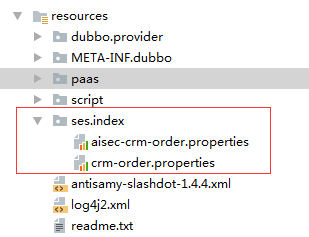
|  |  |  |
| --- | --- | --- |
| 字段 | 描述 | 是否必填 |
| ccs.appname | ccs app name | 是 |
| ccs.zk\_address | ccs 缓存地址 | 是 |
| ses.appname | ses域名称 | 是 |
| ses.address | es 环境地址 | 是 |
| ses.mode | 服务模式，统一写成SDK | 是 |
| ses.isAuth | es鉴权开关，值为’Y’时表示开启鉴权，其他值表示不开启鉴权 | 否，保留项 |
| ses.user | 连接es用户名密码，只在isAuth=‘Y’时有效，格式: username:password | 否，保留项 |
| ses.clusterName | es集群名称，只在isAuth=‘Y’时有效 | 否，保留项 |

将ses.properties文件上传到自动配置中心进行管理。

### 索引文件配置

各个域管理自己的索引文件，将索引文件存放在项目的resources/ses/index目录下。

此处要求索引文件名必须与各个域的域名称相对应，以appname来命名索引文件。



以crm-order.properties为例：

**/com/aii/ipaas/sdkmode-paas-ses-elasticsearch-mapped= { "SES001": { "indexname": "crm-order-inx", "mappingid": "orderId", "mapping": { "ses-test-inx":{"properties": {"orderId":{"type":"integer"}, "productInfo":{"properties": {"offerId":{"type":"integer"}, "offerName":{"type":"text" }}}, "custInfo":{"properties": {"custId":{"type":"integer" }, "custName":{ "type":"text" } }}, "date":{"type": "date"}}}},"shards":"1","replicas":"1"}}**

**/com/aii/ipaas/paas-sesns-ses-mapped= {"****crm-order-index": "SES001"}**

**/com/aii/ipaas/sdkmode-paas-ses-elasticsearch-mapped**

**/com/aii/ipaas/paas-sesns-ses-mapped**

**以上两项为固定Key值，各个域的索引配置文件中需要包含这两项，且名称不可更改。**

**/com/aii/ipaas/paas-sesns-ses-mapped中配置索引名称的映射,以上图为例：”crm-order-index”表示crm-order域中使用的索引名称，”SES001”表示映射名称，一个映射名称对应一个es服务器中实际的索引。**

**/com/aii/ipaas/sdkmode-paas-ses-elasticsearch-mapped中配置索引的各项属性，属性包括：**

**indexname -– es中索引名称**

**mappingid –- es索引的mapping的主键，即\_id**

**mapping –- es索引的mapping**

**shards -- es索引分片数，分片可以将一个索引内部的数据分布地存储于多个节点。**

**replicas – es索引副本数，index的冗余备份，可以用于防止数据丢失。**

Mapping创建可以使用kibana环境：

环境地址：<http://10.1.239.44:5601/app/kibana#/dev_tools/console?_g=()>



将创建的内容拷贝到各个index的创建文件(如crm-order.properties)中。

### 刷新索引缓存

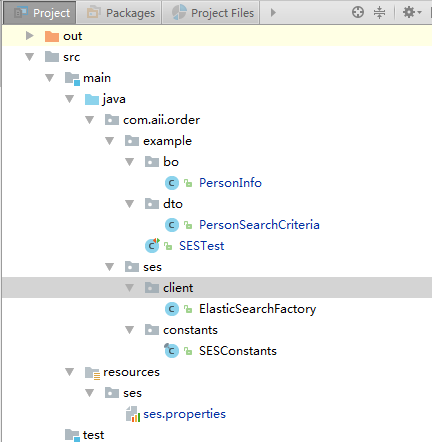
新建索引文件或者修改了已有的索引文件之后，需要刷新索引缓存。

运行ipaas-common-tool.jar包中com.aii.ipaas.common.ses.SESServiceStart类的main方法，刷新索引缓存。

# 3、索引操作

以下只介绍了索引应用中的一部分，更多的操作请参考es\_api.chm文档

以ESTest项目为例讲述下具体如何调用：



**com.aii.order.ses 该目录下存放调用ES服务的服务类**

**com.aii.order.ses.client.ElasticSearchFactory 域封装的获取ES客户端的工厂类**

**com.aii.constants.SESConsants 存放该域下所有的索引名称常量类**

**com.aii.order.example.bo.PersonInfo 需要索引的对象**

**com.aii.order.example.dto.PersonSearchCriteria 索引的查询条件**

## 3.1 获取客户端

ES客户端：**com.aii.ipaas.search.ISearchClient**

获取ES客户端的工厂类：**com.aii.ipaas.common. ses.SESClientFactory**

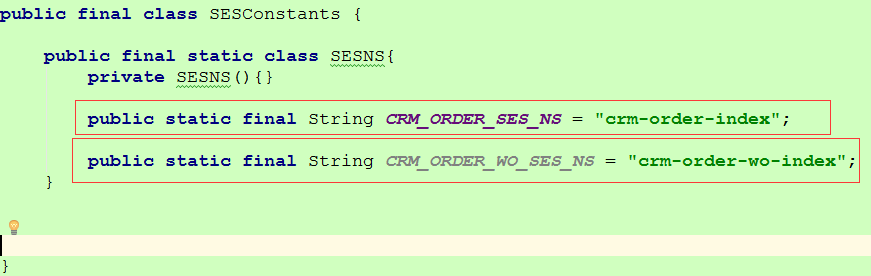
**获取es服务的方法：**

**public static ISearchClient getSearchClient(String sesns)**

sesns --- 索引映射域名称

示例：

ISearchClient custClient = SESClientFactory.*getSearchClient*(SESConstants.SESNS.***CRM\_ORDER\_CUST\_SES\_NS***);



**“crm-order-index”,”crm-order-wo-index”表示不同的sesns.对应crm-order.properties中的配置。**

## 3.2 新建索引

### 插入单条泛型数据

方法：<T> boolean insert(T data)

注：如果该索引中设置的\_id为“personID”，那么对象data中personID必填。

示例代码：

**public void** testInsertObj(){  
 ServiceInfo serviceInfo = **new** ServiceInfo();  
 serviceInfo.setId(6);  
 serviceInfo.setServiceName(**"doSave"**);  
 serviceInfo.setDate(**new** Date());  
 serviceInfo.setDuration(123243);  
 **searchClient**.insert(serviceInfo);  
}

### 插入单条索引数据

方法： boolean insert(Map<String, Object> data)

示例：

**public void** testInsertMap(){  
 Map<String,Object> map = **new** HashedMap();  
 map.put(**"id"**,7);  
 map.put(**"serviceName"**,**"doSave"**);  
 map.put(**"date"**,**new** Date());  
 map.put(**"duration"**,15000);  
 **searchClient**.insert(map);  
}

### 插入单条JSON数据

方法： boolean insert(String json)

示例：

**public void** testInsertJson(){  
 String json = **"{"**+**"\"id\":"**+8+  
 **",\"serviceName\":\"doSave\""**+  
 **",\"date\":\""**+**"2017-10-24"**+**"\""**+  
 **",\"duration\":"**+10000+**"}"**;  
 **searchClient**.insert(json);  
}

### 插入多条泛型数据

方法：<T> boolean bulkInsert(List<T> datas)

如果该索引中设置的\_id为“personID”

示例：

**public void** testBulkInsert(){  
 List<ServiceInfo> serviceInfos = **new** ArrayList<>();  
 ServiceInfo serviceInfo1 = **new** ServiceInfo();  
 serviceInfo1.setId(9);  
 serviceInfo1.setServiceName(**"doUpdate"**);  
 serviceInfo1.setDate(**new** Date());  
 serviceInfo1.setDuration(123243);  
  
 ServiceInfo serviceInfo2 = **new** ServiceInfo();  
 serviceInfo2.setId(10);  
 serviceInfo2.setServiceName(**"doUpdate"**);  
 serviceInfo2.setDate(**new** Date());  
 serviceInfo2.setDuration(123240);  
  
 serviceInfos.add(serviceInfo1);  
 serviceInfos.add(serviceInfo2);  
 **searchClient**.bulkInsert(serviceInfos);  
}

### 插入多条数据，格式为Map

方法：boolean bulkMapInsert(List<Map<String, Object>> datas)

示例：

**public void** testBulkInsertMap(){  
 Map<String,Object> json1 = **new** HashMap();  
 json1.put(**"id"**,1);  
 json1.put(**"serviceName"**,**"action1"**);  
 json1.put(**"date"**,**"2017-10-20"**);  
 json1.put(**"duration"**,12000);  
  
 Map<String,Object> json2 = **new** HashMap<>();  
 json2.put(**"id"**,2);  
 json2.put(**"serviceName"**,**"action2"**);  
 json2.put(**"date"**,**"2017-10-19"**);  
 json2.put(**"duration"**,13000);  
  
 Map<String,Object> json3 = **new** HashMap<>();  
 json3.put(**"id"**,3);  
 json3.put(**"serviceName"**,**"action3"**);  
 json3.put(**"date"**,**"2017-10-20"**);  
 json3.put(**"duration"**,13000);  
  
 Map<String,Object> json4 = **new** HashMap<>();  
 json4.put(**"id"**,4);  
 json4.put(**"serviceName"**,**"action4"**);  
 json4.put(**"date"**,**"2017-10-19"**);  
 json4.put(**"duration"**,15000);  
  
 Map<String,Object> json5 = **new** HashMap<>();  
 json5.put(**"id"**,5);  
 json5.put(**"serviceName"**,**"action5"**);  
 json5.put(**"date"**,**"2017-10-18"**);  
 json4.put(**"duration"**,15000);  
  
 List<Map<String,Object>> datas = **new** ArrayList();  
 datas.add(json1);  
 datas.add(json2);  
 datas.add(json3);  
 datas.add(json4);  
 datas.add(json5);  
 **searchClient**.bulkMapInsert(datas);  
}

### 插入多条数据，格式为JSON

方法：boolean bulkJsonInsert(List<String> jsons)

示例：

**public void** testBulkInsertJson(){  
 List<String> jsons = **new** ArrayList<>();  
 String json1 = **"{"**+**"\"id\":"**+11+  
 **",\"serviceName\":\"doDelete\""**+  
 **",\"date\":\""**+**"2017-10-24"**+**"\""**+  
 **",\"duration\":"**+10030+**"}"**;  
 String json2 = **"{"**+**"\"id\":"**+12+  
 **",\"serviceName\":\"doDelete\""**+  
 **",\"date\":\""**+**"2017-10-25"**+**"\""**+  
 **",\"duration\":"**+10030+**"}"**;  
 jsons.add(json1);  
 jsons.add(json2);  
 **searchClient**.bulkJsonInsert(jsons);  
}

## 3.3 删除索引

### 根据索引标识删除

方法：boolean delete(String id)

Id表示索引\_id

代码：

**public void** testDelete(){  
 **int** id = 14;  
 **searchClient**.delete(id+**""**);  
}

### 根据查询条件删除多条

方法：boolean delete(List<SearchCriteria> searchCriteria)

示例：

**public void** testDeleteByQry(){  
 List<SearchCriteria> scs = **new** ArrayList<>();  
 SearchCriteria sc = **new** SearchCriteria();  
 SearchOption searchOption = **new** SearchOption();  
 searchOption.setSearchType(SearchOption.SearchType.***term***);  
 sc.setOption(searchOption);  
 sc.setField(**"id"**);  
 sc.addFieldValue(**"13"**);  
 scs.add(sc);  
 **searchClient**.delete(scs);  
}

### 批量删除

方法：boolean bulkDelete(List<String> ids)

示例：

**public void** testBulkDelete(){  
 List<String> ids = **new** ArrayList<>();  
 ids.add(**"9"**);  
 ids.add(**"10"**);  
 **searchClient**.bulkDelete(ids);  
}

## 3.4 更新索引（合并模式）

合并模式表示会将新的文档与已存在的进行信息合并，不支持更新一个不存在的文档。

### 更新单个文档，泛型

方法：<T> boolean update(String id, T data)

id表示索引的\_id值

该方法如果更新一个不存在的索引会报错。

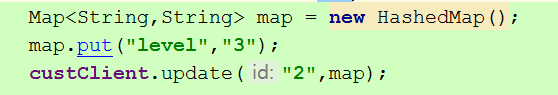
示例：

**public void** testUpdate(){  
 List<SearchCriteria> conditions = **new** ArrayList<>();  
 SearchCriteria sc = **new** SearchCriteria();  
 SearchOption searchOption = **new** SearchOption();  
 searchOption.setSearchType(SearchOption.SearchType.***term***);  
 sc.setField(**"id"**);  
 sc.addFieldValue(**"7"**);  
 sc.setOption(searchOption);  
 conditions.add(sc);  
 Result<ServiceInfo> result = **searchClient**.search(conditions,0,10,**null**,ServiceInfo.**class**);  
 **if** (**null** != result){  
 List<ServiceInfo> list = result.getContents();  
 **for**(ServiceInfo serviceInfo:list){  
 serviceInfo.setDuration(14000);  
 **searchClient**.update(serviceInfo.getId()+**""**,serviceInfo);  
 }  
 }  
}

### 更新单个文档，Map形式

方法：boolean update(String id, Map<String, Object> data)

示例：



### 更新单个文档，JSON形式

方法：boolean update(String id, String json)

示例：



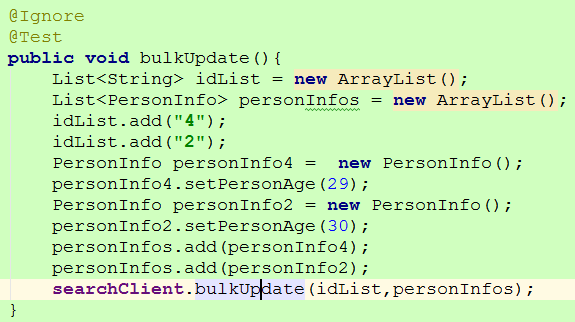
### 更新多个文档，泛型

方法：<T> boolean bulkUpdate(List<String> ids, List<T> datas)

ids中存放索引的\_id值

该方法如果更新一个不存在的索引会报错。如果有一个错误，则失败。

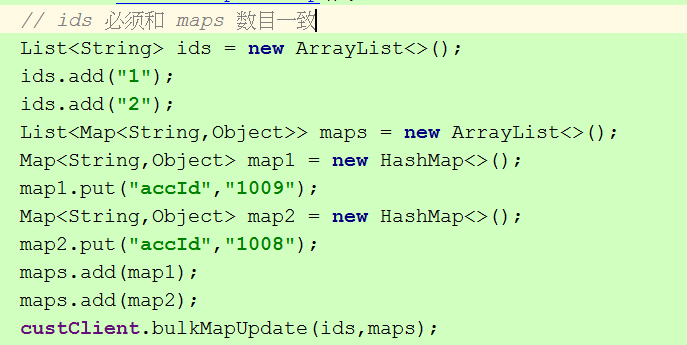
代码：



### 更新多个文档，Map形式

方法：boolean bulkMapUpdate(List<String> ids, List<Map<String, Object>> datas)

示例：

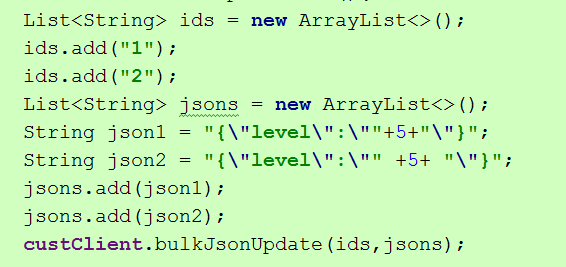


### 更新多个文档，JSON形式

方法：boolean bulkJsonUpdate(List<String> ids, List<String> jsons)

Ids 与jsons 的数据要一一对应

示例：



## 3.5 更新索引(新增模式)

新增模式就是支持更新一个不存在的文档

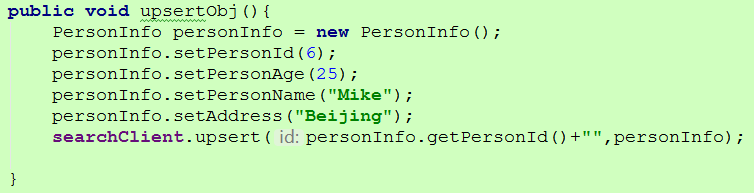
### 更新单个文档，泛型

方法：<T> boolean upsert(String id, T data)

如果更新一个不存在的索引，会新建该索引

id表示索引的\_id值

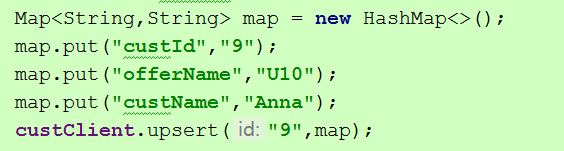
代码：



### 更新单个文档，Map形式

方法：boolean upsert(String id, Map<String, Object> data)

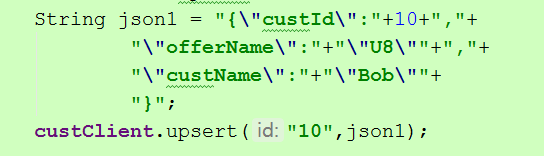
示例：



### 更新单个文档，JSON形式

方法：boolean upsert(String id, String json)

示例：



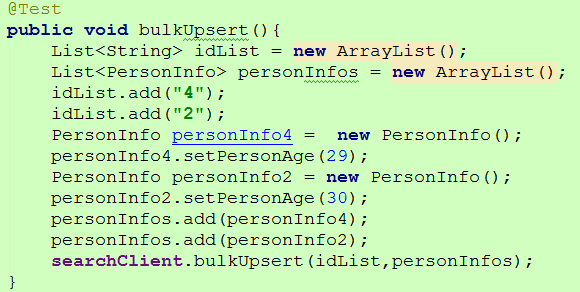
### 更新多个文档，泛型

方法：<T> boolean bulkUpsert(List<String> ids, List<T> datas)

ids中存放索引的\_id值

upsert()方法，如果更新一个不存在的索引，会新建该索引。如果有一个错误，则失败

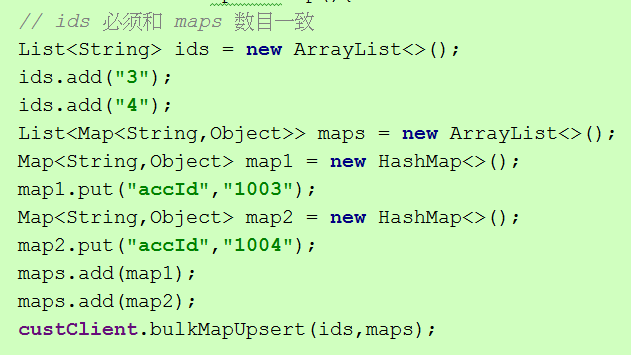
示例：



### 更新多个文档，Map形式

方法：boolean bulkMapUpsert(List<String> ids, List<Map<String, Object>> datas)

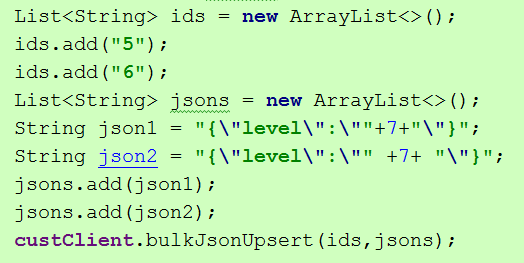
示例：



### 更新多个文档，JSON形式

方法：boolean bulkJsonUpsert(List<String> ids, List<String> jsons)

示例：



## 3.6 查询索引

### SearchCriteria查询，返回泛型

返回全部字段：

方法：<T> Result<T> search(List<SearchCriteria> searchCriterias, int from, int offset, @Nullable List<Sort> sorts,

Class<T> clazz);

返回指定字段，将需要返回的字段放入resultFields中：

方法：String search(List<SearchCriteria> searchCriterias, int from, int offset, @Nullable List<Sort> sorts, String[] resultFields)

#### 查询相等（分词）

代码：

**public void** testSearchCriteriaEqual(){  
 *// search serviceName = 'doSave'* List<SearchCriteria> conditions = **new** ArrayList<>();  
 SearchCriteria sc = **new** SearchCriteria();  
 SearchOption searchOption = **new** SearchOption();  
 searchOption.setSearchType(SearchOption.SearchType.***match***);  
 sc.setOption(searchOption);  
 sc.setField(**"serviceName"**);  
 sc.addFieldValue(**"doSave"**);  
 conditions.add(sc);  
 Result<ServiceInfo> result = **searchClient**.search(conditions,0,10,**null**,ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

#### 查询区间

代码：

**public void** testSearchCriteriaRange(){  
 *// 15000>=duration>=10000* List<SearchCriteria> conditions = **new** ArrayList<>();  
 SearchCriteria sc = **new** SearchCriteria();  
 sc.setOption(**new** SearchOption(SearchOption.SearchLogic.***must***, SearchOption.SearchType.***range***));  
 sc.setField(**"duration"**);  
 sc.addFieldValue(**"10000"**);  
 sc.addFieldValue(**"15000"**);  
 Result<ServiceInfo> result = **searchClient**.search(conditions,0,10,**null**,ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

#### 查询条件之间and关系

代码：

**public void** testSearchCriteriaAnd(){  
 List<SearchCriteria> conditions = **new** ArrayList<>();  
 SearchCriteria sc1 = **new** SearchCriteria();  
 sc1.setOption(**new** SearchOption(SearchOption.SearchLogic.***must***,SearchOption.SearchType.***match***));  
 sc1.setField(**"serviceName"**);  
 sc1.addFieldValue(**"doSave"**);  
 SearchCriteria sc2 = **new** SearchCriteria();  
 sc2.setOption(**new** SearchOption(SearchOption.SearchLogic.***must***,SearchOption.SearchType.***range***));  
 sc2.setField(**"duration"**);  
 sc2.addFieldValue(**"10000"**);  
 sc2.addFieldValue(**"15000"**);  
 conditions.add(sc1);  
 conditions.add(sc2);  
 Result<ServiceInfo> result = **searchClient**.search(conditions,0,10,**null**,ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

#### 查询条件之间or关系

代码：

**public void** testSearchCriteriaOr(){  
 List<SearchCriteria> conditions = **new** ArrayList<>();  
 SearchCriteria sc1 = **new** SearchCriteria();  
 sc1.setOption(**new** SearchOption(SearchOption.SearchLogic.***should***,SearchOption.SearchType.***match***));  
 sc1.setField(**"servicename"**);  
 sc1.addFieldValue(**"doSave"**);  
 SearchCriteria sc2 = **new** SearchCriteria();  
 sc2.setOption(**new** SearchOption(SearchOption.SearchLogic.***should***,SearchOption.SearchType.***range***));  
 sc2.setField(**"duration"**);  
 sc2.addFieldValue(**"10000"**);  
 sc2.addFieldValue(**"15000"**);  
 conditions.add(sc1);  
 conditions.add(sc2);  
 Result<ServiceInfo> result = **searchClient**.search(conditions,0,10,**null**,ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
  
}

#### 查询not关系

代码：

**public void** testSearchCriteriaNot(){  
 List<SearchCriteria> conditions = **new** ArrayList<>();  
 SearchCriteria sc = **new** SearchCriteria();  
 sc.setOption(**new** SearchOption(SearchOption.SearchLogic.***must\_not***,SearchOption.SearchType.***match***));  
 sc.setField(**"serviceName"**);  
 sc.addFieldValue(**"doSave"**);  
 conditions.add(sc);  
 Result<ServiceInfo> result = **searchClient**.search(conditions,0,10,**null**,ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

### DSL查询，返回泛型

返回全部字段：

方法：<T> Result<T> searchByDSL(String dslJson, int from, int offset, @Nullable List<Sort> sorts, Class<T> clazz)

指定返回字段，将需要返回的字段放入resultFields中：

方法：<T> Result<T> searchByDSL(String dslJson, int from, int offset, @Nullable List<Sort> sorts, Class<T> clazz, String[] resultFields);

#### 查询相等（分词）

示例：

**public void** testSearchDSLEqual(){  
 String serviceName = **"doSave"**;  
 String dslJson = **"{\"match\":{\"serviceName\":\""**+serviceName+**"\"}}"**;  
 Result<ServiceInfo> result = **searchClient**.searchByDSL(dslJson,0,10,**null**, ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

#### 查询区间

示例：

**public void** testSearchDSLRange(){  
 **long** durationMin = 10000;  
 **long** durationMax = 15000;  
 String dslJson = **"{\"range\":{\"duration\":{\"gte\":"**+durationMin+**",\"lte\":"**+durationMax+**"}}}"**;  
 Result<ServiceInfo> result = **searchClient**.searchByDSL(dslJson,0,10,**null**, ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
  
}

#### 查询条件之间and关系

示例：

**public void** testSearchDSLAnd(){  
 String serviceName = **"doSave"**;  
 **int** id = 6;  
 String dslJson = **"{\"bool\":{\"must\":[{\"match\":{\"serviceName\":\""**+serviceName+**"\"}},{\"match\":{\"id\":"**+id+**"}}]}}"**;  
 Result<ServiceInfo> result = **searchClient**.searchByDSL(dslJson,0,10,**null**, ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

#### 查询条件之间or关系

示例：

**public void** testSearchDSLOr(){  
 String serviceName1 = **"doSave"**;  
 String serviceName2 = **"doDelete"**;  
 String dslJson = **"{\"bool\":{\"should\":[{\"match\":{\"serviceName\":\""**+serviceName1+**"\"}},{\"match\":{\"serviceName\":\""**+serviceName2+**"\"}}]}}"**;  
 Result<ServiceInfo> result = **searchClient**.searchByDSL(dslJson,0,10,**null**, ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

#### 查询not关系

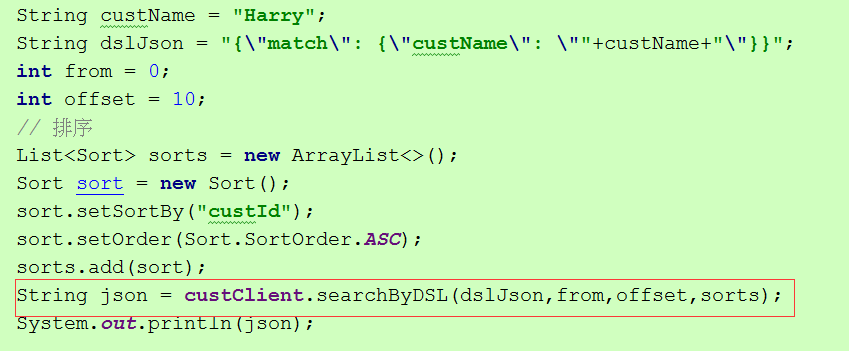
**public void** testSearchDSLNot(){  
 String serviceName = **"doSave"**;  
 String dslJson = **"{\"bool\":{\"must\_not\":{\"match\":{\"serviceName\":\""**+serviceName+**"\"}}}}"**;  
 Result<ServiceInfo> result = **searchClient**.searchByDSL(dslJson,0,10,**null**, ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

### DSL查询，返回JSON

返回全部字段：

方法：String searchByDSL(String dslJson, int from, int offset, @Nullable List<Sort> sorts)

示例：



指定返回字段，将需要返回的字段放入resultFields中：

方法：String searchByDSL(String dslJson, int from, int offset, @Nullable List<Sort> sorts, String[] resultFields);

### SQL查询，返回泛型

返回全部字段：

方法：<T> Result<T> searchBySQL(String querySQL, int from, int offset, @Nullable List<Sort> sorts, Class<T> clazz)

指定返回结果，将需要返回的字段放入resultFields中：

方法：<T> Result<T> searchBySQL(String querySQL, int from, int offset, @Nullable List<Sort> sorts, Class<T> clazz,

String[] resultFields)

当关键字有多个时，只会匹配第一个关键字

示例：

**public void** testSearchSql(){  
 String keyword = **"doSave"**;  
 Result<ServiceInfo> result = **searchClient**.searchBySQL(keyword,0,10,**null**,ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

### SQL查询，返回JSON

返回全部字段：

方法：String searchBySQL(String querySQL, int from, int offset, @Nullable List<Sort> sorts)

指定返回结果，将需要返回的字段放入resultFields中：

方法：String searchBySQL(String querySQL, int from, int offset, @Nullable List<Sort> sorts, String[] resultFields)

### 全文检索，返回泛型

返回全部字段：

方法：<T> Result<T> fullTextSearch(String text, int from, int offset, @Nullable List<Sort> sorts, List<SearchCriteria> postFilters, Class<T> clazz)

注意：只能关键字分词查询，不支持模糊查询，比如搜索词组Kate Green,你可以搜分词Kate、Green都能匹配到Kate Green，但是如果你搜”K”,就匹配不到了。

参数说明：

|  |  |
| --- | --- |
| 参数 | 描述 |
| keyword | 查询关键字 |
| from | 起始位置 |
| offset | 偏移量 |
| sorts | 排序字段 |
| postFilters | 对结果过滤条件，如果不需要过滤，传null |
| clazz | 结果类型 |

关键字可以指定多个，以空格隔开

示例：

**public void** testFullTextSearch(){  
 String keyword = **"doSave"**;  
 *// 设置后置查询* List<SearchCriteria> postCondition = **new** ArrayList<>();  
 SearchCriteria sc = **new** SearchCriteria();  
 SearchOption searchOption = **new** SearchOption();  
 searchOption.setSearchType(SearchOption.SearchType.***range***);  
 sc.setOption(searchOption);  
 sc.setField(**"duration"**);  
 sc.addFieldValue(**"13000"**);  
 sc.addFieldValue(**"15000"**);  
 postCondition.add(sc);  
 Result<ServiceInfo> result = **searchClient**.fullTextSearch(keyword,0,10,**null**,postCondition,ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

### 全文检索支持聚合，返回泛型

支持字段聚合，返回对象中包含聚合的结果数量

方法：

<T> Result<T> fullTextSearch(String keyword, AggField aggField, **int** from, **int** offset,  
 @Nullable List<Sort> sorts,List<SearchCriteria> postFilters, Class<T> clazz);

如果指定对哪些字段进行全文索引，各字段是或者关系，使用以下方法：

<T> Result<T> fullTextSearch(String keyword, List<String> qryFields, AggField aggField, int from,  
 int offset, @Nullable List<Sort> sorts,List<SearchCriteria> postFilters, Class<T> clazz);

参数说明：

|  |  |
| --- | --- |
| 参数 | 描述 |
| keyword | 查询关键字 |
| qryFields | 指定字段全文索引 |
| aggField | 聚合字段 |
| from | 起始位置 |
| offset | 偏移量 |
| sorts | 排序字段 |
| postFilters | 对结果过滤条件，如果不需要过滤，传null |
| clazz | 结果类型 |

### 模糊检索，返回泛型

方法：

<T> Result<T> fuzzySearch(String keyword,String field, int from, int offset, List<Sort> sorts, List<SearchCriteria> postFilters, Class<T> clazz);

说明：支持按某个字段的模糊查询以及全文模糊查询

参数说明：

|  |  |
| --- | --- |
| 参数 | 描述 |
| keyword | 查询关键字，格式为“”关键字\*”，“关键字?”，  ？ 匹配任何字符，\*匹配零个或多个字符 |
| Field | 查询字段，为空时表示全文检索 |
| from | 起始位置 |
| offset | 偏移量 |
| sorts | 排序字段 |
| postFilters | 对结果过滤条件，如果不需要过滤，传null |
| clazz | 结果类型 |

示例：

**public void** testFuzzySearch(){  
 String keyword = **"action\*"**;  
 *// 对检索后的结果进行过滤* List<SearchCriteria> condition = **new** ArrayList<>();  
 SearchCriteria sc = **new** SearchCriteria();  
 sc.setField(**"duration"**);  
 SearchOption searchOption = **new** SearchOption();  
 searchOption.setSearchType(SearchOption.SearchType.***range***);  
 sc.setOption(searchOption);  
 sc.addFieldValue(**"14000"**);  
 sc.addFieldValue(**"15000"**);  
 condition.add(sc);  
 Result<ServiceInfo> result = **searchClient**.fuzzySearch(keyword,**null**,0,10,**null**,condition,ServiceInfo.**class**);  
 Assert.*assertNotNull*(result);  
}

## 3.7 聚合

方法：

通过SearchCriteria方式搜索：

Result<List<AggResult>> aggregate(List<SearchCriteria> searchCriterias, AggField field)

说明：支持对满足查询条件的数据，按照某个字段进行聚合，支持嵌套聚合

支持的分组聚合数量最大为1000

参数说明：

|  |  |
| --- | --- |
| 参数 | 描述 |
| searchCriterias | 查询条件 |
| field | 聚合字段 ,注意，聚合字段不能是text类型 |

聚合类型配置在枚举类：com.aii.ipaas.search.vo.AggType

支持的聚合包括：

COUNT（数量）

AVG（平均值）

STATS（统计）

SUM（总数）

MIN（最小值）

MAX（最大值）

PERCENTILES（百分比）

PERCENTILES\_RANK（百分比等级）

TERMS（项）

DATE\_HISTOGRAM（日期直方图）

RANGE（范围）

DATE\_RANGE（日期范围）

IPV4\_RANGE（IP地址范围）

SIGNIFICANT\_TERMS（有效项）

示例：

**public void** testAggregate(){  
 List<AggField> subFields = **new** ArrayList<>();  
 AggField aggField = **new** AggField();  
 aggField.setAggType(AggType.***TERMS***);  
 aggField.setField(**"serviceName"**);  
 AggField subField = **new** AggField();  
 subField.setAggType(AggType.***STATS***);  
 subField.setField(**"duration"**);  
 subFields.add(subField);  
 aggField.setSubAggs(subFields);  
 List<SearchCriteria> searchCriteriaList = **new** ArrayList<>();  
 SearchCriteria searchCriteria = **new** SearchCriteria();  
 searchCriteria.setField(**"date"**);  
 searchCriteria.setOption(**new** SearchOption(SearchOption.SearchLogic.***must***,SearchOption.SearchType.***range***));  
 searchCriteria.addFieldValue(**"2017-10-01"**);  
 searchCriteria.addFieldValue(**"2017-10-30"**);  
 searchCriteriaList.add(searchCriteria);  
 Result<List<AggResult>> result = **searchClient**.aggregate(searchCriteriaList,aggField);  
}

通过DSL方式搜索：

Result<List<AggResult>> aggregate(String dslJson, AggField field)

参数说明：

|  |  |
| --- | --- |
| 参数 | 描述 |
| searchCriterias | 查询条件 |
| field | 聚合字段 ,注意，聚合字段不能是text类型 |

示例：

**public void** testAggregateDSL(){  
 List<AggField> subFields = **new** ArrayList<>();  
 AggField aggField = **new** AggField();  
 aggField.setAggType(AggType.***TERMS***);  
 aggField.setField(**"serviceName"**);  
 AggField subField = **new** AggField();  
 subField.setAggType(AggType.***STATS***);  
 subField.setField(**"duration"**);  
 subFields.add(subField);  
 aggField.setSubAggs(subFields);  
 String startDate = **"2017-10-01"**;  
 String endDate = **"2017-10-30"**;  
 String dslJson = **"{\"range\":{\"date\":{\"gte\":\""**+startDate+**"\",\"lte\":\""**+endDate+**"\"}}}"**;  
 Result<List<AggResult>> result = **searchClient**.aggregate(dslJson,aggField);  
  
}

# 4、实时同步数据库

参数说明：

Input:

* jdbc\_driver\_library：  
  数据库驱动路径，这里我填写的是绝对路径，可自行尝试相对路径；
* jdbc\_driver\_class：  
  驱动名称；
* jdbc\_connection\_string：  
  数据库的连接字符串；  
  product为数据库名；  
  ?autoReconnect=true&useSSL=false自动重连并禁用SSL；
* jdbc\_user：  
  数据库用户名；
* jdbc\_password：  
  数据库密码；
* schedule：  
  重复执行导入任务的时间间隔；
* statement：  
  导入的表（查询SQL，可以过滤数据）
* statement\_filepath：  
  导入的表（查询SQL文件路径）
* jdbc\_paging\_enabled:

是否支持分页

* jdbc\_page\_size:

每页查询数目

* jdbc\_page\_size:

每页查询数目

* use\_column\_value:

更新时使用增量值而不是时间戳，类型为boolean，默认为false

* tracking\_column:

如果跟踪列值不是时间戳，则该列将被跟踪

* tracking\_column\_type:

跟踪列类型，类型为numeric、timestamp

* record\_last\_run

是否记录最后执行记录，类型为boolean

* clean\_run

是否清楚执行记录，类型为boolean

* last\_run\_metadata\_path

最后执行记录的保存地址，类型为string

output:

* index:  
  索引名称（类似数据库名称）；
* document\_type：  
  类型名称（类似数据库表名）；
* document\_id：  
  类似主键；
* hosts：  
  要导入到的Elasticsearch所在的主机；

更多参考：

https://www.elastic.co/guide/en/logstash/current/plugins-inputs-jdbc.html#plugins-inputs-jdbc-use\_column\_value

### 全量更新

1. 配置文件：



配置后将文件放置到logstash服务器安装目录的conf目录下：



2）启动同步

输入命令：./bin/logstash -f conf/logstash-mysql.conf



### 增量更新

可以根据时间戳或者某个为数值型的列作为更新的跟踪列。



注：document\_id后面的主键必须小写

### 获取查询客户端

ES客户端：**com.aii.ipaas.search.ISearchClient**

获取ES客户端的工厂类：**com.aii.ipaas.common. ses.SESClientFactory**

**获取es服务的方法：**

**public static** ISearchClient getSearchClient(String indexName, String type)

参数说明：

indexName --- 索引名称（相当于库名）

type --- 索引类型（相当于表名）

**注意：这个方法只适合获取ES中存在的索引，如果获取不存在的索引，会抛异常。**

### 对delete操作泼冷水

到目前为止，所有google,stackoverflow,elastic.co,github上面搜索的插件和实时同步的信息，告诉我们：目前同步delete还没有好的解决方案。

暂行方案：

首先，软件删除而非物理删除数据，新增一个 flag 列，标识记录是否已经被删除，这样，相同的记录也会存在于Elasticsearch。可以执行简单的term查询操作,检索出已经删除的数据信息。   
其次，若需要执行cleanup清理数据操作（物理删除），只需要在数据库和ES中同时删除掉标记位deleted的记录即可。如：mysql执行：delete from cc where cc.flag=’deleted’; ES同样执行对应删除操作。