三、 运单管理功能

1. 运单列表显示

waybill_manage.html 运单管理

```
<table id="tt" class="easyui-datagrid"
    url="../../waybill_pageQuery.action" fit="true"
    <thead>
```

条件查询

字段模糊查询，涉及大数据量性能问题 ？

Java 解决 大数据量 字段模糊查询，解决方案： 建立数据索引库，全文检索方式查询

全文检索： 检索文本中每个词 与 搜索项进行比对

全文索引： 采用分词器，对文本每个词进行切分，建立词条，方便进行查找

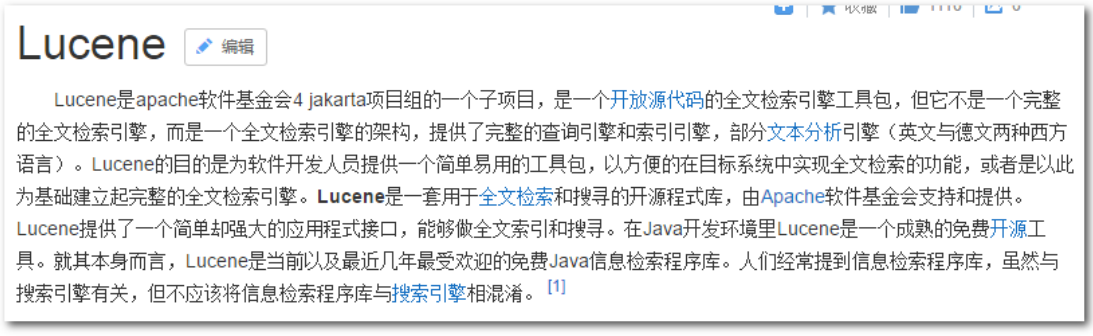
北京市 --- 词条 --- 1,3,4,5,6,9,10 …

海淀区

朝阳

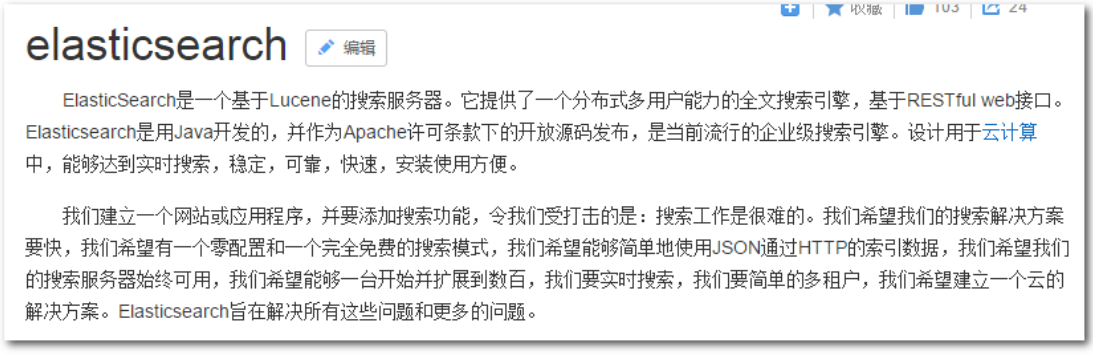
这种索引方式称为倒排索引，像字典目录页 ，“我们” 127 页

2. Java 语言全文检索技术简介



Lucene 就是一套 全文检索 编程 API ， 基于 Lucene 对数据建立索引，进行查询

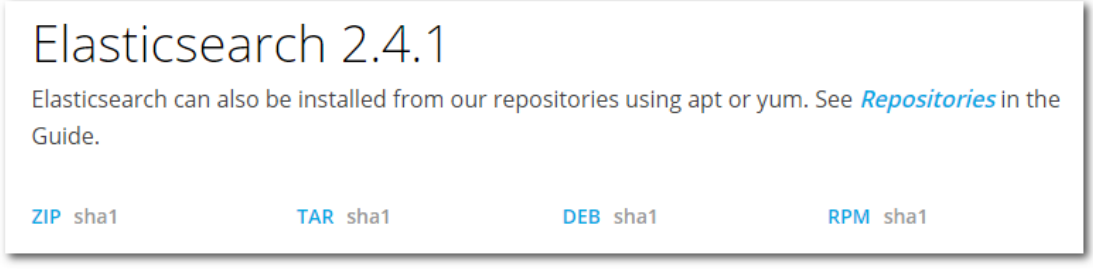
什么是 ElasticSearch ?



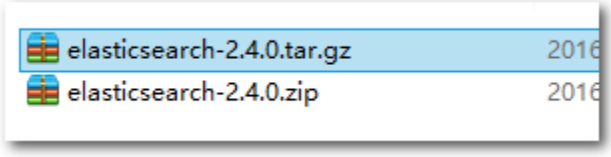
现在企业开发中，更常用是的 solr 搜索服务器和 ElasticSearch 搜索服务器

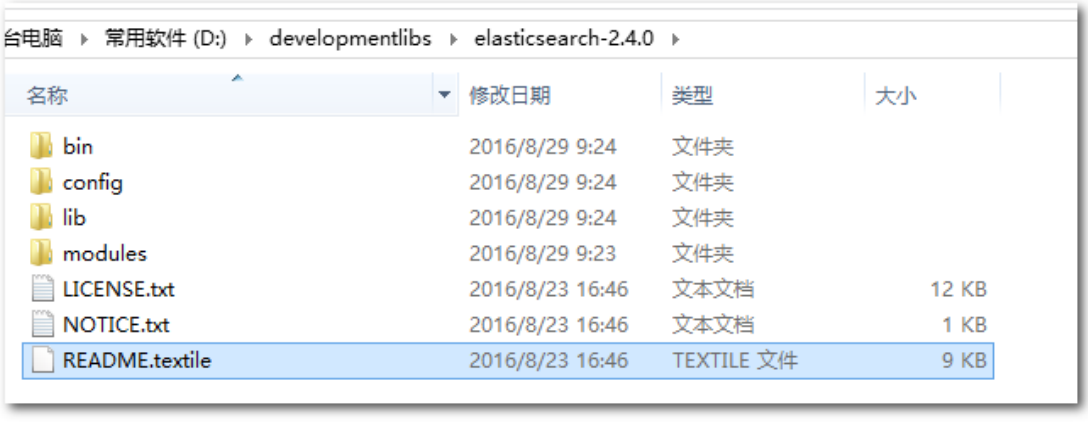
3. ElasticSearch 安装配置使用入门

官网： https://www.elastic.co/products/elasticsearch



Window 系统下载 zip 版本，linux 系统下载 tar 版本





bin 存放 elasticSearch 运行命令

config 存放配置文件

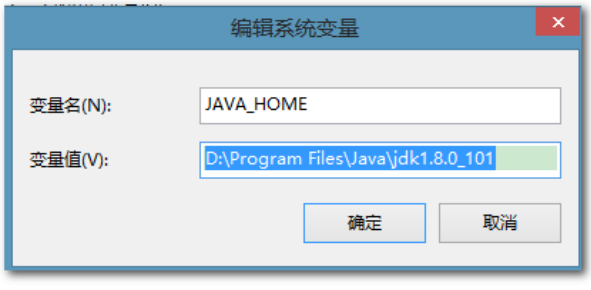
lib 存放 elasticSearch 运行依赖 jar 包

modules 存放 elasticSearch 模块

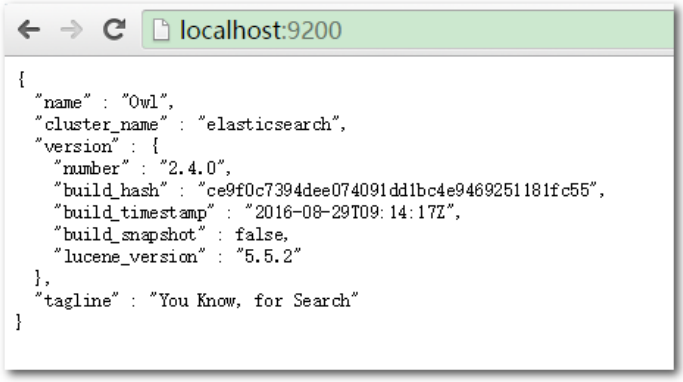
plugins 存放插件

运行 elasticSearch/bin/elasticsearch.bat 文件

配置 JAVA_HOME 环境变量

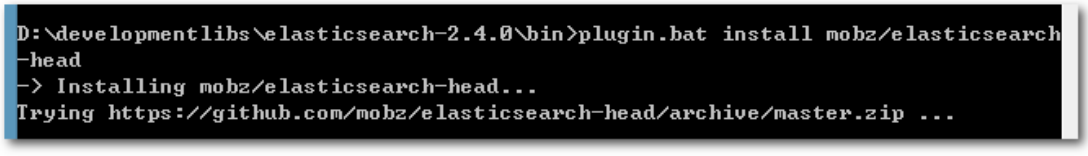


访问 http://127.0.0.1:9200

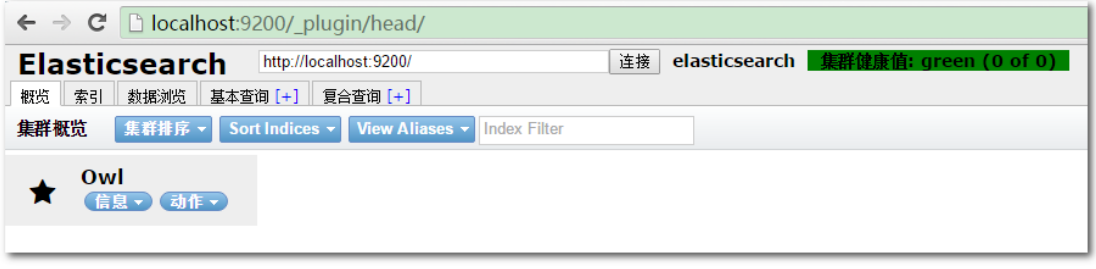


4. ElasticSearch 插件安装 es head

%elasticsearch%/bin/plugin.bat install mobz/elasticsearch-head

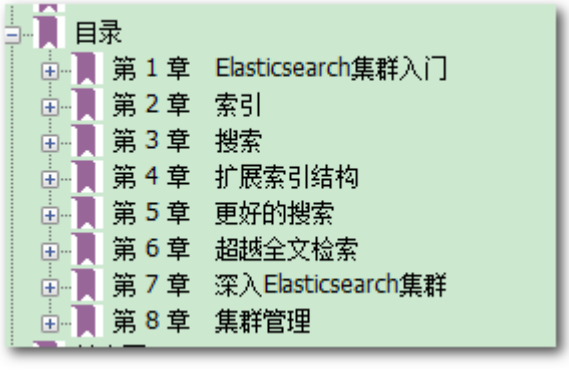


访问 http://localhost:9200/_plugin/head/



5. ElasticSearch 基本操作入门

参考书籍 ： Elasticsearch 服务器开发（第 2 版）.pdf



全文检索： 针对文本中每个词，创建词条建立索引，进行搜索。

ElasticSearch 操作服务器上的数据，通过 Rest API

1.4 用 REST API 操作数据

Elasticsearch REST API可用于各种任务。有了它，可以管理索引，更改实例参数，检查节点和群集状态，索引数据，搜索数据或者通过GET API检索文档。但是现在，我们将集中在API中的CRUD（create-retrieve-update-delete，增删改查）部分，它让我们能像使用NoSQL数据库一样使用Elasticsearch。

```
curl -XPUT http://localhost:9200/blog/article/1 -d '{"title": "New version of Elasticsearch released!", content": "Version 1.0 released today!", "tags": ["announce", "elasticsearch", "release"] }'
```

如果操作 Elasticsearch 上数据，访问提供 Rest API 的 URL 地址，传递 json 数据给服务器

使用JSON标记，一个文档可以如下所示的例子来表示：

```
{
  "id": "1",
  "title": "New version of Elasticsearch released!",
  "content": "Version 1.0 released today!",
  "priority": 10,
  "tags": ["announce", "elasticsearch", "release"]
}
```

5.1. ElasticSearch 基础 数据架构的主要概念

1. 索引

索引（index）是Elasticsearch对逻辑数据的逻辑存储，所以它可以分为更小的部分。你可以把索引看成关系型数据库的表。然而，索引的结构是为快速有效的全文索引准备的，特别是它不存储原始值。如果你知道MongoDB，可以把Elasticsearch的索引看成MongoDB里的一个集合。如果你熟悉CouchDB，可以把索引看成CouchDB数据库索引。Elasticsearch可以把索引存放在一台机器或者分散在多台服务器上，每个索引有一或多个分片（shard），每个分片可以有多个副本（replica）。

2. 文档

存储在Elasticsearch中的主要实体叫文档（document）。用关系型数据库来类比的话，一个文档相当于数据库表中的一行记录。当比较Elasticsearch中的文档和MongoDB中的文档，你会发现两者都可以有不同的结构，但Elasticsearch的文档中，相同字段必须有相同类型。这意味着，所有包含title字段的文档，title字段类型都必须一样，比如string。

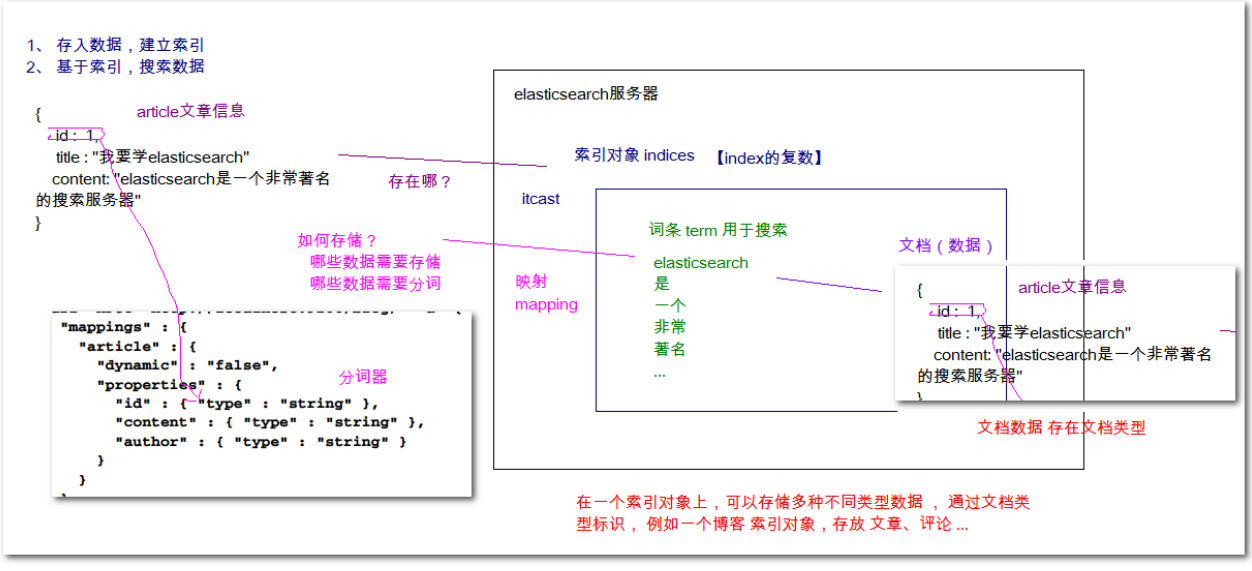
3. 文档类型

在Elasticsearch中，一个索引对象可以存储很多不同用途的对象。例如，一个博客应用程序可以保存文章和评论。文档类型让我们轻易地区分单个索引中的不同对象。每个文档可以有不同的结构，但在实际部署中，将文件按类型区分对数据操作有很大帮助。当然，需要记住一个限制，不同的文档类型不能为相同的属性设置不同的类型。例如，在同一索引中的所有文档类型中，一个叫title的字段必须具有相同的类型。

4. 映射

在有关全文搜索基础知识部分，我们提到了分析的过程：为建索引和搜索准备输入文本，文档中的每个字段都必须根据不同类型做相应的分析。举例来说，对数值字段和从网页抓取的文本字段有不同的分析，比如前者的数字不应该按字母顺序排序，后者的第一步是忽略HTML标签，因为它们是无用的信息噪音。Elasticsearch在映射中存储有关字段的信息。每一个文档类型都有自己的映射，即使我们没有明确定义。

图解



索引对象： 存储数据的表结构 ， 任何搜索数据，存放在索引对象上

映射： 数据如何存放到索引对象上，需要有一个映射配置， 数据类型、是否存储、是否分词 ...

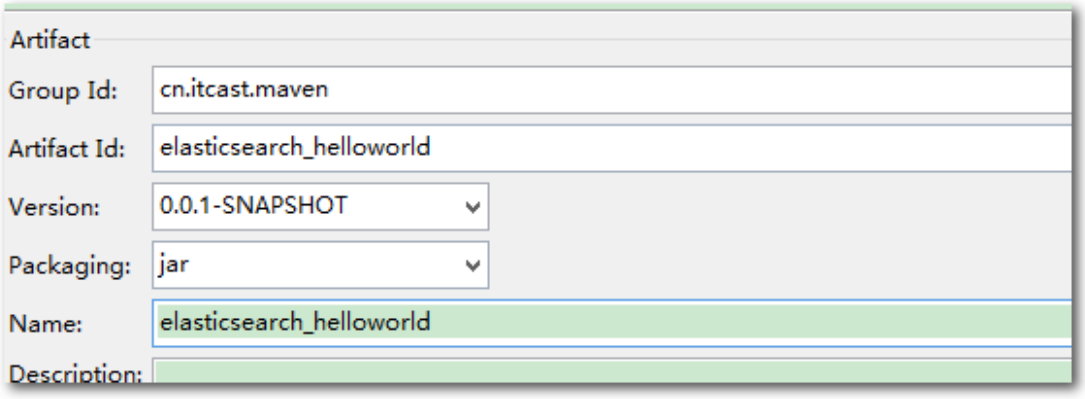
文档： 一条数据记录， 存在索引对象上

文档类型： 一个索引对象 存放多种类型数据， 数据用文档类型进行标识

编程： 建立索引对象 --- 建立映射 --- 存储数据【文档】 --- 指定文档类型进行搜索数据【文档】

5.2. 新建文档（自动创建索引和映射 ）

1、 需要新建 maven 项目



2、 基于 maven 的 pom 导入坐标依赖

```
<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>2.4.0</version>
</dependency>
```

ElasticSearch2.4.0 依赖 lucene5.5.2 版本

当直接在 ElasticSearch 建立文档对象时，如果索引不存在的，默认会自动创建，映射采用默认方式

ElasticSearch 服务默认端口 9300

Web 管理平台端口 9200

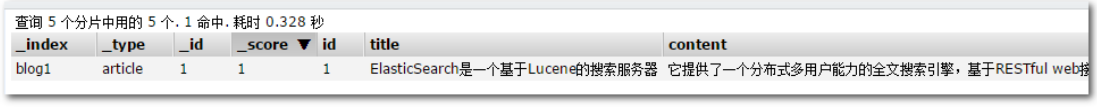
3、 建立文档， 自动创建索引

```
// 直接在ElasticSearch中建立文档，自动创建索引
public void demo1() throws IOException {
    // 创建连接搜索服务器对象
    Client client = TransportClient
            .builder()
            .build()
            .addTransportAddress(
                    new InetSocketTransportAddress(InetAddress
                            .getByName("127.0.0.1"), 9300));
    // 描述json 数据
    /*
     * {id:xxx, title:xxx, content:xxx}
     */
    XContentBuilder builder = XContentFactory
            .jsonBuilder()
            .startObject()
            .field("id", 1)
            .field("title", "ElasticSearch是一个基于Lucene的搜索服务器")
            .field("content",
                    "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。Elasticsearch是用Java开发的
            .endObject();
    // 建立文档对象
    client.prepareIndex("blog1", "article", "1").setSource(builder).get();

    // 关闭连接
    client.close();
```

自动创建索引

自动创建索引映射

文档数据 （type 文档类型 ）



5.3. 搜索文档数据

查询数据 主要依赖 QueryBuilder 对象 ， 可以通过 QueryBuilders 获取

boolQuery() 布尔查询，可以用来组合多个查询条件

fuzzyQuery() 相似度查询

matchAllQuery() 查询所有数据

regexpQuery() 正则表达式查询

termQuery() 词条查询

wildcardQuery() 模糊查询

```
// 搜索在elasticSearch中创建 文档对象
public void demo2() throws IOException {
    // 创建连接搜索服务器对象
    Client client = TransportClient
            .builder()
            .build()
            .addTransportAddress(
                    new InetSocketTransportAddress(InetAddress
                            .getByName("127.0.0.1"), 9300));
    // 搜索数据
    // get() === execute().actionGet()
    SearchResponse searchResponse = client.prepareSearch("blog1")
            .setTypes("article").setQuery(QueryBuilders.matchAllQuery())
            .get();
    SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
    System.out.println("查询结果有：" + hits.getTotalHits() + "条");
    Iterator<SearchHit> iterator = hits.iterator();
    while (iterator.hasNext()) {
        SearchHit searchHit = iterator.next(); // 每个查询对象
        System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
        System.out.println("title:" + searchHit.getSource().get("title"));
    }

    // 关闭连接
    client.close();
}
```

6. 各种查询对象 Query 的使用

ElasticSearch 支持所有 Lucene 查询，并对其进行简化封装

TermQuery 词条查询

WildcardQuery 模糊查询

FuzzyQuery 相似度查询

BooleanQuery 布尔查询

1、ElasticSearch 提供 QueryBuileders.queryStringQuery(搜索内容) 查询方法，对所有字段进行分词查询

```
SearchResponse searchResponse = client.prepareSearch("blog1")
        .setTypes("article")
        .setQuery(QueryBuilders.queryStringQuery("全文")).get();
```

2、 只想查询 content 里包含全文 ，使用 wildcardQuery 磨合查询 *任意字符串 ?任意单个字符

```
SearchResponse searchResponse = client.prepareSearch("blog1")
        .setTypes("article")
        .setQuery(QueryBuilders.wildcardQuery("content", "*全文*")).get();
```

发现查询不到!!!! ，涉及词条查询理解， 说明没有词条包含“全文”

3、 查询 content 词条为“搜索” 内容，使用 TermQuery

```
SearchResponse searchResponse = client.prepareSearch("blog1")
        .setTypes("article")
        .setQuery(QueryBuilders.termQuery("content", "搜索")).get();
```

发现查询不到!!! ，说明没有 “搜索”这个词条

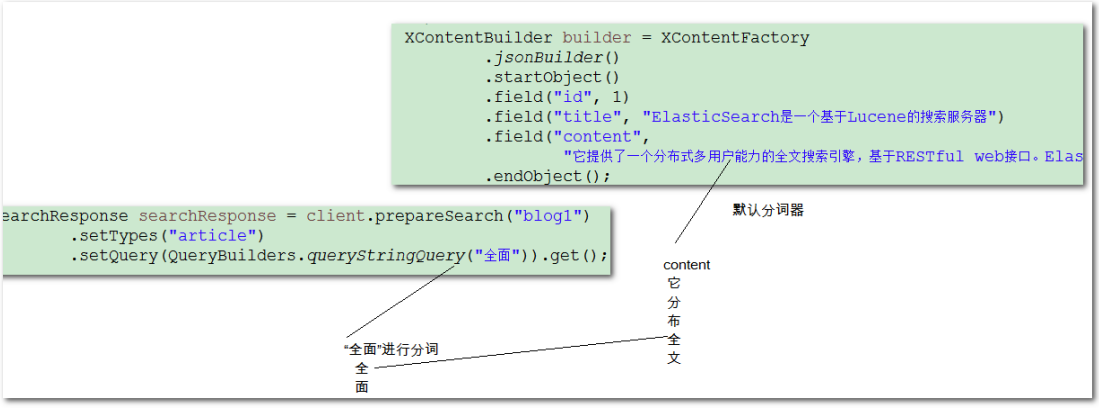
词条： 就是将文本内容存入搜索服务器，搜索服务器进行分词之后内容

“ElasticSearch 是一个基于 Lucene 的搜索服务器”

分词（好的）： ElasticSearch、是、一个、基于、Lucene、搜索、服务、服务器

默认分词（差）： ElasticSearch、 是、一、个、基、于、搜、索

搜索“全面” 能够查询到



7. IK 分词器和 ElasticSearch 集成使用

ElasticSearch 默认采用分词器， 单个字分词 ， 效果很差

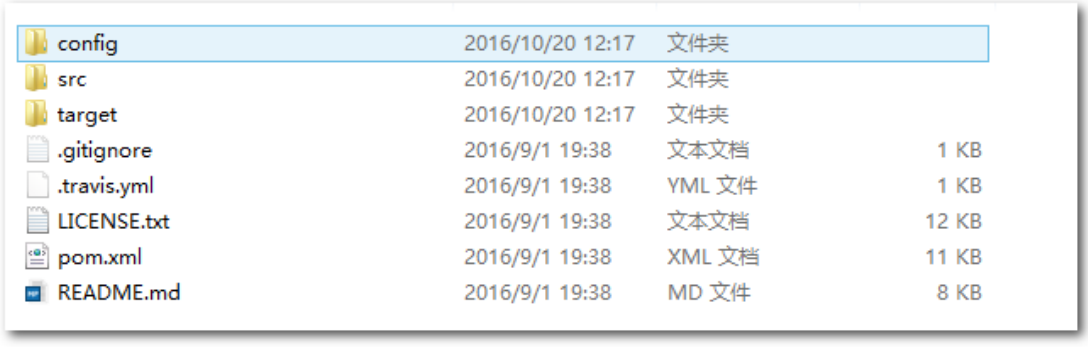
IK Analyzer是一个开源的，基于java语言开发的轻量级的中文分词工具包。从2006年12月推出1.0版开始， IKAnalyzer已经推出了3个大版本。最初，它是以开源项目Luence为应用主体的，结合词典分词和文法分析算法的中文分词组件。新版本的IK Analyzer 3.0则发展为面向Java的公用分词组件，独立于Lucene项目，同时提供了对Lucene的默认优化实现。

1.2 IK Analyzer 3.0特性

- 采用了特有的“正向迭代最细粒度切分算法“，具有50万字/秒的高速处理能力。
- 采用了多子处理器分析模式，支持：英文字母（IP地址、Email、URL）、数字（日期，常用中文数量词，罗马数字，科学计数法），中文词汇（姓名、地名处理）等分词处理。
- 优化的词典存储，更小的内存占用。支持用户词典扩展定义
- 针对Lucene全文检索优化的查询分析器IKQueryParser；采用歧义分析算法优化查询关键字的搜索排列组合，能极大的提高Lucene检索的命中率。

下载 https://github.com/medcl/elasticsearch-analysis-ik/tree/2.x

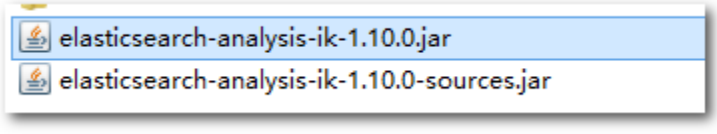
1、 下载开源项目



2、 打包 ik 分词器

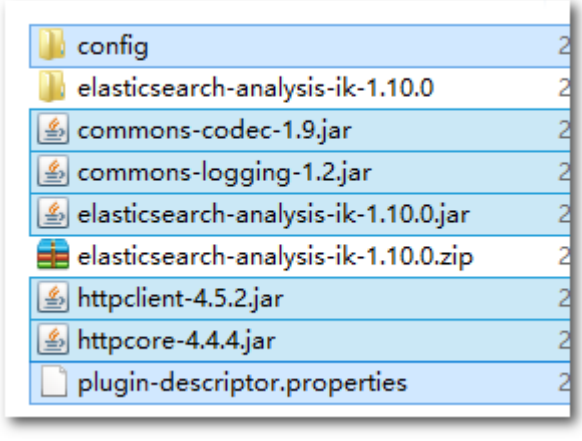
mvn clean 清空

mvn package 打包



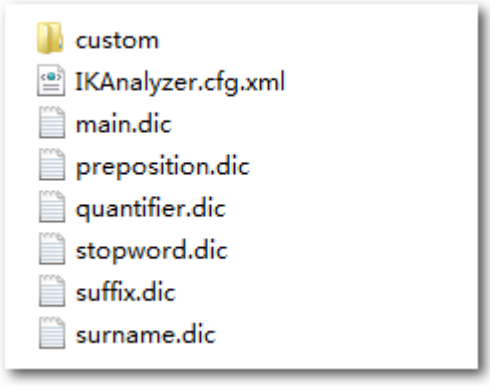
3、 进入 target/release 目录

将下列文件 ， 拷贝到 %es%/plugins/analysis-ik



4、 进入 target/release/config 目录

将所有配置文件，复制 %es%/config 下



5、 配置 elasticsearch.yml

```
index.analysis.analyzer.ik.type: "ik"
```

6、 重启 es

```
2016-10-21 15:55:50,280][INFO ][ik-analyzer                      ] try load config fro
D:\developmentlibs\elasticsearch-2.4.0\config\analysis-ik\IKAnalyzer.cfg.xml
2016-10-21 15:55:50,281][INFO ][ik-analyzer                      ] try load config fro
D:\developmentlibs\elasticsearch-2.4.0\plugins\analysis-ik\config\IKAnalyzer.c
.xml
2016-10-21 15:55:51,289][INFO ][ik-analyzer                      ] [Dict Loading] cust
\mydict.dic
2016-10-21 15:55:51,295][INFO ][ik-analyzer                      ] [Dict Loading] cust
\single_word_low_freq.dic
2016-10-21 15:55:51,306][INFO ][ik-analyzer                      ] [Dict Loading] cust
\ext_stopword.dic
```

发现 ik 分词器被加载

7、 访问

http://localhost:9200/_analyze?analyzer=ik&pretty=true&text=我是中国人

```
{
  "tokens" : [ {
    "token" : "我",
    "start_offset" : 0,
    "end_offset" : 1,
    "type" : "CN_CHAR",
    "position" : 0
  }, {
    "token" : "中国人",
    "start_offset" : 2,
    "end_offset" : 5,
    "type" : "CN_WORD",
    "position" : 1
  }, {
    "token" : "中国",
    "start_offset" : 2,
    "end_offset" : 4,
    "type" : "CN_WORD",
    "position" : 2
  }, {
    "token" : "国人",
    "start_offset" : 3,
    "end_offset" : 5,
    "type" : "CN_WORD",
    "position" : 3
  } ]
}
```

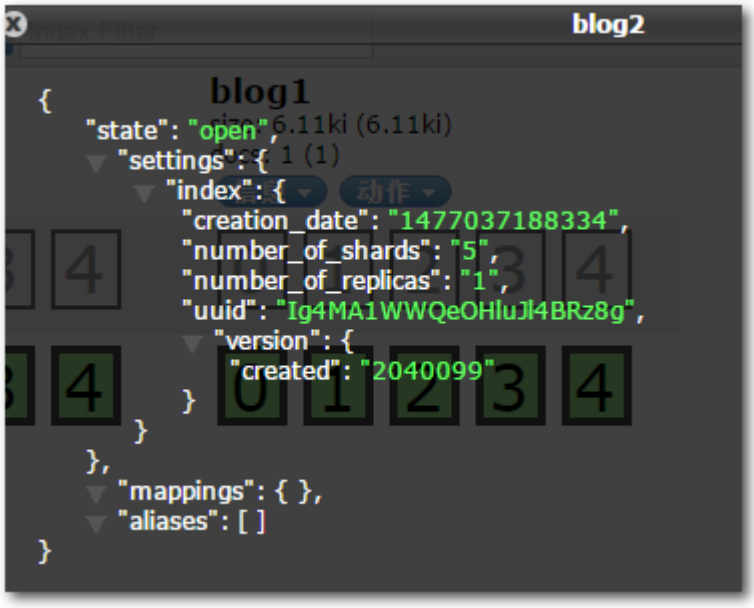

8. ElasticSearch 常用编程操作

在 ElasticSearch 没有索引情况下，插入文档，默认创建索引和索引映射 （无法使用 ik 分词器）

8.1. 索引相关操作

- 创建索引

```
// 创建索引
client.admin().indices().prepareCreate("blog2").get();
```



默认创建好索引，mappings 为空

- 删除索引

```
// 删除索引
client.admin().indices().prepareDelete("blog2").get();
```

8.2. 映射相关操作

```
curl -XPUT 'http://localhost:9200/blog/' -d '{
  "mappings" : {
    "article" : {
      "dynamic" : "false",
      "properties" : {
        "id" : { "type" : "string" },
        "content" : { "type" : "string" },
        "author" : { "type" : "string" }
      }
    }
  }
}'
```

调用 client.admin().indices().putMapping(mapping).get();

```
// 添加映射
XContentBuilder builder = XContentFactory.jsonBuilder().startObject()
        .startObject("article").startObject("properties")
        .startObject("id").field("type", "integer")
        .field("store", "yes").endObject().startObject("title")
        .field("type", "string").field("store", "yes")
        .field("analyzer", "ik").endObject().startObject("content")
        .field("type", "string").field("store", "yes")
        .field("analyzer", "ik").endObject().endObject().endObject()
        .endObject();

PutMappingRequest mapping = Requests.putMappingRequest("blog2")
        .type("article").source(builder);
client.admin().indices().putMapping(mapping).get();
```

8.3. 文档相关操作

- 建立文档

  1、 直接在 XContentBuilder 中构建 json 数据，建立文档

```
 */
XContentBuilder builder = XContentFactory
        .jsonBuilder()
        .startObject()
        .field("id", 1)
        .field("title", "ElasticSearch是一个基于Lucene的搜索服务器")
        .field("content",
                "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。Elasticsearch是
        .endObject();
// 建立文档对象
client.prepareIndex("blog1", "article", "1").setSource(builder).get();
```

2、 对一个已经存在对象，转换为 json ， 建立文档

```
// 实体类
public class Article {
    private Integer id;
    private String title;
    private String content;
```

如何将 Article 对象，转换为 json 数据 ---- Jackson 转换开发包

Jackson 是一个 Java 用来处理 JSON 格式数据的类库，性能非常好。

引入 jackson

1.x

```
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-core-asl</artifactId>
    <version>1.9.13</version>
</dependency>
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.13</version>
</dependency>
```

2.x

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.8.1</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.8.1</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.8.1</version>
```

</dependency>

```
ObjectMapper objectMapper = new ObjectMapper();

// 建立文档
client.prepareIndex("blog2", "article", article.getId().toString())
        .setSource(objectMapper.writeValueAsString(article)).get();
```

- 修改文档

方式一： 使用 prepareUpdate、prepareIndex 都可以

```
// 修改文档
client.prepareUpdate("blog2", "article", article.getId().toString())
        .setDoc(objectMapper.writeValueAsString(article)).get();
```

方式二： 直接使用 update

```
// 修改文档
client.update(
        new UpdateRequest("blog2", "article", article.getId()
                .toString()).doc(objectMapper
                .writeValueAsString(article))).get();
```

- 删除文档

方式一： prepareDelete

```
// 删除文档
client.prepareDelete("blog2", "article", article.getId().toString())
        .get();
```

方式二： 直接使用 delete

```
// 删除文档
client.delete(
        new DeleteRequest("blog2", "article", article.getId()
                .toString())).get();
```

8.4. 查询文档分页操作

1、 批量向数据表 插入 100 条记录

```
ObjectMapper objectMapper = new ObjectMapper();

for (int i = 1; i <= 100; i++) {
    // 描述json 数据
    Article article = new Article();
    article.setId(i);
    article.setTitle(i + "搜索工作其实很快乐");
    article.setContent(i
            + "我们希望我们的搜索解决方案要快，我们希望有一个零配置和一个完全免费的搜索模式，我们希望自

    // 建立文档
    client.prepareIndex("blog2", "article", article.getId().toString())
            .setSource(objectMapper.writeValueAsString(article)).get();
}
```

2、 分页查询

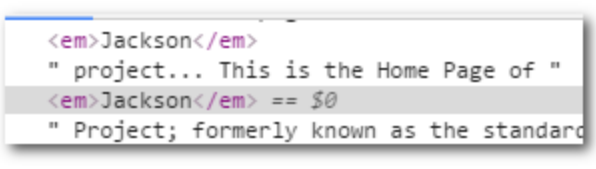
searchRequestBuilder 的 setFrom【从 0 开始】 和 setSize【查询多少条】方法实现

```
// 搜索数据
// get() === execute().actionGet()
SearchRequestBuilder searchRequestBuilder = client
        .prepareSearch("blog2").setTypes("article")
        .setQuery(QueryBuilders.matchAllQuery());

// 查询第2页数据，每页20条
searchRequestBuilder.setFrom(20).setSize(20);

SearchResponse searchResponse = searchRequestBuilder.get();
```

8.5. 查询结果高亮显示

在百度搜索 jackson

页面源码分析



1、 配置应用高亮

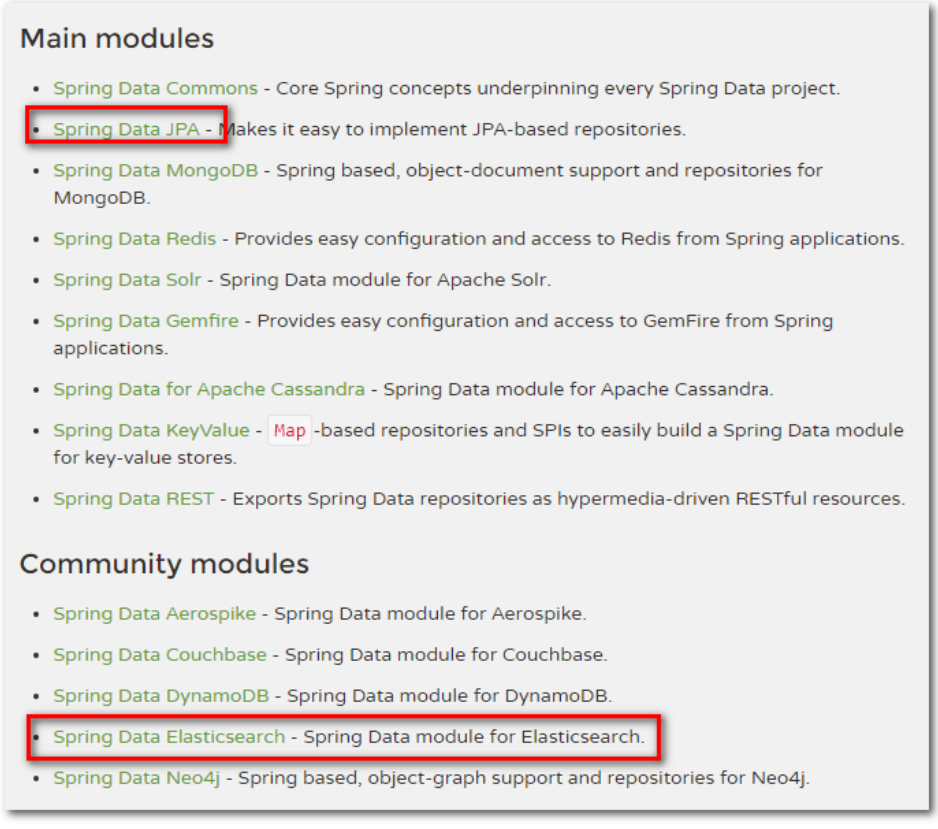
```
// 搜索数据
SearchRequestBuilder searchRequestBuilder = client
        .prepareSearch("blog2").setTypes("article")
        .setQuery(QueryBuilders.termQuery("title", "搜索"));

// 高亮定义
searchRequestBuilder.addHighlightedField("title"); // 对title字段进行高亮
searchRequestBuilder.setHighlighterPreTags("<em>"); // 前置元素
searchRequestBuilder.setHighlighterPostTags("</em>");// 后置元素

SearchResponse searchResponse = searchRequestBuilder.get();
```

2、 对结果的高亮片段做拼接处理，替换原有内容

```
// 将高亮处理后内容，替换原有内容（原有内容，可能会出现显示不全）
Map<String, HighlightField> highlightFields = searchHit
        .getHighlightFields();
HighlightField titleField = highlightFields.get("title");

// 获取到原有内容中每个高亮显示集中位置fragment 就是高亮片段
Text[] fragments = titleField.fragments();
String title = "";
for (Text text : fragments) {
    title += text;
}
// 将查询结果转换为对象
Article article = objectMapper.readValue(
        searchHit.getSourceAsString(), Article.class);

// 用高亮后内容，替换原有内容
article.setTitle(title);

System.out.println(article);
```

9. Spring Data ElasticSearch 使用

9.1. Spring Data ElasticSearch 简介



Main modules

- Spring Data Commons - Core Spring concepts underpinning every Spring Data project.
- Spring Data JPA - Makes it easy to implement JPA-based repositories.
- Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.
- Spring Data Redis - Provides easy configuration and access to Redis from Spring applications.
- Spring Data Solr - Spring Data module for Apache Solr.
- Spring Data Gemfire - Provides easy configuration and access to GemFire from Spring applications.
- Spring Data for Apache Cassandra - Spring Data module for Apache Cassandra.
- Spring Data KeyValue - `Map`-based repositories and SPIs to easily build a Spring Data module for key-value stores.
- Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.

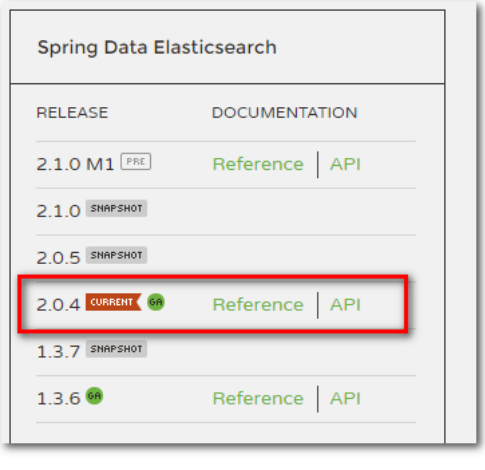
Community modules

- Spring Data Aerospike - Spring Data module for Aerospike.
- Spring Data Couchbase - Spring Data module for Couchbase.
- Spring Data DynamoDB - Spring Data module for DynamoDB.
- Spring Data Elasticsearch - Spring Data module for Elasticsearch.
- Spring Data Neo4j - Spring based, object-graph support and repositories for Neo4j.

什么是 spring data elasticSearch ?

Spring Data ElasticSearch 基于 spring data API 简化 elasticSearch 操作 ， 将原始操作 elasticSearch 的客户端 API 进行封装

官方网站：http://projects.spring.io/spring-data-elasticsearch/



Spring Data Elasticsearch

RELEASE	DOCUMENTATION
2.1.0 M1 PRE	Reference | API
2.1.0 SNAPSHOT	
2.0.5 SNAPSHOT	
2.0.4 CURRENT GA	Reference | API
1.3.7 SNAPSHOT	
1.3.6 GA	Reference | API

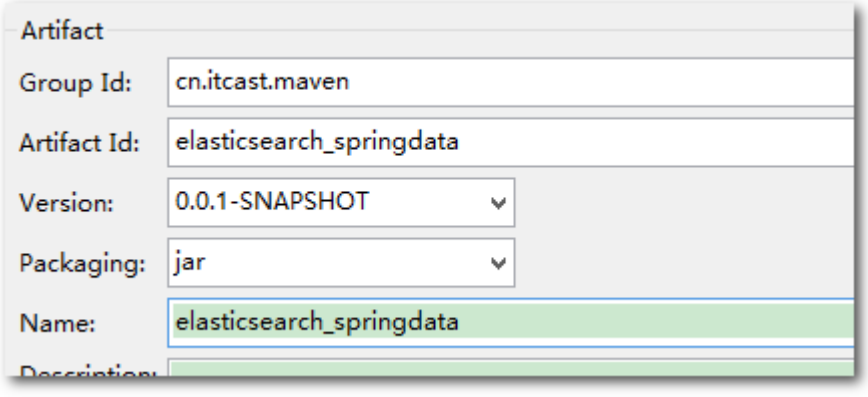
下载官方规范文档

Maven 坐标

```
<dependencies>
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-elasticsearch</artifactId>
        <version>2.0.4.RELEASE</version>
    </dependency>
</dependencies>
```

9.2. Spring Data ElasticSearch 入门案例

1、 创建 maven 工程



2、 基于 maven 导入坐标

Spring data elasticsearch 对 elasticsearch api 简化封装

```
<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>2.4.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-elasticsearch</artifactId>
    <version>2.0.4.RELEASE</version>
</dependency>
```

导入 spring-test 和 junit 编写测试用例

```xml
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>4.1.7.RELEASE</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
```

Slf4j-log4j 日志包

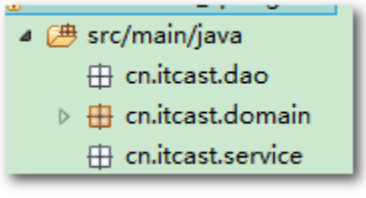
```xml
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.12</version>
</dependency>
```

3、 在 src/main/resources 下建立 applicationContext.xml 和 log4j.properties

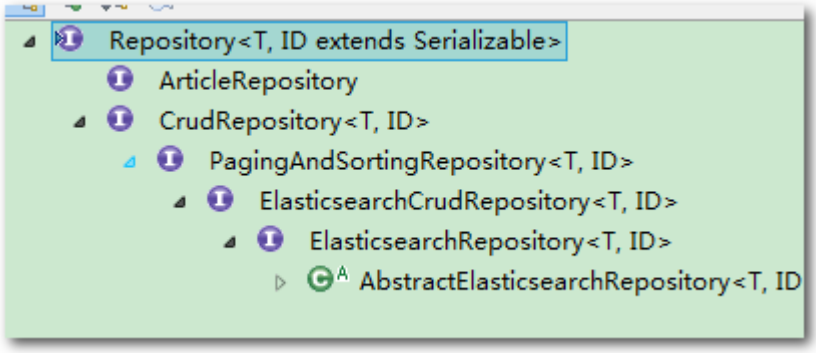
引入 spring data elasticsearch 名称空间

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:elasticsearch="http://www.springframework.org/schema/data/elasticsearch"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/data/elasticsearch
        http://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd ">
```

4、 创建 domain、dao、service 包



5、 编写 DAO



编写 DAO 自动操作 elasticsearch 继承 ElasticsearchRepository 接口

```
public interface ArticleRepository extends
        ElasticsearchRepository<Article, Integer> {
```

配置 applicationContext.xml

```
<!-- 扫描DAO包 自动创建实现 -->
<elasticsearch:repositories base-package="cn.itcast.dao" />
```

6、 编写 Service

```
@Service
public class ArticleServiceImpl implements ArticleService {

    @Autowired
    private ArticleRepository articleRepository;

    public void save(Article article) {
        articleRepository.save(article);
    }
```

配置 applicationContext.xml

```
<!-- 扫描Service包 -->
<context:component-scan base-package="cn.itcast.service" />
```

7、 配置 applicationContext.xml 连接 elasticsearch

```
<!-- 配置elasticsearch 连接 -->
<elasticsearch:transport-client id="client" cluster-nodes="localhost:9300" />

<!-- spring data elasticsearch DAO 必须依赖 elasticsearchTemplate  -->
<bean id="elasticsearchTemplate"
    class="org.springframework.data.elasticsearch.core.ElasticsearchTemplate">
    <constructor-arg name="client" ref="client" />
</bean>
```

8、 索引和映射如何创建 --- 基于 spring data elasticsearch 注解

在使用 spring data elasticsearch 开发， 需要将索引和映射信息 配置实体类上面

@Document 文档对象 （索引信息、文档类型 ）

@Id 文档主键 唯一标识

@Field 每个文档的字段配置（类型、是否分词、是否存储、分词器 ）

```
@Document(indexName = "blog3", type = "article")
public class Article {
    @Id
    @Field(index = FieldIndex.not_analyzed, store = true)
    private Integer id;
    @Field(index = FieldIndex.analyzed, analyzer = "ik", store = true, searchAnalyzer = "ik")
    private String title;
    @Field(index = FieldIndex.analyzed, analyzer = "ik", store = true, searchAnalyzer = "ik")
    private String content;
```

通过 ElasticsearchTemplate 创建索引和添加映射

```
@Test
public void createIndex() {
    elasticsearchTemplate.createIndex(Article.class);
    elasticsearchTemplate.putMapping(Article.class);
}
```

错误： 导包冲突

java.lang.NoClassDefFoundError: org/springframework/core/ResolvableTypeProvider

解决： spring 导包版本要一致

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>4.2.8.RELEASE</version>
</dependency>
```

错误： MapperParsingException[No type specified for field [title]]

解决： 配置类型

```java
@Document(indexName = "blog3", type = "article")
public class Article {
    @Id
    @Field(index = FieldIndex.not_analyzed, store = true, type = FieldType.Integer)
    private Integer id;
    @Field(index = FieldIndex.analyzed, analyzer = "ik",
            store = true, searchAnalyzer = "ik", type = FieldType.String)
    private String title;
    @Field(index = FieldIndex.analyzed, analyzer = "ik",
            store = true, searchAnalyzer = "ik", type = FieldType.String)
    private String content;
```

9、 Spring data Search CRUD 操作

CurdRepository 提供增删改查 save、delete、findAll 、findOne

PagingAndSortingRepository 提供分页和排序

```java
public void save(Article article) {
    articleRepository.save(article);
}

public void delete(Article article) {
    articleRepository.delete(article);
}

public Article findOne(Integer id) {
    return articleRepository.findOne(id);
}

public Iterable<Article> findAll() {
    return articleRepository.findAll(new Sort(new Sort.Order(
            Sort.Direction.ASC, "id")));
}

public Page<Article> findAll(Pageable pageable) {
    return articleRepository.findAll(pageable);
}
```

10、条件查询 （分页）

Example 44. Query creation from method names

```java
public interface BookRepository extends Repository<Book, String>
{
    List<Book> findByNameAndPrice(String name, Integer price);
}
```

The method name above will be translated into the following Elasticsearch json query

```
{ "bool" :
    { "must" :
        [
            { "field" : {"name" : "?"} },
            { "field" : {"price" : "?"} }
        ]
    }
}
```

查询标题方法

```
public interface ArticleRepository extends
        ElasticsearchRepository<Article, Integer> {

    List<Article> findByTitle(String title);

}
```

分页条件查询，只需要在查询方法中，添加 Pageable 对象

排序条件查询，只需要在查询方法中，添加 Sort 对象

```
public interface ArticleRepository extends
        ElasticsearchRepository<Article, Integer> {

    List<Article> findByTitle(String title);

    Page<Article> findByTitle(String title, Pageable pageable);

}
```

10. 运单管理功能 同步操作 索引库

10.1. 录入运单，同步创建索引

1、 在 bos_management 引入 elasticsearch 和 spring data elasticsearch 支持

```
<!-- elasticsearch  -->
<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>2.4.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-elasticsearch</artifactId>
    <version>2.0.4.RELEASE</version>
</dependency>
```

提升 spring 依赖版本 4.1.7 提升 4.2.8

2、 需要在实体类 WayBill 对象，添加 elasticsearch 索引和映射信息

操作 bos_domain 项目

```java
@Document(indexName = "bos", type = "waybill")
public class WayBill implements Serializable {

    @Id
    @GeneratedValue
    @Column(name = "C_ID")
    @org.springframework.data.annotation.Id
    @Field(index = FieldIndex.not_analyzed, store = true, type = FieldType.Integer
    private Integer id;
    @Column(name = "C_WAY_BILL_NUM", unique = true)
    @Field(index = FieldIndex.not_analyzed, store = true, type = FieldType.String)
    private String wayBillNum; // 运单编号
    @OneToOne
```

3、 配置 elasticsearch

applicationContext-elasticsearch.xml

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:elasticsearch="http://www.springframework.org/schema/data/elasticsearch"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/data/elasticsearch
        http://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd">

    <!-- 搜索DAO 扫描 -->
    <elasticsearch:repositories base-package="cn.itcast.bos.index" />

    <!-- 配置Client -->
    <elasticsearch:transport-client id="client" cluster-nodes="127.0.0.1:9300"/>

    <!-- 配置搜索模板 -->
    <bean id="elasticsearchTemplate"
        class="org.springframework.data.elasticsearch.core.ElasticsearchTemplate">
        <constructor-arg name="client" ref="client" />
    </bean>
</beans>
```

4、 操作索引库 创建 DAO

```java
public interface WayBillIndexRepository extends
        ElasticsearchRepository<WayBill, Integer> {
```

5、 修改 Service 代码

```
@Autowired
private WayBillIndexRepository wayBillIndexRepository;

@Override
public void save(WayBill wayBill) {
    // 判断运单号是否存在
    WayBill persistWayBill = wayBillRepository.findByWayBillNum(wayBill
            .getWayBillNum());
    if (persistWayBill.getId() == null) {
        // 运单不存在
        wayBillRepository.save(wayBill);
        // 保存索引
        wayBillIndexRepository.save(wayBill);
    } else {
        // 运单存在
        try {
            Integer id = persistWayBill.getId();
            BeanUtils.copyProperties(persistWayBill, wayBill);
            persistWayBill.setId(id);
            // 保存索引
            wayBillIndexRepository.save(persistWayBill);
```

Bug 修复 空指针异常

```
            .getWayBillNum());
if (persistWayBill == null || persistWayBill.getId() == null) {
    // 运单不存在
```

10.2. 搜索运单，基于索引库查询

通过索引库完成查询

1、 将查询 form 的数据，转换 json 格式，绑定数据表格上

对查询内容，添加 form 元素

```
<form id="searchForm">
    运单号: <input name="wayBillNum" style="line-height:26px;border:1px solid #ccc">
    发货地: <input name="sendAddress" style="line-height:26px;border:1px solid #ccc" >
    收货地: <input name="recAddress" style="line-height:26px;border:1px solid #ccc" >
```

点击查询按钮，执行 doSearch 方法

```
<a href="javascript:void" class="easyui-linkbutton"
    plain="true" onclick="doSearch()">查询</a>
```

```
$.fn.serializeJson=function(){
    var serializeObj={};
    var array=this.serializeArray();
    var str=this.serialize();
    $(array).each(function(){
        if(serializeObj[this.name]){
            if($.isArray(serializeObj[this.name])){
                serializeObj[this.name].push(this.value);
            }else{
                serializeObj[this.name]=[serializeObj[this.name],this.value];
            }
        }else{
            serializeObj[this.name]=this.value;
        }
    });
    return serializeObj;
};
```

将 form 数据转换 json 绑定 datagrid 上

```
function doSearch() {
    // 将form内容，转换为json
    var queryParams = $("#searchForm").serializeJson();

    // 绑定数据表格 重新加载数据
    $('#tt').datagrid('load', queryParams);
}
```

2、 修改 WayBillAction 的 pageQuery 方法


```
@Action(value = "waybill_pageQuery", results = { @Result(name = "success", type = "json") })
public String pageQuery() {
    // 无条件查询
    Pageable pageable = new PageRequest(page - 1, rows, new Sort(
            new Sort.Order(Sort.Direction.DESC, "id")));
    // 调用业务层进行查询
    Page<WayBill> pageData = wayBillService.findPageData(model, pageable);
    // 压入值栈返回
    pushPageDataToValueStack(pageData);

    return SUCCESS;
}
```

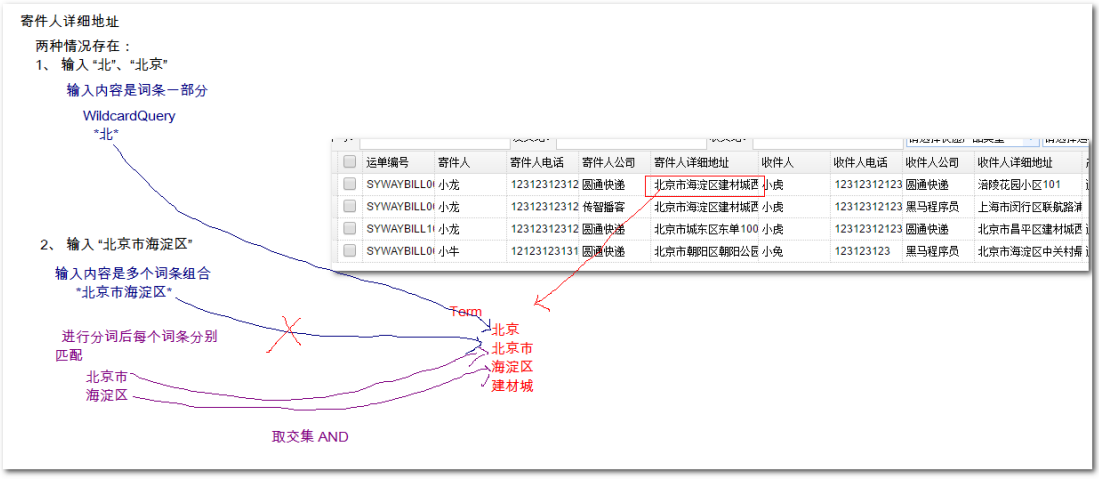
3、 编写业务层 findPageData 方法

```java
@Override
public Page<WayBill> findPageData(WayBill wayBill, Pageable pageable) {
    // 判断WayBill 中条件是否存在
    if (StringUtils.isBlank(wayBill.getWayBillNum())
            && StringUtils.isBlank(wayBill.getSendAddress())
            && StringUtils.isBlank(wayBill.getRecAddress())
            && StringUtils.isBlank(wayBill.getSendProNum())
            && (wayBill.getSignStatus() == null || wayBill.getSignStatus() == 0)) {
        // 无条件查询 、查询数据库
        return wayBillRepository.findAll(pageable);
    } else {
        // 查询条件
        // must 条件必须成立 and
        // must not 条件必须不成立 not
        // should 条件可以成立 or
        BoolQueryBuilder query = new BoolQueryBuilder(); // 布尔查询，多条件组合查询
        // 向组合查询对象添加条件
        if (StringUtils.isNoneBlank(wayBill.getWayBillNum())) {
            // 运单号查询
            QueryBuilder tempQuery = new TermQueryBuilder("wayBillNum",
                    wayBill.getWayBillNum());
            query.must(tempQuery);
        }
```

```java
        if (StringUtils.isNoneBlank(wayBill.getSendAddress())) {
            // 发货地 模糊查询
            QueryBuilder wildcardQuery = new WildcardQueryBuilder(
                    "sendAddress", "*" + wayBill.getSendAddress() + "*");
            query.must(wildcardQuery);
        }
        if (StringUtils.isNoneBlank(wayBill.getRecAddress())) {
            // 收货地 模糊查询
            QueryBuilder wildcardQuery = new WildcardQueryBuilder(
                    "recAddress", "*" + wayBill.getRecAddress() + "*");
            query.must(wildcardQuery);
        }
        if (StringUtils.isNoneBlank(wayBill.getSendProNum())) {
            // 速运类型 等值查询
            QueryBuilder termQuery = new TermQueryBuilder("sendProNum",
                    wayBill.getSendProNum());
            query.must(termQuery);
        }
        if (StringUtils.isNoneBlank(wayBill.getSendProNum())) {
            // 速运类型 等值查询
            QueryBuilder termQuery = new TermQueryBuilder("sendProNum",
                    wayBill.getSendProNum());
            query.must(termQuery);
        }
```

```java
        if (wayBill.getSignStatus() != null && wayBill.getSignStatus() != 0) {
            // 签收状态查询
            QueryBuilder termQuery = new TermQueryBuilder("signStatus",
                    wayBill.getSignStatus());
            query.must(termQuery);
        }

        SearchQuery searchQuery = new NativeSearchQuery(query);
        searchQuery.setPageable(pageable); // 分页效果
        // 有条件查询 、查询索引库
        return wayBillIndexRepository.search(searchQuery);
    }
```

运单搜索，基于索引库词条完成搜索



添加输入多个词条组合，分词查询效果代码

```
}
if (StringUtils.isNoneBlank(wayBill.getSendAddress())) {
    // 发货地 模糊查询
    // 情况一：输入"北" 是查询词条一部分，使用模糊匹配词条查询
    QueryBuilder wildcardQuery = new WildcardQueryBuilder(
            "sendAddress", "*" + wayBill.getSendAddress() + "*");

    // 情况二：输入"北京市海淀区" 是多个词条组合，进行分词后 每个词条匹配查询
    QueryBuilder queryStringQueryBuilder = new QueryStringQueryBuilder(
            wayBill.getSendAddress()).field("sendAddress")
            .defaultOperator(Operator.AND);

    // 两种情况取or关系
    BoolQueryBuilder boolQueryBuilder = new BoolQueryBuilder();
    boolQueryBuilder.should(wildcardQuery);
    boolQueryBuilder.should(queryStringQueryBuilder);

    query.must(boolQueryBuilder);
}
```