# 今日任务

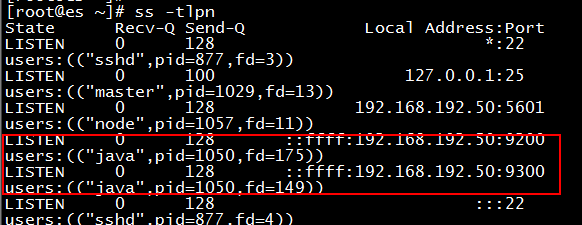
1. 熟悉ES Java客户端版本特性
2. 掌握常用API操作
3. 安装、配置IK中文分词器

# Java客户端

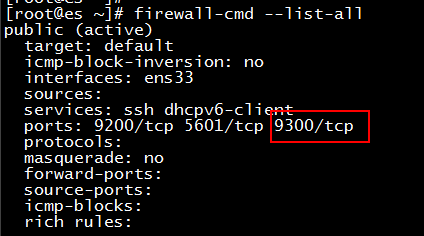
默认es启动后对外开放2个端口号：9200,9300

**9200 对应的是http协议 restful api调用**

**9300 对应的是tcp协议 java 客户端、内部通信**



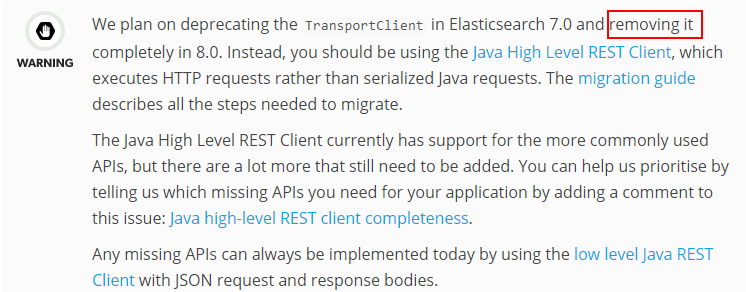
请注意如果使用java客户端，默认的连接端口号是**9300**，一定请注意，注意，注意！！

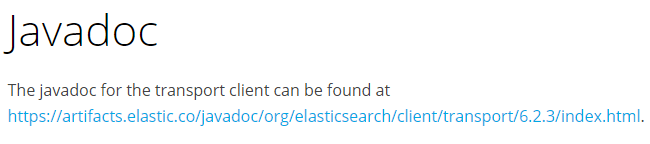


一定要开防火墙！！！！或者练习时直接禁用防火墙也行。

## api说明

未来版本过时的**transportClient**客户端，其实一点都不旧，是ES自己更新太快了，目前企业用的都是这个。







注意：新的java rest api是在6.0测试版中才开始加入的。

api：<https://www.elastic.co/guide/en/elasticsearch/client/java-api/index.html>

rest api网址：

<https://www.elastic.co/guide/en/elasticsearch/client/java-rest/6.2/java-rest-high.html>

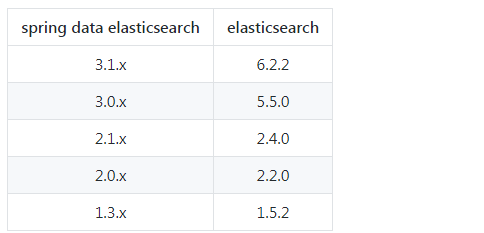
**重要提示：以上虽然给出了网站链接，但并不是要同学们自己去学习，而是想给大家再提个醒，前面就跟大家说过，ES更新实在是太快了，几乎是疯狂的，关键是各个版本之间差异很大，API向下的兼容性很差，在做服务器升级时一定要谨慎选择版本。另外虽然官网提示transportclient在后期版本中会删除，但目前我建议还是使用该客户端。**

**各版本使用时请注意版本号的兼容性！！而且客户端和服务器也要保持一致！！**

**一般主版本和次版本一致即可！！**

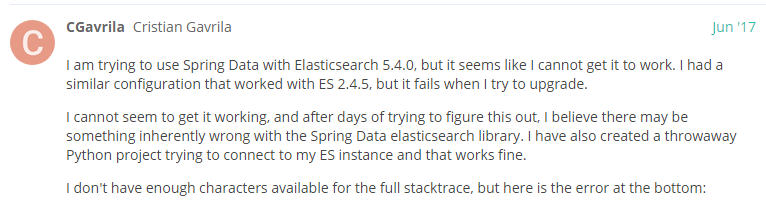
## 坑的故事

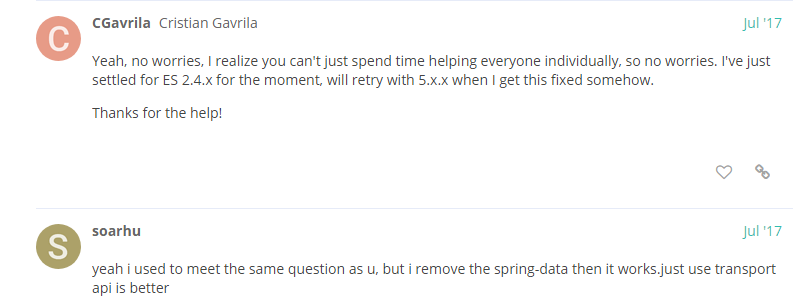
大家知道spring工程体系中，有个spring-data-\*的子工程，不知道也没关系，后面我们讲redis的时候会详细说，简单点，就是spring为方便程序员访问nosql数据库做的一个工具包，就像spring给关系数据库提供的JDBCTemplate一样。所以针对ES，spring提供了一个spring-data-elasticsearch的工程：<https://github.com/spring-projects/spring-data-elasticsearch> ，我呢，和其他的程序员一样，开心的不得了，就使用了起来，而且官网还很贴心的给出了版本使用对照表，如下：



经过一下午的采坑，说明如下，目前es本身开发进度太快，导致spring-data-es和spring-boot-es开发跟不上es的节奏，对新版本的支持很不好，按照官网文档介绍，成功入坑，所有的版本按官网配置后，一直报错！！

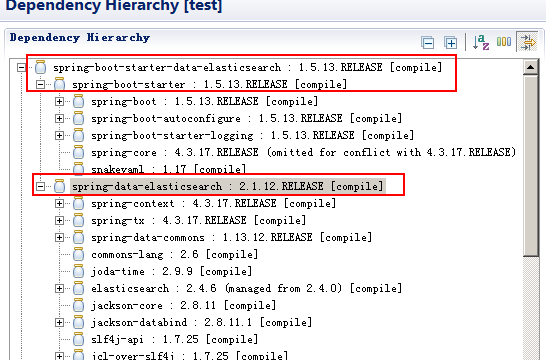
官网目前支持到es-2.4.6，这也是目前官网默认导入的版本，据说这个版本可以使用，但本人没有测试过，由于我们使用的是es-6.5.4，所以目前官网提供的ElasticsearchTemplate类在创建的时候一直报错，各种issues，开发者本人也没有最终去解决，只是说是spring自己的问题，如下图：





具体网址：<https://discuss.elastic.co/t/errors-when-using-spring-data-elasticsearch-version-3-0-0-build-snapshot-and-elasticsearch-5-4-0/90525/4>

有同学会想到在maven中指定版本，但是会出现各种依赖不匹配问题，就是恐怖的maven依赖地狱。



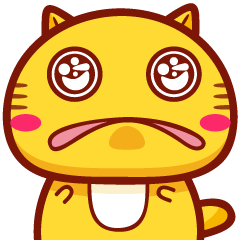
经过此次事件，我们又进一步认识到，开源的软件，新版本真的要谨慎使用！！！！

**有时候即使提供了完善的文档，也不敢保证就一定不会入坑，不要盲目迷信所谓的官网或者专家，有时候能相信的人只有你自己！！**

最后决定放弃spring官方的boot以及data，自己来写boot啦，没办法了！！！！！

C:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.pngC:\Users\ADMINI~1\AppData\Local\Temp\SGPicFaceTpBq\4952\116C8189.png

**重要提示：不到万不得已，新手不要尝试自己去创造轮子。**



## 拥抱开源

具体开发的过程，就不带大家去写了，因为这个还是有点难度的，直接提供打包好的jar包给同学们使用了，去课件资源里去寻找吧。这个据我所知，可是目前唯一的一个版本哦。所以我也不藏着了，将它开源了，放在了我的码云里：<https://gitee.com/codefarmercode/elasticsearch-spring-boot-starter>

提示：如果有同学使用过solr，一定知道IK分词器，不过原版的不能进行智能分词配置，我增强了该功能，也把它开源了：<https://gitee.com/codefarmercode/IKAnalyzer>

以后说不定就有哪位同学需要哦，别忘了这有现成的，还是经过线上检验过的。

**福利：以后同学们工作中需要进一步提高的，请继续关注本人推出的java进阶课程。**

# API练习



建立一个新的搜索微服务工程go9-search，使用我们自己的starter，来测试练习。导入starter包就可以了。

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.go9</groupId>  <artifactId>go9</artifactId>  <version>1.0.0-SNAPSHOT</version>  </parent>  <artifactId>go9-search</artifactId>  <properties>  <elasticsearch.version>6.5.4</elasticsearch.version>  </properties>  <dependencies>  <dependency>  <groupId>com.cjl</groupId>  <artifactId>elasticsearch-spring-boot-starter</artifactId>  <version>1.2.0.RELEASE</version>  </dependency>  </dependencies>  </project> |

配置yml

|  |
| --- |
| spring:  application:  name: go9-search-service  elasticsearch:  cluster-nodes: 192.168.57.20:9300  server:  port: 9999  logging:  level:  root: info  #com.go9.goods.mapper: debug  # 默认日志文件名  # file: log  # 默认日志路径  # path: ./log  # logback.xml路径，默认为classpath:logback.xml  # config: ./logback.xml |

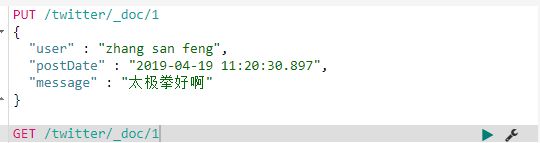
## 启动类

|  |
| --- |
| **package** com.go9.search;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **public** **class** SearchApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SearchApplication.**class**, args);  }  } |

## 测试代码

|  |
| --- |
| package com.cjl.es.api;  import static org.junit.Assert.\*;  **import org.elasticsearch.client.Client;**  import org.junit.Test;  import org.junit.runner.RunWith;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.test.context.junit4.SpringRunner;  import com.cjl.elasticsearch.spring.boot.autoconfig.ElasticSearchAutoConfig;  **@RunWith(SpringRunner.class)**  **@SpringBootTest(classes= {ElasticSearchAutoConfig.class})**  **public class ApiTest** {  **@Autowired**  **private Client client;**  @Test  public void testIndex() {    }  } |

## Index



|  |
| --- |
| @Test  **public** **void** testIndex() **throws** IOException {  System.***out***.println("client already loaded..........");  IndexResponse indexResponse = client.prepareIndex("twitter", "\_doc", "2")  .setSource(XContentFactory.*jsonBuilder*().startObject()  .field("user", "zhang wu ji")  .field("postDate", **new** Date())  .field("message","明教教主。。。。。。。。。")  .endObject()  ).get();  Result result = indexResponse.getResult();  System.***out***.println(result.getLowercase());  } |

## Update

|  |
| --- |
| @Test  **public** **void** testUpdate() **throws** IOException {  UpdateResponse updateResponse = client.prepareUpdate("twitter", "\_doc", "2")  .setDoc(XContentFactory.*jsonBuilder*().startObject().field("user","曾阿牛").endObject())  .get();  Result result = updateResponse.getResult();  System.***out***.println(result.getLowercase());  } |

## Get

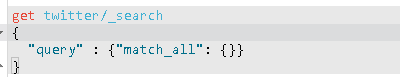


|  |
| --- |
| @Test  **public** **void** testGet() **throws** IOException {  GetResponse response = client.prepareGet("twitter", "\_doc", "2").get();  String sourceAsString = response.getSourceAsString();  System.***out***.println(sourceAsString);  } |

## Delete

|  |
| --- |
| @Test  **public** **void** testDelete() **throws** IOException {  DeleteResponse deleteResponse = client.prepareDelete("twitter", "\_doc", "2")  .get();  Result result = deleteResponse.getResult();  System.***out***.println(result.getLowercase());  } |

## Search



查询模板

|  |
| --- |
| SearchResponse response = client.prepareSearch("index1", "index2")  .setTypes("type1", "type2")  .setSearchType(SearchType.DFS\_QUERY\_THEN\_FETCH)  .setQuery(QueryBuilders.termQuery("multi", "test")) // Query  .setPostFilter(QueryBuilders.rangeQuery("age").from(12).to(18)) // Filter  .setFrom(0).setSize(60).setExplain(true)  .get(); |

以上条件都不是必须的，最简单的查询：返回所有记录

|  |
| --- |
| @Test  **public** **void** testSearchAll() {  SearchResponse searchResponse = client.prepareSearch("twitter").get();  SearchHits hits = searchResponse.getHits();  **long** totalHits = hits.getTotalHits();  *assertEquals*(5, totalHits);  } |

**分页查询**

|  |
| --- |
| @Test  public void testSearchPaged() {  //默认就是分页的，当然可以设置查找第几页的结果,例如下面查第2页的数据，每页10条数据  SearchResponse searchResponse = client.prepareSearch("bank")  .setFrom(10).setSize(10).get();  SearchHits hits = searchResponse.getHits();  int length = hits.getHits().length;  assertEquals(10, length);  } |

**遍历结果模板**

|  |
| --- |
| for (SearchHit hit : scrollResp.getHits().getHits()) {  //Handle the hit...  } |

## 设置搜索上限

|  |
| --- |
| @Test  public void testSearchLimit() {  //这里只是一个大概的限制，不能作为准确的值去使用，请注意  SearchResponse searchResponse = client.prepareSearch("bank").setTerminateAfter(200).get();  if (searchResponse.isTerminatedEarly()) {  SearchHits hits = searchResponse.getHits();  long totalHits = hits.getTotalHits();  //所以下面断言只是想看一下实际查询的数量  assertEquals(20, totalHits);  }  } |

|  |
| --- |
| SearchResponse sr = client.prepareSearch(INDEX)  .setTerminateAfter(1000)  .get();  if (sr.isTerminatedEarly()) {  // We finished early  } |

## 聚合

|  |
| --- |
| SearchResponse sr = node.client().prepareSearch()  .setQuery( /\* your query \*/ )  .addAggregation( /\* add an aggregation \*/ )  .execute().actionGet(); |

### 结构型

**按“州”分组，计算余额平均值，并按降序排序：**

|  |
| --- |
| SearchResponse sr = client.prepareSearch().addAggregation(AggregationBuilders.terms("group\_by\_state")  .field("state.keyword").size(30).order(BucketOrder.aggregation("avg\_balance", false)).subAggregation(AggregationBuilders.avg("avg\_balance").field("balance")))  .execute().actionGet();  Aggregation agg = sr.getAggregations().asList().get(0);  System.out.println(agg); |

### Metric(量化)

以下都是量化聚合的一些子类类型

Min Max Sum Avg Stats

|  |
| --- |
| SearchResponse sr = client.prepareSearch().addAggregation( AggregationBuilders  .min("min")  .field("balance"))  .execute().actionGet();  Min min = sr.getAggregations().get("min");  System.out.println(min.getValue());    sr = client.prepareSearch().addAggregation( AggregationBuilders  .stats("stats")  .field("balance"))  .execute().actionGet();  Stats stats = sr.getAggregations().get("stats");  System.out.println(stats.getMin());  System.out.println(stats.getMax());  System.out.println(stats.getAvg());  System.out.println(stats.getCount());  System.out.println(stats.getSum()); |

### Bucket(非量化)

**按性别分组统计人数，然后按降序排序**

|  |
| --- |
| SearchResponse sr = client.prepareSearch().addAggregation(  AggregationBuilders.terms("group\_by\_genders").field("gender.keyword").order(BucketOrder.count(false)))  .execute().actionGet();  Aggregation aggregation = sr.getAggregations().get("group\_by\_genders");  System.out.println(aggregation); |

## DSL

导入查询构建静态类

import static org.elasticsearch.index.query.QueryBuilders.\*;

### 搜索所有

matchAllQuery();

|  |
| --- |
| SearchResponse sr = client.prepareSearch("bank").setQuery(QueryBuilders.matchAllQuery()).get();  System.out.println(sr.getHits().totalHits); |

### 全文检索

#### match

|  |
| --- |
| SearchResponse sr = client.prepareSearch("bank").setQuery(QueryBuilders.matchQuery("address", "mill lane")).get();  System.out.println(sr.getHits().totalHits); |

#### multi match

|  |
| --- |
| multiMatchQuery( "kimchy elasticsearch", "user", "message"); |

### Terms查询

**精确查询，常用来查询结构化的数据，例如：数字，日期，枚举**

**例如：查询姓名是cjl的记录**

termQuery("name", "cjl");

**查询姓名是cjl或者 cuihua的文档**

termsQuery("name", "cjl", "cuihua");

**查询年龄在10（包含）到20（不包含）的文档**

rangeQuery("age").gte("10").lt("20");

# 分词器

分词器一般不直接使用，会集成在**分析器**里使用。ES是易用的，是因为有很多默认的配置，做全文检索时，背后的功臣是分析器。看个例子：

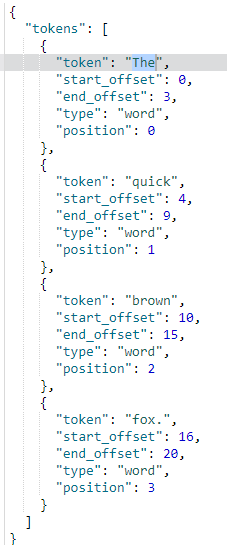
POST \_analyze

{

"analyzer": "whitespace",

"text": "The quick brown fox."

}



Es默认的分析器是**standard，不过可惜的是不能对中文进行分词。**它自带了一个中日韩语言分析器：**cjk**

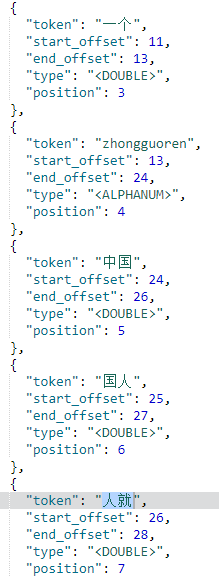
POST \_analyze

{

"analyzer": "cjk",

"text": "woshi我是yige一个zhongguoren中国人就"

}



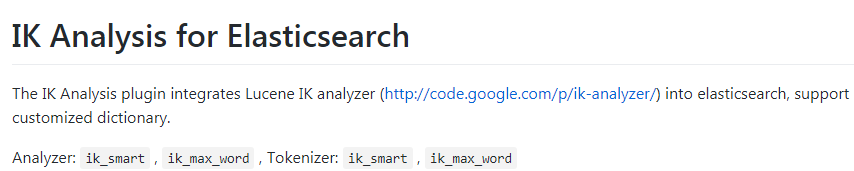
显然效果不理想。

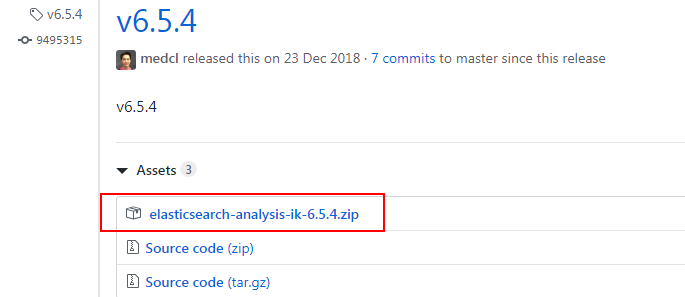
## 中文分词器

**国内常用中文分词器：Smartcn , IK analysis**

|  |
| --- |
| 中文分词一直是自然语言处理的一个痛处，早在2010年的时候，就曾经有项目涉及到相关的应用（Lunce构建全文搜索引擎），那时的痛，没想到8年后的今天依然存在，切分效果、扩展支持、业务应用等方面依然不甚理想。收费的版本不提了，原因自不必言表，开源版本中，发现之前曾经活跃的版本，大多已经没落（好几年没更新了），存活下来的寥寥无几。我是一个守旧的人，评估版本的选择有些保守，至少目前为止，只看1.0正式版本之后的版本，0.XX的不在考虑范围之内，用了一个周末的时间，对比了十多款的样子，个人感觉源于中科院ICTCLAS的**smartcn**和IKAnanlyzer效果还是不错的。 |

## elasticsearch-analysis-ik



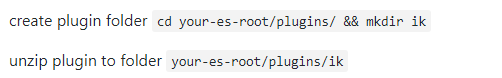


## 下载



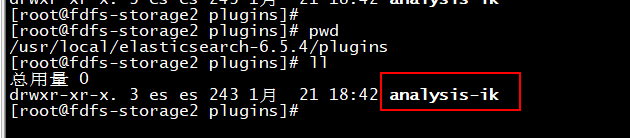
或者直接在资料包中获取，考虑到网络的问题。

## 安装



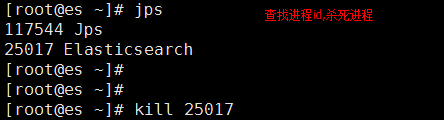


**unzip elasticsearch-analysis-ik-6.5.4.zip -d /usr/local/elasticsearch-6.5.4/plugins/**



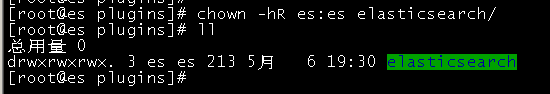
**注意：一定要有目录，不能直接解压到根目录**

## 重启



修改权限(如果不能执行，有可能是权限问题，修改即可)：





启动：



**小技巧：**

**默认插件安装时有版本检测，如果不符合会报异常，启动失败，可以修改插件的配置文件：plugin-descriptor.properties**

**将es 的版本修改为你实际安装的版本号**

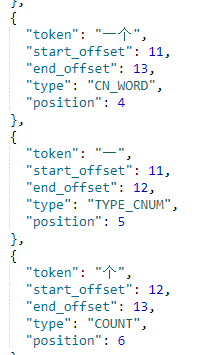
## 测试

|  |
| --- |
| ik\_max\_word: 会将文本做最细粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华人民共和国,中华人民,中华,华人,人民共和国,人民,人,民,共和国,共和,和,国国,国歌”，会穷尽各种可能的组合，适合 Term Query；  ik\_smart: 会做最粗粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华人民共和国,国歌”，适合 Phrase 查询。 |

分析器：Analyzer : ik\_smart**,**ik\_max\_word

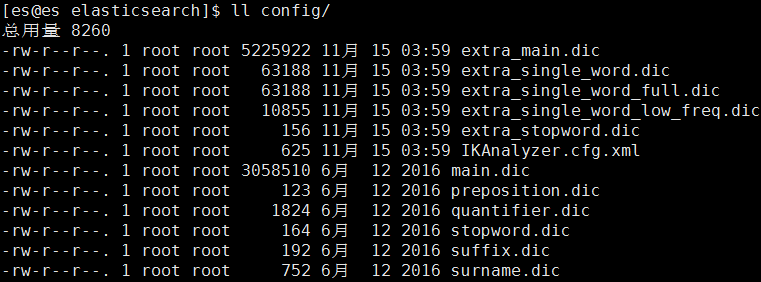
分词器：Tokenizer : ik\_smart**,**ik\_max\_word

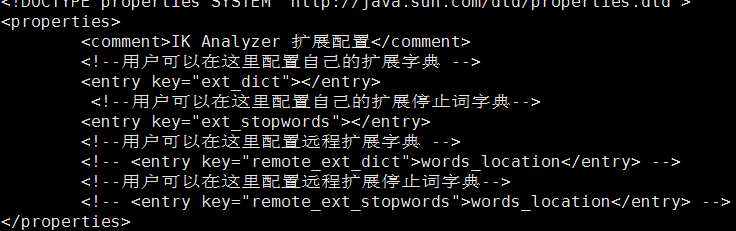
|  |
| --- |
| POST \_analyze  {  "analyzer": "ik\_max\_word",  "text": "woshi我是yige一个zhongguoren中国人就"  } |



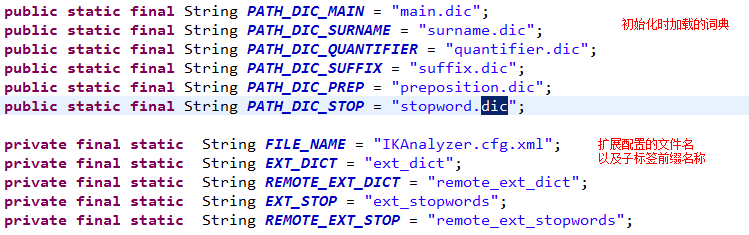
## 扩展词典

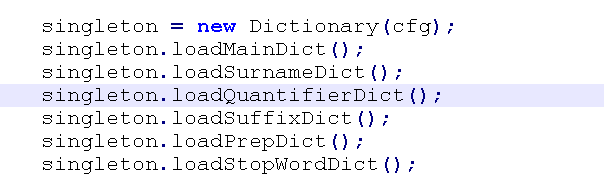
首先查看插件配置文件目录：





通过查看源码（Dictionary）：





直观的看到默认加载了哪些词典，以及加载的先后顺序。

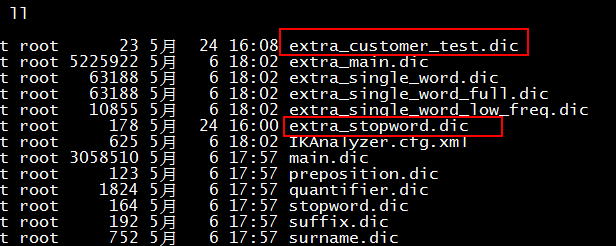
以及如何扩展字典：

默认目录下有几个写好的“扩展字典文件”**extra**\_\*.dic

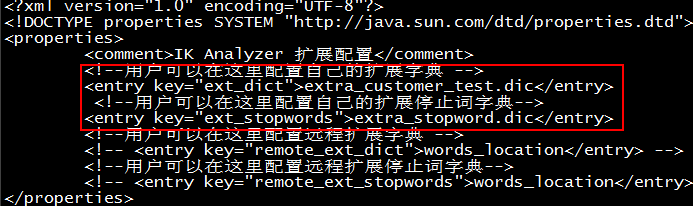
可以在词典里自由添加“字”，注意一行一个，还有就是打开文件时不要修改文件的默认编码“**UTF-8**”，换句话说不要使用会改变文件编码格式的编辑器打开文件！！例如windows自带的记事本。

## 扩展测试

我们**添加**了一个扩展词典，使用原来的**停词**字典，进行测试



**config/IKAnalyzer.cfg.xml**



重新启动后测试

我们添加了“**蓝瘦香菇**”，停掉了“**之乎也**”



