# 0.常用热词词库的配置方式

## 0.1 采用IK 内置词库

优点：部署方便，不用额外指定其他词库位置

缺点：分词单一化，不能指定想分词的词条

## 0.2 IK 外置静态词库

优点：部署相对方便，可以通过编辑指定文件分词文件得到想要的词条

缺点：需要指定外部静态文件，每次需要手动编辑整个分词文件，然后放到指定的文件目录下，重启ES后才能生效

## 0.3 IK 远程词库

优点：通过指定一个静态文件代理服务器来设置IK分词的词库信息

缺点：需要手动编辑整个分词文件来进行词条的添加， IK源码中判断头信息Last-Modified ETag 标识来判断是否更新，有时不生效

结合上面的优缺点，决定采用Mysql作为外置热词词库，定时更新热词 和 停用词。

本文以 Elasticsearch 7.3.2 版本为例.

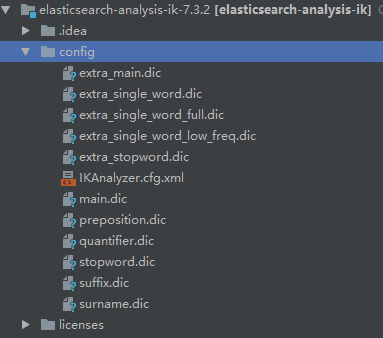
# 1.下载源码

## 1.1下载IK

下载合适的ElasticSearch对应版本的IK分词器：<https://github.com/medcl/elasticsearch-analysis-ik/archive/v7.3.2.zip>

## 1.2 配置文件说明

我们来查看它config文件夹下的文件：



分析IKAnalyzer.cfg.xml 配置文件：

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd"**>**
3. **<properties>**
4. **<comment>**IK Analyzer 扩展配置**</comment>**
5. <!--用户可以在这里配置自己的扩展字典 -->
6. **<entry** key="ext\_dict"**></entry>**
7. <!--用户可以在这里配置自己的扩展停止词字典-->
8. **<entry** key="ext\_stopwords"**></entry>**
9. <!--用户可以在这里配置远程扩展字典 -->
10. <!-- <entry key="remote\_ext\_dict">words\_location</entry> -->
11. <!--用户可以在这里配置远程扩展停止词字典-->
12. <!-- <entry key="remote\_ext\_stopwords">words\_location</entry> -->
13. **</properties>**

ext\_dict：对应的扩展热词词典的位置，多个热词文件之间使用分号来进行间隔

ext\_stopwords:对应扩展停用词词典位置，多个之间用分号进行间隔

remote\_ext\_dict：远程扩展热词位置 如：https://xxx.xxx.xxx.xxx/ext\_hot.txt

remote\_ext\_stopwords：远程扩展停用词位置 如：<https://xxx.xxx.xxx.xxx/ext_stop.txt>

# 2. 修改源码

## 2.1 Pom修改

在pom文件中, 修改<elasticsearch.version>7.3.2</elasticsearch.version>的依赖与使用的es版本调整一致, 默认下载的源码中为 7.0.0, 需要进行修改, 否则部署到服务器上, 加载IK插件时会报错

## 2.2 类修改

org.wltea.analyzer.dic.Dictionary单例类的初始化方法initial，在这里需要创建一个我们自定义的线程，并且启动它:

如下 第20行所示

1. /\*\*
2. \* 词典初始化 由于IK Analyzer的词典采用Dictionary类的静态方法进行词典初始化
3. \* 只有当Dictionary类被实际调用时，才会开始载入词典， 这将延长首次分词操作的时间 该方法提供了一个在应用加载阶段就初始化字典的手段
4. \*
5. \* @return Dictionary
6. \*/
7. **public** **static** **synchronized** **void** initial(Configuration cfg) {
8. **if** (singleton == **null**) {
9. **synchronized** (Dictionary.**class**) {
10. **if** (singleton == **null**) {
12. singleton = **new** Dictionary(cfg);
13. singleton.loadMainDict();
14. singleton.loadSurnameDict();
15. singleton.loadQuantifierDict();
16. singleton.loadSuffixDict();
17. singleton.loadPrepDict();
18. singleton.loadStopWordDict();
20. **new** Thread(**new** MySQLDictReloadThread()).start();
22. **if**(cfg.isEnableRemoteDict()){
23. // 建立监控线程
24. **for** (String location : singleton.getRemoteExtDictionarys()) {
25. // 10 秒是初始延迟可以修改的 60是间隔时间 单位秒
26. pool.scheduleAtFixedRate(**new** Monitor(location), 10, 60, TimeUnit.SECONDS);
27. }
28. **for** (String location : singleton.getRemoteExtStopWordDictionarys()) {
29. pool.scheduleAtFixedRate(**new** Monitor(location), 10, 60, TimeUnit.SECONDS);
30. }
31. }
32. }
33. }
34. }
35. }

MySQLDictReloadThread类：就是死循环，不断调用Dictionary.getSingleton().reLoadMainDict()，去重新加载词典, 如下所示:

1. **package** org.wltea.analyzer.dic;
3. **import** org.apache.logging.log4j.Logger;
4. **import** org.wltea.analyzer.help.ESPluginLoggerFactory;
6. /\*\*
7. \* MySQLDictReloadThread Runnable实现类，去执行Dictionary.reLoadSQLDict() 加载热词
8. \*
9. \* created by yangyu on 2019-09-26
10. \*/
11. **public** **class** MySQLDictReloadThread **implements** Runnable{
13. **private** **static** **final** Logger logger = ESPluginLoggerFactory.getLogger(MySQLDictReloadThread.**class**.getName());
15. @Override
16. **public** **void** run() {
17. **while** (**true**) {
18. logger.info("[==========]reloading hot dict from mysql......");
19. Dictionary.getSingleton().reLoadMainDict();
20. }
21. }
22. }

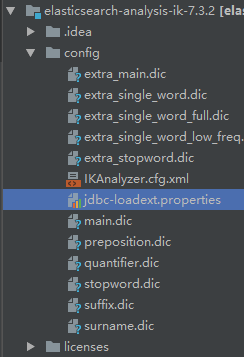
Dictionary.reLoadMainDict()中，在如下第13行去加载MySQL中的热词

1. **void** reLoadMainDict() {
2. logger.info("重新加载词典...");
3. // 新开一个实例加载词典，减少加载过程对当前词典使用的影响
4. Dictionary tmpDict = **new** Dictionary(configuration);
5. tmpDict.configuration = getSingleton().configuration;
6. tmpDict.loadMainDict();
7. tmpDict.loadStopWordDict();
8. \_MainDict = tmpDict.\_MainDict;
9. \_StopWords = tmpDict.\_StopWords;
11. logger.info("Start ReloadMysqlDict start...");
12. // 从mysql加载词典
13. **this**.reLoadMySQLDict();
14. logger.info("Start ReloadMysqlDict end...");
16. logger.info("重新加载词典完毕...");
17. }

reLoadMySQLDict()方法如下:

1. **private** **void** reLoadMySQLDict() {
2. Connection conn = **null**;
3. Statement stmt = **null**;
4. ResultSet rs = **null**;
5. **try** {
6. Path file = PathUtils.get(getDictRoot(), "jdbc-loadext.properties");
7. props.load(**new** FileInputStream(file.toFile()));
9. logger.info("Dictionary reLoadMySQLDict jdbc-reload.properties...");
10. **for**(Object key : props.keySet()) {
11. logger.info(key + "=" + props.getProperty(String.valueOf(key)));
12. }
14. Class.forName(props.getProperty("jdbc.driver"));
16. conn = DriverManager.getConnection(props.getProperty("jdbc.url"),props.getProperty("jdbc.user"),props.getProperty("jdbc.password"));
18. logger.info("query hot dict from mysql, " + props.getProperty("jdbc.reload.sql") + "......");
19. stmt = conn.createStatement();
20. rs = stmt.executeQuery(props.getProperty("jdbc.reload.sql"));
21. **while**(rs.next()) {
22. String theWord = rs.getString("word");
23. **if** (**null** == theWord) {
24. **continue**;
25. }
26. logger.info("hot word from mysql: " + theWord);
27. \_MainDict.fillSegment(theWord.trim().toCharArray());
28. }
30. logger.info("query hot stopword dict from mysql, " + props.getProperty("jdbc.reload.stopword.sql") + "......");
31. stmt = conn.createStatement();
32. rs = stmt.executeQuery(props.getProperty("jdbc.reload.stopword.sql"));
33. **while**(rs.next()) {
34. String theWord = rs.getString("stop\_word");
35. **if** (**null** == theWord) {
36. **continue**;
37. }
38. logger.info("hot stopword from mysql: " + theWord);
39. \_StopWords.fillSegment(theWord.trim().toCharArray());
40. }
42. Thread.sleep(Long.parseLong(String.valueOf(props.get("jdbc.reload.interval"))));
43. } **catch** (Exception ex) {
44. logger.error("Dictionary reLoadMySQLDict erorr", ex);
45. } **finally** {
46. **if**(rs != **null**) {
47. **try** {
48. rs.close();
49. } **catch** (SQLException e) {
50. logger.error("error", e);
51. }
52. }
53. **if**(stmt != **null**) {
54. **try** {
55. stmt.close();
56. } **catch** (SQLException e) {
57. logger.error("error", e);
58. }
59. }
60. **if**(conn != **null**) {
61. **try** {
62. conn.close();
63. } **catch** (SQLException e) {
64. logger.error("error", e);
65. }
66. }
67. }
68. }

## 2.3 Jdbc配置



jdbc示例内容如下:

1. jdbcjdbc.url=jdbc:mysql://jrdsn3d3c6w5883.mysql.rds.aliyuncs.com:3306/erp\_ic?useUnicode=true&characterEncoding=utf-8
2. jdbc.driver=com.mysql.jdbc.Driver
3. jdbc.user=demoadmin
4. jdbc.password=demo\_admin
5. jdbc.reload.sql=select word from hot\_words
6. jdbc.reload.stopword.sql=select stop\_word from hot\_stopwords
7. jdbc.reload.interval=60000

## 2.4 修改Java安全权限

在实验过程中会遇到如下错误:

1. Caused by: java.security.AccessControlException: access denied ("java.util.PropertyPermission" "\*" "read,write")

这一句说明是java的安全管理机制限制了程序的读写操作，出现类似这个错误, 可以修改 源码中 src/main/resources/plugin-security.policy 配置文件, 示例如下 :

1. grant {
2. // needed because of the hot reload functionality
3. permission java.net.SocketPermission "\*", "connect,resolve";
4. permission java.util.PropertyPermission "\*", "read,write";
5. permission java.lang.RuntimePermission "getClassLoader";
6. permission javax.management.MBeanServerPermission "createMBeanServer";
7. };

然后在VM参数中写入下面这句话，指定程序的policy文件

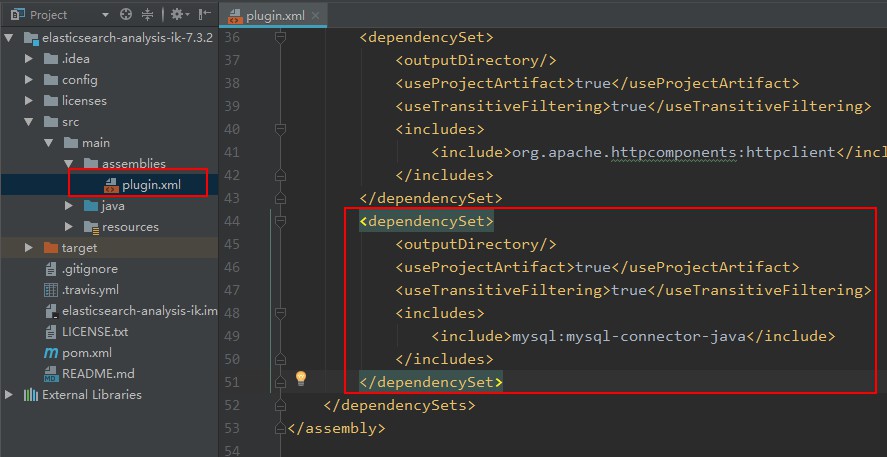
1. -Djava.security.policy= -Djava.security.policy=/data/elasticsearch-7.3.2/plugins/ik/plugin-security.policy

# 3.打包

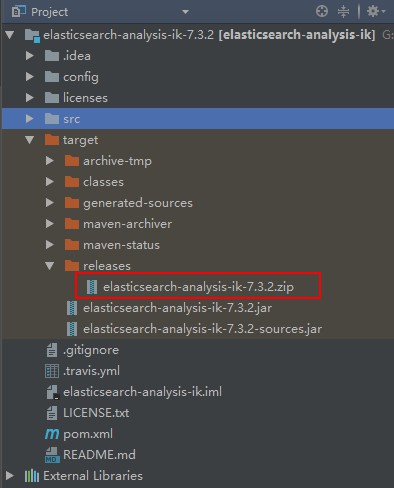
因为我们链接的是mysql数据库，所以maven项目要引入mysql驱动：

1. **<dependency>**
2. **<groupId>**mysql**</groupId>**
3. **<artifactId>**mysql-connector-java**</artifactId>**
4. **<version>**5.1.22**</version>**
5. **</dependency>**

仅仅这样还不够，还需要修改plugin.xml文件，将上述Mysql驱动包添加进去



打包结果如下 :

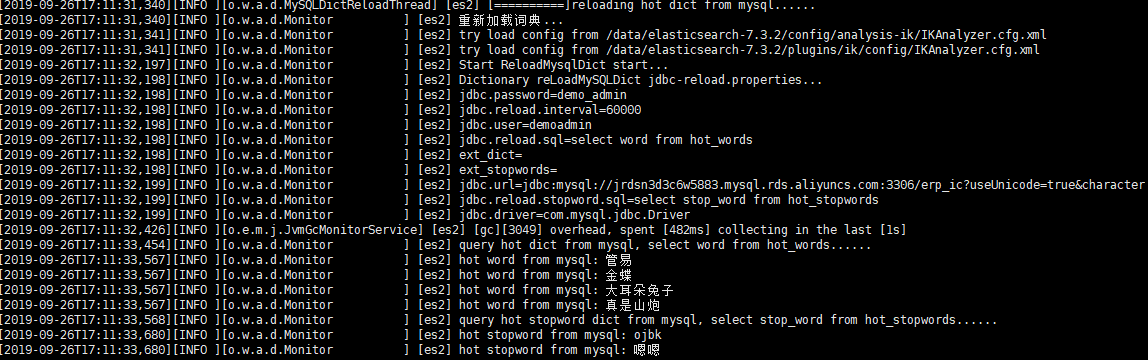


# 4.测试结果

在Mysql中插入热词:



在Elasticsearch启动过程中, 会有如下日志:



在修改IK之前, “管易易金蝶蝶” 分词结果如下:



在修改IK之后, “管易易金蝶蝶” 分词结果如下, “管易”已被作为一个热词进行处理



由于是动态添加热词库, 分词效果会随着Mysql分词库的变化而变化.