文档：4、Java API整合ElasticSearch与分词器...

链接：http://note.youdao.com/noteshare?id=33e69d60b733dc3d2266d0c28331aae2&sub=F0EC279F090B4130A3961B9C1A339

一、Java API操作ES（上）

相关依赖：

```
<dependencies>
 <!-- ES的高阶的客户端API -->
 <dependency>
 <groupId>org.elasticsearch.client</groupId>
 <artifactId>elasticsearch-rest-high-level-client</artifactId>
 <version>7.6.1</version>
 </dependency>
 <dependency>
 <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.11.1</version>
  </dependency>
  <!-- 阿里巴巴出品的一款将Java对象转换为JSON、将JSON转换为Java对象的库 -->
  <dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.62</version>
  </dependency>
  <dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
  <dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>6.14.3</version>
  <scope>test</scope>
  </dependency>

</dependencies>
```

使用JavaAPI来操作ES集群

初始化连接

使用的是RestHighLevelClient去连接ES集群，后续操作ES中的数据

```
private RestHighLevelClient restHighLevelClient;

public JobFullTextServiceImpl() {
// 建立与ES的连接
// 1. 使用RestHighLevelClient构建客户端连接。
// 2. 基于RestClient.builder方法来构建RestClientBuilder
// 3. 用HttpHost来添加ES的节点
RestClientBuilder restClientBuilder = RestClient.builder(
new HttpHost("192.168.21.130", 9200, "http")
 , new HttpHost("192.168.21.131", 9200, "http")
 , new HttpHost("192.168.21.132", 9200, "http"));
 restHighLevelClient = new RestHighLevelClient(restClientBuilder);
 }
```

添加职位数据到ES中

* 使用IndexRequest对象来描述请求

    * 可以设置请求的参数：设置ID、并设置传输ES的数据——注意因为ES都是使用JSON（DSL）来去操作数据的，所以需要使用一个FastJSON的库来将对象转换为JSON字符串进行操作

```
@Override
public void add(JobDetail jobDetail) throws IOException {
 //1. 构建IndexRequest对象，用来描述ES发起请求的数据。
 IndexRequest indexRequest = new IndexRequest(JOB_IDX);

 //2. 设置文档ID。
 indexRequest.id(jobDetail.getId() + "");

 //3. 使用FastJSON将实体类对象转换为JSON。
 String json = JSONObject.toJSONString(jobDetail);

 //4. 使用IndexRequest.source方法设置文档数据，并设置请求的数据为JSON格式。
 indexRequest.source(json, XContentType.JSON);

 //5. 使用ES High level client调用index方法发起请求，将一个文档添加到索引中。
 restHighLevelClient.index(indexRequest, RequestOptions.DEFAULT);
}
```

查询/删除/搜索/分页

```

* 新增：IndexRequest
* 更新：UpdateRequest
* 删除：DeleteRequest
* 根据ID获取：GetRequest
* 关键字检索：SearchRequest
```

```
@Override
public JobDetail findById(long id) throws IOException {
 // 1. 构建GetRequest请求。
 GetRequest getRequest = new GetRequest(JOB_IDX, id + "");

 // 2. 使用RestHighLevelClient.get发送GetRequest请求，并获取到ES服务器的响应。
 GetResponse getResponse = restHighLevelClient.get(getRequest, RequestOptions.DEFAULT);

 // 3. 将ES响应的数据转换为JSON字符串
 String json = getResponse.getSourceAsString();

 // 4. 并使用FastJSON将JSON字符串转换为JobDetail类对象
 JobDetail jobDetail = JSONObject.parseObject(json, JobDetail.class);

 // 5. 记得：单独设置ID
 jobDetail.setId(id);

 return jobDetail;
}
```

```
@Override
public void update(JobDetail jobDetail) throws IOException {
 // 1. 判断对应ID的文档是否存在
```

```
    // a) 构建GetRequest
    GetRequest getRequest = new GetRequest(JOB_IDX, jobDetail.getId() + "");

    // b) 执行client的exists方法，发起请求，判断是否存在
    boolean exists = restHighLevelClient.exists(getRequest, RequestOptions.DEFAULT);

    if(exists) {
    // 2. 构建UpdateRequest请求
    UpdateRequest updateRequest = new UpdateRequest(JOB_IDX, jobDetail.getId() + "");

    // 3. 设置UpdateRequest的文档，并配置为JSON格式
    updateRequest.doc(JSONObject.toJSONString(jobDetail), XContentType.JSON);

    // 4. 执行client发起update请求
    restHighLevelClient.update(updateRequest, RequestOptions.DEFAULT);
    }
}
```

```
@Override
public void deleteById(long id) throws IOException {
    // 1. 构建delete请求
    DeleteRequest deleteRequest = new DeleteRequest(JOB_IDX, id + "");

    // 2. 使用RestHighLevelClient执行delete请求
    restHighLevelClient.delete(deleteRequest, RequestOptions.DEFAULT);

}
```

```
@Override
public List<JobDetail> searchByKeywords(String keywords) throws IOException {
    // 1.构建SearchRequest检索请求
    // 专门用来进行全文检索、关键字检索的API
    SearchRequest searchRequest = new SearchRequest(JOB_IDX);

    // 2.创建一个SearchSourceBuilder专门用于构建查询条件
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();

    // 3.使用QueryBuilders.multiMatchQuery构建一个查询条件（搜索title、jd），并配置到SearchSourceBuilder
    MultiMatchQueryBuilder multiMatchQueryBuilder = QueryBuilders.multiMatchQuery(keywords, "title", "jd");

    // 将查询条件设置到查询请求构建器中
    searchSourceBuilder.query(multiMatchQueryBuilder);

    // 4.调用SearchRequest.source将查询条件设置到检索请求
    searchRequest.source(searchSourceBuilder);

    // 5.执行RestHighLevelClient.search发起请求
    SearchResponse searchResponse = restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
    SearchHit[] hitArray = searchResponse.getHits().getHits();

    // 6.遍历结果
    ArrayList<JobDetail> jobDetailArrayList = new ArrayList<>();

    for (SearchHit documentFields : hitArray) {
    // 1)获取命中的结果
    String json = documentFields.getSourceAsString();

    // 2)将JSON字符串转换为对象
    JobDetail jobDetail = JSONObject.parseObject(json, JobDetail.class);

    // 3)使用SearchHit.getId设置文档ID
```

```
 jobDetail.setId(Long.parseLong(documentFields.getId()));

 jobDetailArrayList.add(jobDetail);
 }

 return jobDetailArrayList;
}
```

```
@Override
public Map<String, Object> searchByPage(String keywords, int pageNum, int pageSize) throws IOException {
 // 1.构建SearchRequest检索请求
 // 专门用来进行全文检索、关键字检索的API
 SearchRequest searchRequest = new SearchRequest(JOB_IDX);

 // 2.创建一个SearchSourceBuilder专门用于构建查询条件
 SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();

 // 3.使用QueryBuilders.multiMatchQuery构建一个查询条件（搜索title、jd），并配置到SearchSourceBuilder
 MultiMatchQueryBuilder multiMatchQueryBuilder = QueryBuilders.multiMatchQuery(keywords, "title", "jd");

 // 将查询条件设置到查询请求构建器中
 searchSourceBuilder.query(multiMatchQueryBuilder);

 // 每页显示多少条
 searchSourceBuilder.size(pageSize);
 // 设置从第几条开始查询
 searchSourceBuilder.from((pageNum - 1) * pageSize);

 // 4.调用SearchRequest.source将查询条件设置到检索请求
 searchRequest.source(searchSourceBuilder);

 // 5.执行RestHighLevelClient.search发起请求
 SearchResponse searchResponse = restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
 SearchHit[] hitArray = searchResponse.getHits().getHits();

 // 6.遍历结果
 ArrayList<JobDetail> jobDetailArrayList = new ArrayList<>();

 for (SearchHit documentFields : hitArray) {
 // 1)获取命中的结果
 String json = documentFields.getSourceAsString();

 // 2)将JSON字符串转换为对象
 JobDetail jobDetail = JSONObject.parseObject(json, JobDetail.class);

 // 3)使用SearchHit.getId设置文档ID
 jobDetail.setId(Long.parseLong(documentFields.getId()));

 jobDetailArrayList.add(jobDetail);
 }

 // 8. 将结果封装到Map结构中（带有分页信息）
 // a) total -> 使用SearchHits.getTotalHits().value获取到所有的记录数
 // b) content -> 当前分页中的数据
 long totalNum = searchResponse.getHits().getTotalHits().value;
 HashMap hashMap = new HashMap();
 hashMap.put("total", totalNum);
 hashMap.put("content", jobDetailArrayList);


 return hashMap;
```

```
54 }
```

高亮查询

1. 配置高亮选项

```
// 设置高亮
HighlightBuilder highlightBuilder = new HighlightBuilder();
highlightBuilder.field("title");
highlightBuilder.field("jd");
highlightBuilder.preTags("<font color='red'>");
highlightBuilder.postTags("</font>");

```

2. 需要将高亮的字段拼接在一起，设置到实体类中

```
// 设置高亮的一些文本到实体类中
// 封装了高亮
Map<String, HighlightField> highlightFieldMap = documentFields.getHighlightFields();
HighlightField titleHL = highlightFieldMap.get("title");
HighlightField jdHL = highlightFieldMap.get("jd");

if(titleHL != null) {
 // 获取指定字段的高亮片段
 Text[] fragments = titleHL.getFragments();
 // 将这些高亮片段拼接成一个完整的高亮字段
 StringBuilder builder = new StringBuilder();
 for(Text text : fragments) {
 builder.append(text);
 }
 // 设置到实体类中
 jobDetail.setTitle(builder.toString());
}

if(jdHL != null) {
 // 获取指定字段的高亮片段
 Text[] fragments = jdHL.getFragments();
 // 将这些高亮片段拼接成一个完整的高亮字段
 StringBuilder builder = new StringBuilder();
 for(Text text : fragments) {
 builder.append(text);
 }
 // 设置到实体类中
 jobDetail.setJd(builder.toString());
}
```

二、分词器工作流程

1、切分词语，normalization

给你一段句子，然后将这段句子拆分成一个一个的单个的单词，同时对每个单词进行normalization（时态转换，单复数转换），分词器

recall，召回率：搜索的时候，增加能够搜索到的结果的数量

```
character filter：在一段文本进行分词之前，先进行预处理，比如说最常见的就是，过滤html标签（<span>hello<span> --> hello），&
--> and（I&you --> I and you）
```

```
tokenizer：分词，hello you and me --> hello，you，and，me

token filter：lowercase，stop word，synonymom，liked --> like，Tom --> tom，a/the/an --> 干掉，small --> little
```

一个分词器，很重要，将一段文本进行各种处理，最后处理好的结果才会拿去建立倒排索引

2、内置分词器的介绍

```
Set the shape to semi-transparent by calling set_trans(5)

standard analyzer：set，the，shape，to，semi，transparent，by，calling，set_trans，5（默认的是standard）

simple analyzer：set，the，shape，to，semi，transparent，by，calling，set，trans

whitespace analyzer：Set，the，shape，to，semi-transparent，by，calling，set_trans(5)

stop analyzer:移除停用词，比如a the it等等

 测试：
 POST _analyze
 {
 "analyzer":"standard",
 "text":"Set the shape to semi-transparent by calling set_trans(5)"
 }
```

3、定制分词器

3.1 修改分词器的设置

启用english停用词token filter

```
PUT /my_index
{
  "settings": {
  "analysis": {
  "analyzer": {
  "es_std": {
  "type": "standard",
  "stopwords": "_english_"
  }
   }
   }
   }
}

GET /my_index/_analyze
{
  "analyzer": "standard",
  "text": "a dog is in the house"
}

GET /my_index/_analyze
{
  "analyzer": "es_std",
```

```
  "text":"a dog is in the house"
}
```

3.2 ik分词器详解

ik配置文件地址：es/plugins/ik/config目录

IKAnalyzer.cfg.xml：用来配置自定义词库
main.dic：ik原生内置的中文词库，总共有27万多条，只要是这些单词，都会被分在一起
quantifier.dic：放了一些单位相关的词
suffix.dic：放了一些后缀
surname.dic：中国的姓氏
stopword.dic：英文停用词

ik原生最重要的两个配置文件

main.dic：包含了原生的中文词语，会按照这个里面的词语去分词
stopword.dic：包含了英文的停用词

停用词，stopword

a the and at but

一般，像停用词，会在分词的时候，直接被干掉，不会建立在倒排索引中

3.3 IK分词器自定义词库

（1）自己建立词库：每年都会涌现一些特殊的流行词，网红，蓝瘦香菇，喊麦，鬼畜，一般不会在ik的原生词典里

自己补充自己的最新的词语，到ik的词库里面去

IKAnalyzer.cfg.xml：ext_dict，custom/mydict.dic

补充自己的词语，然后需要重启es，才能生效

（2）自己建立停用词库：比如了，的，啥，么，我们可能并不想去建立索引，让人家搜索

custom/ext_stopword.dic，已经有了常用的中文停用词，可以补充自己的停用词，然后重启es

```
IK分词器源码下载：https://github.com/medcl/elasticsearch-analysis-ik/tree
```

三、IK热更新

3.1、IK分词器源码下载：https://github.com/medcl/elasticsearch-analysis-ik

本案例以ES7.6.1和MySql数据库5.7为例进行配置；

修改源码步骤

3.2、修改maven依赖es版本号

使用工具打开IK源码后，打开pom.xml文件，修改elasticsearch版本号为7.8.0

```
<elasticsearch.version>7.8.0</elasticsearch.version>
```

3.3、引入MySql驱动到项目中

```
<dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <version>5.1.38</version>
```

3.4、开始修改源码

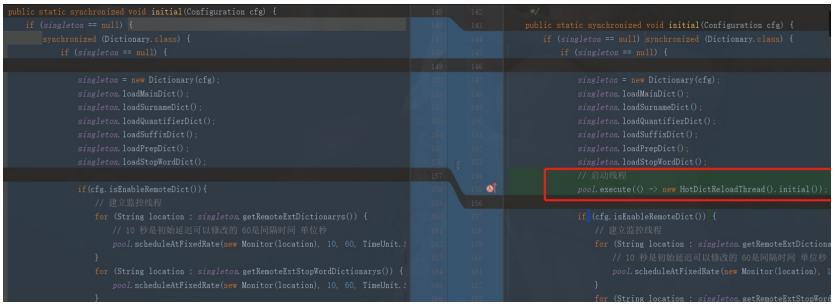
在项目中找到Dictionary类，找到Dictionary单例类的初始化方法**initial方法**，在初始化方法中我们起一个线程，用来执行远程词库的热更新，再修改之前，我们在Dictionary类同目录下新建一个类HotDictReloadThread，代码如下：

```
public class HotDictReloadThread {
 private static final Logger log = ESPluginLoggerFactory.getLogger(HotDictReloadThread.class.getName());
 public void initial(){
 while (true) {
 log.info("正在调用HotDictReloadThread...");
 Dictionary.getSingleton().reLoadMainDict();
 }
 }
```

上述代码的含义为：获取词典单子实例，并执行它的**reLoadMainDict方法**；

完成上述操作后，我们就来开始修改**initial方法**，改动如下图，创建上面新建的类并调用它的initial方法，从而执行Dictionary类的**reLoadMainDict方法**；改动代码如下，在字典实例初始化完成后新起一个线程来执行字典的热更新操作；

```
pool.execute(() -> new HotDictReloadThread().initial());
```



跟着程序一步步走下去，找到Dictionary类的reLoadMainDict方法，可以看到在方面里面，有2个方法**tmpDict.loadMainDict()和tmpDict.loadStopWordDict()**，分别维护的是扩展词库和停用词库，一块先看一下对扩展词库的维护；

在方法tmpDict.loadMainDict()中，我们在最后一行加载远程自定义词库后面新增一个方法this.**loadMySQLExtDict()**，用于加载MySql词库，在加载MySql词库之前，我们需先准备一下MySql相关的配置以及sql语句；在数据库中新建一张表，用户维护扩展词和停用词，表结构如下

```sql
CREATE TABLE `es_lexicon` (
 `lexicon_id` bigint(8) NOT NULL AUTO_INCREMENT COMMENT '词库id',
 `lexicon_text` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL COMMENT '词条关键词',
 `lexicon_type` int(1) NOT NULL DEFAULT 0 COMMENT '0扩展词库 1停用词库',
 `lexicon_status` int(1) NOT NULL DEFAULT 0 COMMENT '词条状态 0正常 1暂停使用',
 `del_flag` int(1) NOT NULL DEFAULT 0 COMMENT '作废标志 0正常 1作废',
 `create_time` datetime(0) NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
 PRIMARY KEY (`lexicon_id`) USING BTREE
```

然后我们在项目的根路径的config目录下新建配置文件jdbc-reload.properties，内容如下

```
# 数据库地址
jdbc.url=jdbc:mysql://127.0.0.1:3306/test?serverTimezone=GMT&autoReconnect=true&useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useAffectedRows=true&useSSL=false
# 数据库用户名
jdbc.user=root
# 数据库密码
jdbc.password=123456
# 数据库查询扩展词库sql语句
jdbc.reload.sql=select gel.lexicon_text as word from es_lexicon gel where gel.lexicon_type = 0 and gel.lexicon_status = 0 and gel.del_flag = 0 order by gel.lexicon_id desc
# 数据库查询停用词sql语句
jdbc.reload.stopword.sql=select gel.lexicon_text as word from ges_lexicon gel where gel.lexicon_type = 1 and gel.lexicon_status = 0 and gel.del_flag = 0 order by gel.lexicon_id desc
# 数据库查询间隔时间 每隔10秒请求一次
```

完成了这些基础配置之后，我们再一同看看关于同步MySql词库的方法loadMySQLExtDict()；代码较长，粘贴如下

```
 /**
 * 从MySql中加载动态词库
 */
private void loadMySQLExtDict() {
 Connection conn = null;
 Statement stmt = null;
 ResultSet rs = null;
 try {
 Path file = PathUtils.get(getDictRoot(), "jdbc-reload.properties");
 props.load(new FileInputStream(file.toFile()));

 logger.info("[==========]jdbc-reload.properties");
 for(Object key : props.keySet()) {
 logger.info("[==========]" + key + "=" + props.getProperty(String.valueOf(key)));
 }

 logger.info("[==========]query hot dict from mysql, " + props.getProperty("jdbc.reload.sql") + "......");
// Class.forName(props.getProperty("jdbc.className"));
 conn = DriverManager.getConnection(
 props.getProperty("jdbc.url"),
 props.getProperty("jdbc.user"),
 props.getProperty("jdbc.password"));
 stmt = conn.createStatement();
 rs = stmt.executeQuery(props.getProperty("jdbc.reload.sql"));

 while(rs.next()) {
 String theWord = rs.getString("word");
 logger.info("[==========]正在加载Mysql自定义IK扩展词库词条: " + theWord);
 _MainDict.fillSegment(theWord.trim().toCharArray());
 }

 Thread.sleep(Integer.valueOf(String.valueOf(props.get("jdbc.reload.interval"))) * 1000);
 } catch (Exception e) {
 logger.error("erorr", e);
```

```
35  } finally {
36  if(rs != null) {
37  try {
38  rs.close();
39  } catch (SQLException e) {
40  logger.error("error", e);
41  }
42  }
43  if(stmt != null) {
44  try {
45  stmt.close();
46  } catch (SQLException e) {
47  logger.error("error", e);
48  }
49  }
50  if(conn != null) {
51  try {
52  conn.close();
53  } catch (SQLException e) {
54  logger.error("error", e);
55  }
56  }
57  }
```

在上述代码中，通过加载配置文件，获取数据库连接，执行扩展词sql，将结果集添加到扩展词库中；

同理，同步MySql停用词的逻辑也是一样的，这里我直接把代码粘贴过来；停用词方法调用顺序为tmpDict.loadStopWordDict()，在方法后面，新增一个方法调用this.loadMySQLStopwordDict()，新方法中处理通用词逻辑，代码如下

```
1  /**
2  * 从MySql中加载远程停用词库
3  */
4  private void loadMySQLStopwordDict() {
5   Connection conn = null;
6   Statement stmt = null;
7   ResultSet rs = null;
8
9   try {
10   Path file = PathUtils.get(getDictRoot(), "jdbc-reload.properties");
11   props.load(new FileInputStream(file.toFile()));
12
13   logger.info("[==========]jdbc-reload.properties");
14   for(Object key : props.keySet()) {
15   logger.info("[==========]" + key + "=" + props.getProperty(String.valueOf(key)));
16   }
17   logger.info("[==========]query hot stopword dict from mysql, " + props.getProperty("jdbc.reload.stopword.sql") +
"......");
18  // Class.forName(props.getProperty("jdbc.className"));
19   conn = DriverManager.getConnection(
20   props.getProperty("jdbc.url"),
21   props.getProperty("jdbc.user"),
22   props.getProperty("jdbc.password"));
23   stmt = conn.createStatement();
24   rs = stmt.executeQuery(props.getProperty("jdbc.reload.stopword.sql"));
25
26   while(rs.next()) {
27   String theWord = rs.getString("word");
28   logger.info("[==========]正在加载Mysql自定义IK停用词库词条: " + theWord);
29   _StopWords.fillSegment(theWord.trim().toCharArray());
30   }
```

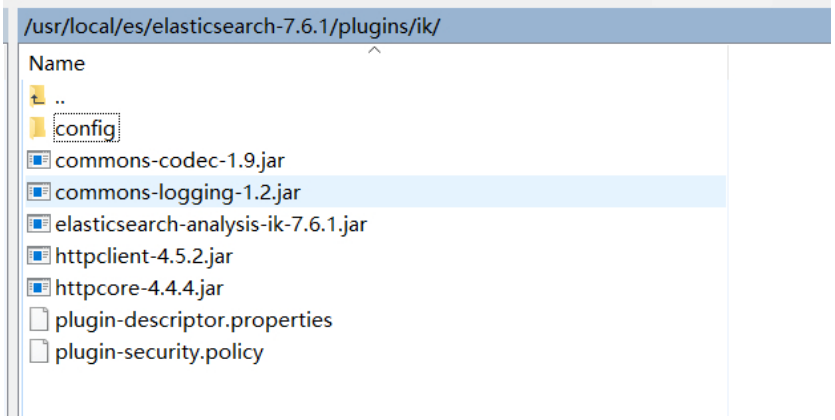
```
31  Thread.sleep(Integer.valueOf(String.valueOf(props.get("jdbc.reload.interval"))) * 1000);
32  } catch (Exception e) {
33  logger.error("erorr", e);
34  } finally {
35  if(rs != null) {
36  try {
37  rs.close();
38  } catch (SQLException e) {
39  logger.error("error", e);
40  }
41  }
42  if(stmt != null) {
43  try {
44  stmt.close();
45  } catch (SQLException e) {
46  logger.error("error", e);
47  }
48  }
49  if(conn != null) {
50  try {
51  conn.close();
52  } catch (SQLException e) {
53  logger.error("error", e);
54  }
55  }
56  }
```

完成这些，整体代码改造完毕；在上述代码中，有很多的地方是可以进一步优化的，比如扩展词和停用词的大量重复代码，以及读取本地配置文件项可以做到只读取一次等，这个大家可以自行优化；

完成了这些之后，我们就可以开始打包插件了；直接使用maven package命令进行打包，在target/releases/elasticsearch-analysis-ik-7.6.1.zip文件；

安装插件

完成上述步骤后，拿到elasticsearch-analysis-ik-7.8.0.zip插件，我们将其放在ES安装目录下的plugins目录下，新建一个ik文件夹，将其解压到ik文件夹下；目录结构大概如下



完成上述步骤后，我们就可以启动ES了，在启动过程中，可以看到关于IK热更新MySql词库相关的日志输出；在实际过程中，可能会报很多的异常，下面是我所遇到的一些问题以及解决方案；

常见问题

1、异常1：java.sql.SQLException: Column 'word' not found.

此异常是因为编写sql时，查询的数据库字段需要起别名为 word，修改一下sql即可解决这个问题；

2、异常2：Could not create connection to database server

此异常通常是因为引用的mysql驱动和mysql版本号不一致导致的，只需要替换成对应的版本号即可解决，另外，数据库连接我们不需要再额外的去配置显示加载，即不需要写 Class.forName(props.getProperty("jdbc.className"));

3、异常3：no suitable driver found for jdbc:mysql://...

此异常我们需要在环境的JDK安装目录的jre\lib\ext目录下添加mysql驱动mysql-connector-java.jar；比如我本地的是C:\Java\jdk_8u_231\jre\lib\ext 目录，服务器上是/usr/local/jdk/jdk1.8.0_181/jre/lib/ext

4、异常4：AccessControlException: access denied ("java.net.SocketPermission" "127.0.0.1:3306" "connect,resolve")

这个异常，我们修改jdk安装路径下的C:\Java\jdk_8u_231\jre\lib\security目录下的文件**java.policy**，在下面新增一行即可解决

```
permission java.net.SocketPermission "*", "connect,resolve";
```