# elasticsearch\_\_4\_\_java操作之QueryBuilders构建搜索Query

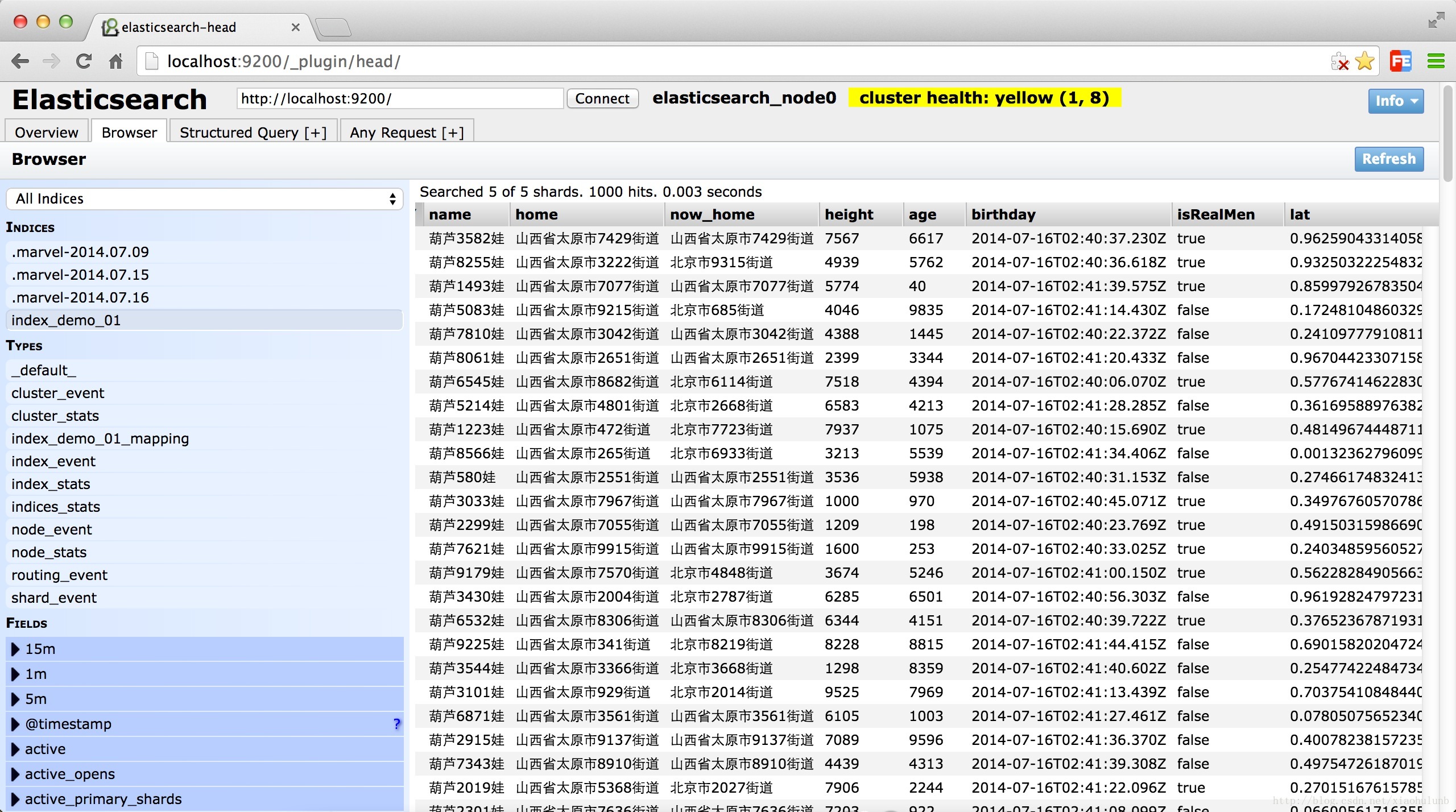
原创 2014年07月16日 14:33:25

* 标签：
* [elasticsearch](http://so.csdn.net/so/search/s.do?q=elasticsearch&t=blog) /
* [数据](http://so.csdn.net/so/search/s.do?q=%E6%95%B0%E6%8D%AE&t=blog) /
* [搜索](http://so.csdn.net/so/search/s.do?q=%E6%90%9C%E7%B4%A2&t=blog) /
* [索引](http://so.csdn.net/so/search/s.do?q=%E7%B4%A2%E5%BC%95&t=blog) /
* [文档](http://so.csdn.net/so/search/s.do?q=%E6%96%87%E6%A1%A3&t=blog)
* 28155

elasticsearch 分布式搜索系列专栏：<http://blog.csdn.net/xiaohulunb/article/category/2399789>

内容涉及代码GitHub地址: [点击打开链接](https://github.com/GourdErwa/MyNote/tree/master/framework/src/main/java/elasticsearch)

官方API：<http://www.elasticsearch.org/guide/en/elasticsearch/client/java-api/current/query-dsl-queries.html>



 @NotSolved未解决

**[java]** [view plain](https://blog.csdn.net/xiaohulunb/article/details/37877435) [copy](https://blog.csdn.net/xiaohulunb/article/details/37877435)

1. **package** com.elasticsearch;
3. **import** org.elasticsearch.action.ActionListener;
4. **import** org.elasticsearch.action.search.SearchResponse;
5. **import** org.elasticsearch.index.query.IndicesQueryBuilder;
6. **import** org.elasticsearch.index.query.QueryBuilder;
7. **import** org.elasticsearch.index.query.QueryBuilders;
9. /\*\*
10. \* Created by lw on 14-7-15.
11. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
12. \* elasticsearch以提供了一个完整的Java查询dsl其余查询dsl。
13. \* QueryBuilders工厂构建
14. \* API:
15. \* <a>http://www.elasticsearch.org/guide/en/elasticsearch/client/java-api/current/query-dsl-queries.html</a>
16. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
17. \*/
18. **public** **class** Es\_QueryBuilders\_DSL {
20. /\*\*
21. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
22. \* match query 单个匹配
23. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
24. \*/
25. **protected** **static** QueryBuilder matchQuery() {
26. **return** QueryBuilders.matchQuery("name", "葫芦4032娃");
27. }

30. /\*\*
31. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
32. \* multimatch  query
33. \* 创建一个匹配查询的布尔型提供字段名称和文本。
34. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
35. \*/
36. **protected** **static** QueryBuilder multiMatchQuery() {
37. //现住址和家乡在【山西省太原市7429街道】的人
38. **return** QueryBuilders.multiMatchQuery(
39. "山西省太原市7429街道",     // Text you are looking for
40. "home", "now\_home"       // Fields you query on
41. );
42. }
44. /\*\*
45. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
46. \* boolean query and 条件组合查询
47. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
48. \*/
49. **protected** **static** QueryBuilder booleanQuery() {
51. **return** QueryBuilders
52. .boolQuery()
53. .must(QueryBuilders.termQuery("name", "葫芦3033娃"))
54. .must(QueryBuilders.termQuery("home", "山西省太原市7967街道"))
55. .mustNot(QueryBuilders.termQuery("isRealMen", **false**))
56. .should(QueryBuilders.termQuery("now\_home", "山西省太原市"));
57. }
59. /\*\*
60. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
61. \* ids query
62. \* 构造一个只会匹配的特定数据 id 的查询。
63. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
64. \*/
65. **protected** **static** QueryBuilder idsQuery() {
66. **return** QueryBuilders.idsQuery().ids("CHszwWRURyK08j01p0Mmug", "ojGrYKMEQCCPvh75lHJm3A");
67. }
69. /\*\*
70. \* TODO NotSolved
71. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
72. \* constant score query
73. \* 另一个查询和查询,包裹查询只返回一个常数分数等于提高每个文档的查询。
74. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
75. \*/
76. **protected** **static** QueryBuilder constantScoreQuery() {
77. /\*return // Using with Filters
78. QueryBuilders.constantScoreQuery(FilterBuilders.termFilter("name", "kimchy"))
79. .boost(2.0f);\*/
81. // With Queries
82. **return** QueryBuilders.constantScoreQuery(QueryBuilders.termQuery("name", "葫芦3033娃"))
83. .boost(2.0f);
84. }
86. /\*\*
87. \* TODO NotSolved
88. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
89. \* disjunction max query
90. \* 一个生成的子查询文件产生的联合查询，
91. \* 而且每个分数的文件具有最高得分文件的任何子查询产生的，
92. \* 再加上打破平手的增加任何额外的匹配的子查询。
93. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
94. \*/
95. **protected** **static** QueryBuilder disMaxQuery() {
96. **return** QueryBuilders.disMaxQuery()
97. .add(QueryBuilders.termQuery("name", "kimchy"))          // Your queries
98. .add(QueryBuilders.termQuery("name", "elasticsearch"))   // Your queries
99. .boost(1.2f)
100. .tieBreaker(0.7f);
101. }
103. /\*\*
104. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
105. \* fuzzy query
106. \* 使用模糊查询匹配文档查询。
107. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
108. \*/
109. **protected** **static** QueryBuilder fuzzyQuery() {
110. **return** QueryBuilders.fuzzyQuery("name", "葫芦3582");
111. }
113. /\*\*
114. \* TODO NotSolved
115. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
116. \* has child / has parent
117. \* 父或者子的文档查询
118. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
119. \*/
120. **protected** **static** QueryBuilder hasChildQuery() {
121. **return** // Has Child
122. QueryBuilders.hasChildQuery("blog\_tag",
123. QueryBuilders.termQuery("tag", "something"));
125. // Has Parent
126. /\*return QueryBuilders.hasParentQuery("blog",
127. QueryBuilders.termQuery("tag","something"));\*/
128. }
130. /\*\*
131. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
132. \* matchall query
133. \* 查询匹配所有文件。
134. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
135. \*/
136. **protected** **static** QueryBuilder matchAllQuery() {
137. **return** QueryBuilders.matchAllQuery();
138. }
140. /\*\*
141. \* TODO NotSolved
142. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
143. \* more like this (field) query (mlt and mlt\_field)
144. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
145. \*/
146. **protected** **static** QueryBuilder moreLikeThisQuery() {
147. // mlt Query
148. QueryBuilders.moreLikeThisQuery("home", "now\_home") // Fields
149. .likeText("山西省太原市7429街道")                 // Text
150. .minTermFreq(1)                                 // Ignore Threshold
151. .maxQueryTerms(12);                             // Max num of Terms
152. // in generated queries
154. // mlt\_field Query
155. **return** QueryBuilders.moreLikeThisFieldQuery("home")              // Only on single field
156. .likeText("山西省太原市7429街道")
157. .minTermFreq(1)
158. .maxQueryTerms(12);
159. }
161. /\*\*
162. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
163. \* prefix query
164. \* 包含与查询相匹配的文档指定的前缀。
165. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
166. \*/
167. **protected** **static** QueryBuilder prefixQuery() {
168. **return** QueryBuilders.prefixQuery("name", "葫芦31");
169. }
171. /\*\*
172. \* TODO NotSolved
173. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
174. \* querystring query
175. \* 　　查询解析查询字符串,并运行它。有两种模式,这种经营。
176. \* 第一,当没有添加字段(使用{ @link QueryStringQueryBuilder #字段(String)},将运行查询一次,非字段前缀
177. \* 　　将使用{ @link QueryStringQueryBuilder # defaultField(字符串)}。
178. \* 第二,当一个或多个字段
179. \* 　　(使用{ @link QueryStringQueryBuilder #字段(字符串)}),将运行提供的解析查询字段,并结合
180. \* 　　他们使用DisMax或者一个普通的布尔查询(参见{ @link QueryStringQueryBuilder # useDisMax(布尔)})。
181. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
182. \*/
183. **protected** **static** QueryBuilder queryString() {
184. **return** QueryBuilders.queryString("+kimchy -elasticsearch");
185. }
187. /\*\*
188. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
189. \* range query
190. \* 查询相匹配的文档在一个范围。
191. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
192. \*/
193. **protected** **static** QueryBuilder rangeQuery() {
194. **return** QueryBuilders
195. .rangeQuery("name")
196. .from("葫芦1000娃")
197. .to("葫芦3000娃")
198. .includeLower(**true**)     //包括下界
199. .includeUpper(**false**); //包括上界
200. }
202. /\*\*
203. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
204. \* span queries (first, near, not, or, term)
205. \* 跨度查询
206. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
207. \*/
208. **protected** **static** QueryBuilder spanQueries() {
209. // Span First
210. QueryBuilders.spanFirstQuery(
211. QueryBuilders.spanTermQuery("name", "葫芦580娃"),  // Query
212. 30000                                             // Max查询范围的结束位置
213. );
215. // Span Near TODO NotSolved
216. QueryBuilders.spanNearQuery()
217. .clause(QueryBuilders.spanTermQuery("name", "葫芦580娃")) // Span Term Queries
218. .clause(QueryBuilders.spanTermQuery("name", "葫芦3812娃"))
219. .clause(QueryBuilders.spanTermQuery("name", "葫芦7139娃"))
220. .slop(30000)                                               // Slop factor
221. .inOrder(**false**)
222. .collectPayloads(**false**);
224. // Span Not TODO NotSolved
225. QueryBuilders.spanNotQuery()
226. .include(QueryBuilders.spanTermQuery("name", "葫芦580娃"))
227. .exclude(QueryBuilders.spanTermQuery("home", "山西省太原市2552街道"));
229. // Span Or TODO NotSolved
230. **return** QueryBuilders.spanOrQuery()
231. .clause(QueryBuilders.spanTermQuery("name", "葫芦580娃"))
232. .clause(QueryBuilders.spanTermQuery("name", "葫芦3812娃"))
233. .clause(QueryBuilders.spanTermQuery("name", "葫芦7139娃"));
235. // Span Term
236. //return QueryBuilders.spanTermQuery("name", "葫芦580娃");
237. }

240. /\*\*
241. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
242. \* term query
243. \* 一个查询相匹配的文件包含一个术语。。
244. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
245. \*/
246. **protected** **static** QueryBuilder termQuery() {
247. **return** QueryBuilders.termQuery("name", "葫芦580娃");
248. }

251. /\*\*
252. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
253. \* terms query
254. \* 一个查询相匹配的多个value
255. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
256. \*/
257. **protected** **static** QueryBuilder termsQuery() {
258. **return** QueryBuilders.termsQuery("name", // field
259. "葫芦580娃", "葫芦3812娃")                 // values
260. .minimumMatch(1);               // 设置最小数量的匹配提供了条件。默认为1。
261. }
263. /\*\*
264. \* TODO NotSolved
265. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
266. \* top children  query
267. \* 构建了一种新的评分的子查询，与子类型和运行在子文档查询。这个查询的结果是，那些子父文档文件匹配。
268. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
269. \*/
270. **protected** **static** QueryBuilder topChildrenQuery() {
271. **return** QueryBuilders.topChildrenQuery(
272. "blog\_tag",                                 // field
273. QueryBuilders.termQuery("name", "葫芦3812娃") // Query
274. )
275. .score("max")                               // max, sum or avg
276. .factor(5)
277. .incrementalFactor(2);
278. }
280. /\*\*
281. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
282. \* wildcard query
283. \* 　　实现了通配符搜索查询。支持通配符\* < /tt>,<tt>
284. \* 　　匹配任何字符序列(包括空),<tt> ? < /tt>,
285. \* 　　匹配任何单个的字符。注意该查询可以缓慢,因为它
286. \* 　　许多方面需要遍历。为了防止WildcardQueries极其缓慢。
287. \* 　　一个通配符词不应该从一个通配符\* < /tt>或<tt>
288. \* 　　< /tt> <tt> ?。
289. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
290. \*/
291. **protected** **static** QueryBuilder wildcardQuery() {
292. **return** QueryBuilders.wildcardQuery("name", "葫芦\*2娃");
293. }
295. /\*\*
296. \* TODO NotSolved
297. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
298. \* nested query
299. \* 嵌套查询
300. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
301. \*/
302. **protected** **static** QueryBuilder nestedQuery() {
303. **return** QueryBuilders.nestedQuery("location",               // Path
304. QueryBuilders.boolQuery()                      // Your query
305. .must(QueryBuilders.matchQuery("location.lat", 0.962590433140581))
306. .must(QueryBuilders.rangeQuery("location.lon").lt(0.00000000000000000003))
307. )
308. .scoreMode("total");                  // max, total, avg or none
309. }
311. /\*\*
312. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
313. \* indices query
314. \* 索引查询
315. \* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
316. \*/
317. **protected** **static** IndicesQueryBuilder indicesQuery() {
318. // Using another query when no match for the main one
319. QueryBuilders.indicesQuery(
320. QueryBuilders.termQuery("name", "葫芦3812娃"),
321. Es\_Utils.INDEX\_DEMO\_01, "index2"
322. )       //设置查询索引上执行时使用不匹配指数
323. .noMatchQuery(QueryBuilders.termQuery("age", "葫芦3812娃"));

326. // Using all (match all) or none (match no documents)
327. **return** QueryBuilders.indicesQuery(
328. QueryBuilders.termQuery("name", "葫芦3812娃"),
329. Es\_Utils.INDEX\_DEMO\_01, "index2"
330. )      // 设置不匹配查询,可以是 all 或者 none
331. .noMatchQuery("none");
332. }



337. **public** **static** **void** main(String[] args) {
338. Es\_Utils.startupClient();
339. **try** {
340. searchTest(indicesQuery());
341. } **catch** (Exception e) {
342. e.printStackTrace();
343. } **finally** {
344. Es\_Utils.shutDownClient();
345. }
346. }
348. **private** **static** **void** searchTest(QueryBuilder queryBuilder) {
349. //预准备执行搜索
350. Es\_Utils.client.prepareSearch(Es\_Utils.INDEX\_DEMO\_01)
351. .setTypes(Es\_Utils.INDEX\_DEMO\_01\_MAPPING)
352. .setQuery(queryBuilder)
353. .setFrom(0).setSize(20).setExplain(**true**)
354. .execute()
355. //注册监听事件
356. .addListener(**new** ActionListener<SearchResponse>() {
357. @Override
358. **public** **void** onResponse(SearchResponse searchResponse) {
359. Es\_Utils.writeSearchResponse(searchResponse);
360. }
362. @Override
363. **public** **void** onFailure(Throwable e) {
365. }
366. });
367. }
368. }