ElasticSearch 操作笔记

1. **建立索引**

1.1准备工作

1.1.1 可参照文献

<https://es.yemengying.com/4/4.5/4.5.2.html>

1.1.2 Maven配置

<dependency>  
 <groupId>org.springframework.data</groupId>  
 <artifactId>spring-data-commons</artifactId>  
 <version>1.12.0.RELEASE</version>  
</dependency>

<dependency>  
 <groupId>org.springframework.data</groupId>  
 <artifactId>spring-data-elasticsearch</artifactId>  
 <version>2.0.0.RELEASE</version>  
</dependency>  
<dependency>  
 <groupId>org.elasticsearch</groupId>  
 <artifactId>elasticsearch</artifactId>  
 <version>2.2.0</version>  
</dependency>

注意事项：elasticsearch的jar包版本需与使用的elasticsearch服务器版本一致

1.2 Spring中的ElasticSearch

1.2.1 spring-elasticsearch.xml配置

<!-- 声明初始化elasticsearch的索引库的包名 -->

<elasticsearch:repositories base-package=*"com.tuniu.pmcs.elasticsearch.dao"* />

<elasticsearch:transport-client id=*"client"* cluster-nodes=*"@{elasticsearch.clusterNodes}"* cluster-name=*"@{elasticsearch.clusterName}"* />

<bean name=*"elasticsearchTemplate"* class=*"org.springframework.data.elasticsearch.core.ElasticsearchTemplate"*><constructor-arg name=*"client"* ref=*"client"*/> </bean>

附：



1.2.2 创建索引库

声明接口继承ElasticsearchRepository类



第一个参数ProductDocument 为文档结构

第二个参数为唯一id的类型，即ProductDocument中@id修饰的值

附：



1.2.3 创建文档结构

文档结构的创建和正常pojo的创建基本相似。区别在于注解的使用：

可以使用@JSONField来实现别名



复杂类型的需要通过@Field来解析



其中

@Field(type = Nested) 可用来修饰list对象

@Field(type = Date) 可用来修饰string型的date对象

@Field(type = FieldType.Object) 可用来修饰复杂对象

附：



1.3 本地集群搭建

下载elasticsearch的官方版本(版本需要与程序中的api版本一致) 修改config文件下的elasticsearch.yml

添加



复制多份 只需修改http.port就可以了 相同cluster.name会自动成为集群

1. **存储document**

2.1 接入mq

通过mq接受外部消息，进行elasticsearch的索引文档更新。接入教程看[activemq 操作笔记.docx](activemq%20操作笔记.docx)

参考消息体：

{

"userId": 100002,

"userName": "亚瑟斯蒂",

"chapterId": 2450,

"chapterName": "高中",

"actionId": 445,

"actionName": "暑假",

"parters": [

{

"parterId": 252,

"parterName": "王二"

},

{

"parterId": 263,

"parterName": "赵三"

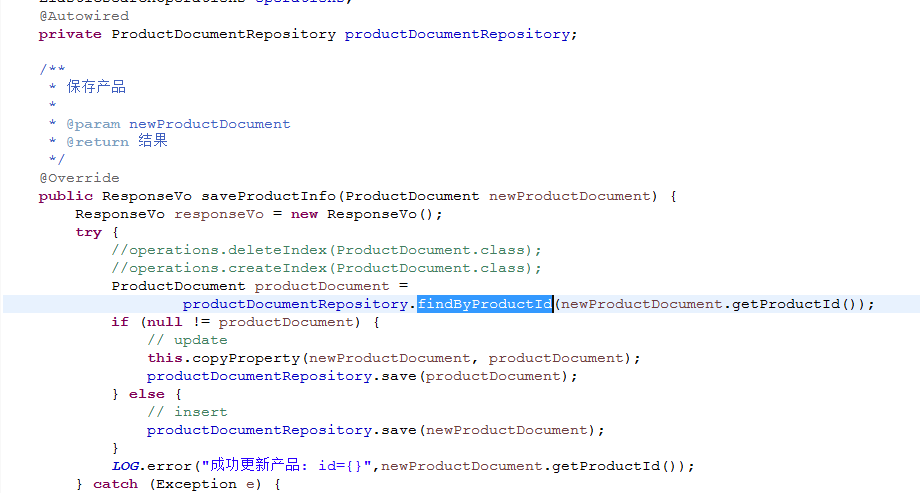
}

],

"createTime": "2015-12-25T17:06:36"

}

2.2获取文档存储



附：



1. **查询相关**

3.1 通过spring-data-elasticsearch的api查询

在Repository定义的时候创建自定义方法用and or 之类的关键字连接需要查询的参数

3.2 通过elasticsearch 自带transport的api查询

3.2.1 创建客户端client

Settings settings = Settings.*settingsBuilder*().put("cluster.name", clusterName).build();  
client = TransportClient.*builder*().settings(settings).build()  
 .addTransportAddress(new InetSocketTransportAddress(InetAddress.*getByName*(host), port));

附：



3.2.2 客户端查询

获取客户端，并指定索引分片，组装查询条件，获取查询结果并返回

Client client=esClient.getClient();  
BoolQueryBuilder boolQueryBuilder = new BoolQueryBuilder();  
boolQueryBuilder.must(new TermQueryBuilder("actionId",actionId));  
SearchResponse searchResponse = client.prepareSearch(*INDEX\_NAME*)  
 .setTypes(*INDEX\_TYPE*)  
 .setQuery(boolQueryBuilder)  
 .setExplain(true)  
 .execute()  
 .actionGet();  
  
SearchHits hits = searchResponse.getHits();  
  
if(hits.getTotalHits()==1){  
 for(SearchHit hit :hits){  
 actionDocument= JsonUtils.*toBean*(hit.getSource(),ActionDocument.class);  
 break;  
 }  
}

附：



3.2.3 查询条件组装

首先需要声明一个BoolQueryBuilder

BoolQueryBuilder boolQueryBuilder = new BoolQueryBuilder()



① 查询指定数字，使用TermQueryBuilder来指定位置和值

boolQueryBuilder.must(new TermQueryBuilder("productId", request.get("productId")));

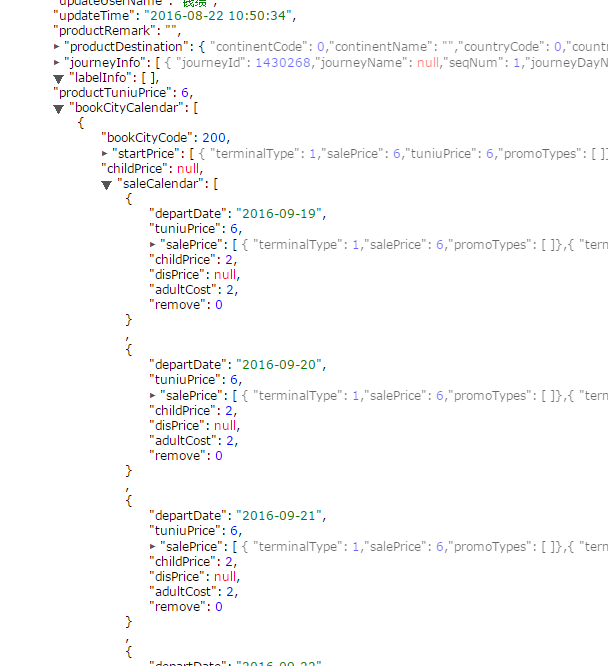
② 汉字模糊查询，使用MatchQueryBuilder

boolQueryBuilder.must(new MatchQueryBuilder("productName", request.get("name")).type(MatchQueryBuilder.Type.*PHRASE*));

③ 查询嵌套在子属性里的团期范围，

NestedQueryBuilder指定子属性根位置，

RangeQueryBuilder组装时间范围gte，lte，from， to



String startDate = request.get("departureBeginDate") == null ? null : String.*valueOf*(request.get("departureBeginDate")).split(" ")[0];  
String endDate = request.get("departureEndDate") == null ? null : String.*valueOf*(request.get("departureEndDate")).split(" ")[0];  
if (!StringUtils.*isEmpty*(startDate) && !StringUtils.*isEmpty*(endDate)) {  
 boolQueryBuilder.must(new NestedQueryBuilder("bookCityCalendar.saleCalendar",  
 new RangeQueryBuilder("bookCityCalendar.saleCalendar.departDate").gte(startDate).lte(endDate)));  
} else if (!StringUtils.*isEmpty*(startDate) && StringUtils.*isEmpty*(endDate)) {  
 boolQueryBuilder.must(new NestedQueryBuilder("bookCityCalendar.saleCalendar",  
 new RangeQueryBuilder("bookCityCalendar.saleCalendar.departDate").from(startDate)));  
} else if (StringUtils.*isEmpty*(startDate) && !StringUtils.*isEmpty*(endDate)) {  
 boolQueryBuilder.must(new NestedQueryBuilder("bookCityCalendar.saleCalendar",  
 new RangeQueryBuilder("bookCityCalendar.saleCalendar.departDate").to(endDate)));  
}

④ 当传入多个值进行匹配时，使用TermsQueryBuilder

boolQueryBuilder.must(new TermsQueryBuilder("productManagerId", request.get("mangers")));

⑤ 复杂的嵌套，BoolQueryBuilder中可以嵌套BoolQueryBuilder

BoolQueryBuilder childQueryBuilder = new BoolQueryBuilder();  
// 推荐标签  
if (request.get("proLabelList") != null && !labelList.isEmpty()) {  
 BoolQueryBuilder labelQueryBuilder = new BoolQueryBuilder();  
 labelQueryBuilder.must(new TermsQueryBuilder("labelInfo.labelId", labelList));  
 if (!StringUtils.*isEmpty*(startDate) && !StringUtils.*isEmpty*(endDate)) {  
 labelQueryBuilder.must(new RangeQueryBuilder("labelInfo.departDate").gte(startDate).lte(endDate));  
 } else if (!StringUtils.*isEmpty*(startDate) && StringUtils.*isEmpty*(endDate)) {  
 labelQueryBuilder.must(new RangeQueryBuilder("labelInfo.departDate").from(startDate));  
 } else if (StringUtils.*isEmpty*(startDate) && !StringUtils.*isEmpty*(endDate)) {  
 labelQueryBuilder.must(new RangeQueryBuilder("labelInfo.departDate").to(endDate));  
 }  
 childQueryBuilder.should(new NestedQueryBuilder("labelInfo", labelQueryBuilder));  
}  
// 高利润标签  
if (request.get("profitLabel") != null) {  
 BoolQueryBuilder profitQueryBuilder = new BoolQueryBuilder();  
 profitQueryBuilder.must(new TermQueryBuilder("profitInfo.isProfit", 1));  
 if (!StringUtils.*isEmpty*(startDate) && !StringUtils.*isEmpty*(endDate)) {  
 profitQueryBuilder.must(new RangeQueryBuilder("profitInfo.departDate").gte(startDate).lte(endDate));  
 } else if (!StringUtils.*isEmpty*(startDate) && StringUtils.*isEmpty*(endDate)) {  
 profitQueryBuilder.must(new RangeQueryBuilder("profitInfo.departDate").from(startDate));  
 } else if (StringUtils.*isEmpty*(startDate) && !StringUtils.*isEmpty*(endDate)) {  
 profitQueryBuilder.must(new RangeQueryBuilder("profitInfo.departDate").to(endDate));  
 }  
 childQueryBuilder.should(new NestedQueryBuilder("profitInfo", profitQueryBuilder));  
}  
boolQueryBuilder.must(childQueryBuilder);

附：



1. **常见问题**

4.1 版本问题

高版本的Elasticsearch依赖spring-data-commons

<dependency>  
 <groupId>org.elasticsearch</groupId>  
 <artifactId>elasticsearch</artifactId>  
 <version>2.2.0</version>  
</dependency>

<dependency>  
 <groupId>org.springframework.data</groupId>  
 <artifactId>spring-data-commons</artifactId>  
 <version>1.12.0.RELEASE</version>  
</dependency>

4.2 索引库检查文档类型

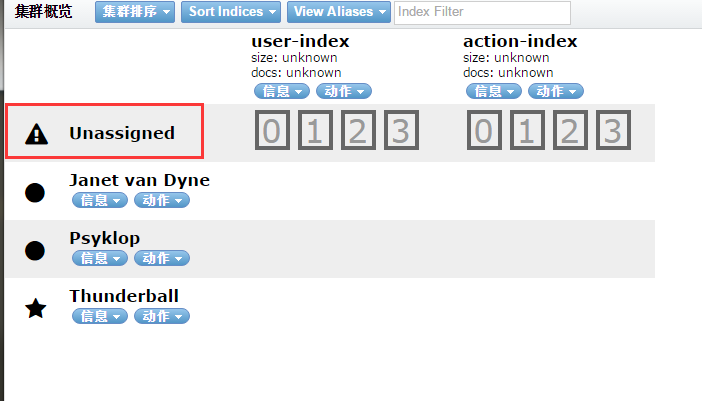
当配置扫索引库包的时候 如果在索引库类有自定义查询方法，elasticsearch在容器启动的时候会检查方法中包含的参数是否在文档类型中存在，如果不存在会报Error creating bean with name 'userRepository': Invocation of init method failed; nested exception is org.springframework.data.mapping.PropertyReferenceException: No property chapterId found for type UserDocument! Did you mean 'chapters'?

<elasticsearch:repositories base-package="com.boneix.elasticsearch.repository" />

public interface ActionRepository extends ElasticsearchRepository<ActionDocument,String> {  
 ActionDocument findByActionIdAndUserId(int actionId, int userId);  
}

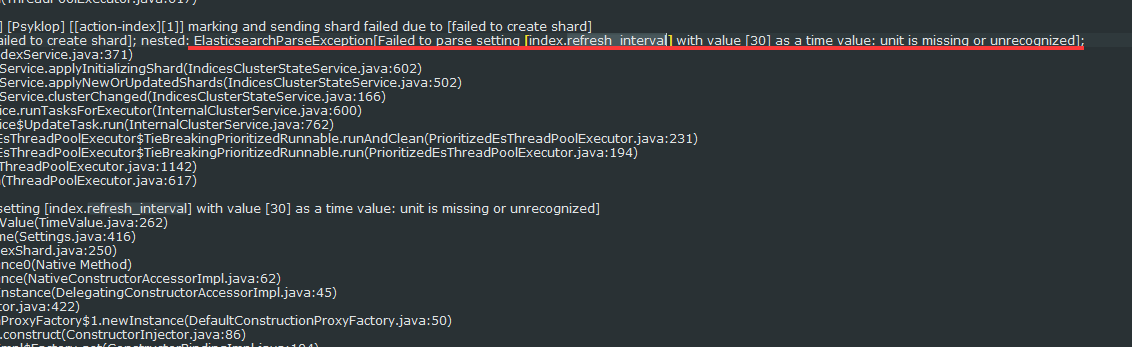
public class ActionDocument {  
  
 private @Id int actionId;  
  
 private String actionName;  
  
 private int userId;

4.3 出现未分配的索引库

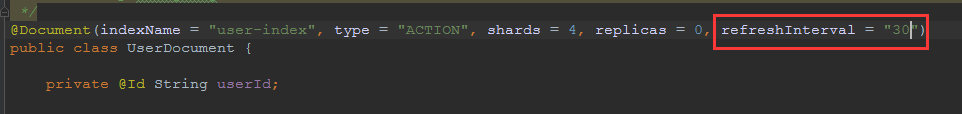


问题原因：

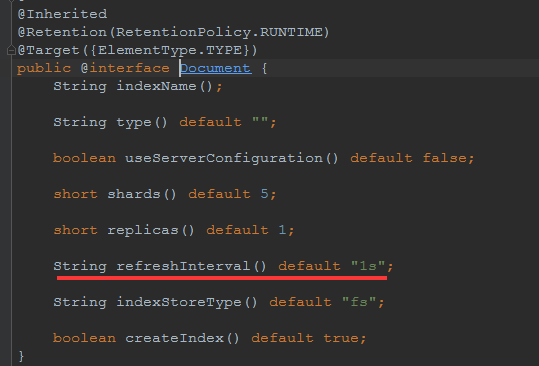
elasticsearch日志如下



设置document的时候refreshInterval设置为30



查看注解document发现refreshInterval默认为1s



原因分析：

新版elasticsearch中对api有所改变 做了严格限制

https://www.elastic.co/guide/en/elasticsearch/reference/2.2/index-modules.html

解决方式：

将refreshInterval=”30s” 然后清空原所有集群数据 接着重启所有集群

