**ElasticSearch里面复杂关系数据的存储方式**

**转载**[**gongpulin**](https://me.csdn.net/gongpulin)**最后发布于2017-12-10 15:58:26 阅读数 2057  收藏**

**展开**

**在传统的数据库里面，对数据关系描述无外乎三种，一对一，一对多和多对多的关系，如果有关联关系的数据，通常我们在建表的时候会添加主外键来建立数据联系，然后在查询或者统计时候通过join来还原或者补全数据，最终得到我们需要的结果数据，那么转化到ElasticSearch里面，如何或者怎样来处理这些带有关系的数据。**

**我们都知道ElasticSearch是一个NoSQL类型的数据库，本身是弱化了对关系的处理，因为像lucene，es，solr这样的全文检索框架对性能要求都是比较高的，一旦出现join这样的操作，性能会非常差，所以在使用搜索框架时，我们应该避免把搜索引擎当做关系型数据库用。**

**当然，现实数据肯定是有关系的，那么在es里面是如何处理和管理这些带有关系的数据呢？**

**大家都知道，es天生对json数据支持的非常完美，只要是标准的json结构的数据，无论多么复杂，无论是嵌套多少层，都能存储到es里面，进而能够查询和分析，检索。在这种机制上，es处理和管理关系主要有三种方式：**

**一，使用objcet和array[object]的字段类型自动存储多层结构的json数据**

**这是es默认的机制，也就是我们并没有设置任何mapping，直接向es服务端插入一条复杂的json数据，也能成功插入，并能支持检索，（能这样操作是因为es默认用的是动态mapping，只要插入的是标准的json结构就会自动转换，当然我们也能控制mapping类型，es里面有动态mapping和静态maping，静态mapping还分严格类型，弱类型，一般类型，在此不再展开，有兴趣的可以从官网了解下）如下面一条数据：**

1. **{**
2. **"name" : "Zach",**
3. **"car" : [**
4. **{**
5. **"make" : "Saturn",**
6. **"model" : "SL"**
7. **},**
8. **{**
9. **"make" : "Subaru",**
10. **"model" : "Imprezza"**
11. **}**
12. **]**
13. **}**

**最终转化成的存储结构是下面这样的：**

1. **{**
2. **"name" : "Zach",**
3. **"car.make" : ["Saturn", "Subaru"]**
4. **"car.model" : ["SL", "Imprezza"]**
5. **}**

**因为es的底层lucene是天生支持多值域的存储，所以在上面看起来像数组的结构，其实在es里面存储的就是这个字段多值域。**

**然后检索的时候.符号就能检索相对应的内容。这样的一条数据，其实已经包含了数据和关系，看起来像一对多的关系，一个人拥有多辆汽车。但实际上并不能算严格意义上的关系，因为lucene底层是扁平化存储的，这样以来多个汽车的数据实际都是存到一起的混杂的，你没办法单独获取到这个人某一辆汽车的数据，因为整条数据都是一个整体，无论什么操作整条数据都会返回。**

**二，使用nested[object]类型，存储拥有多级关系的数据**

**在方案一里面，我们指出了array存储的数组对象，并不是严格意义的关系，因为第二层的数据是没有分离的，如果想要分离，就必须使用nested类型来显式定义数据结构。只有这样，第二层的多个汽车数据才是独立的互不影响，也就是说可以单独获取或查询某一辆汽车的数据。**

**同样的json数据：**

1. **{**
2. **"name" : "Zach",**
3. **"car" : [**
4. **{**
5. **"make" : "Saturn",**
6. **"model" : "SL"**
7. **},**
8. **{**
9. **"make" : "Subaru",**
10. **"model" : "Imprezza"**
11. **}**
12. **]**
13. **}**

**在方案1里面，最终到es里面会存储一条数据，在第二种类型里面，而如果声明了car类型是nested，那么最终存储到es的数量会显示3，这里解释一下3是怎么来的 = 1个root文档+2个汽车文档，nested声明类型，每一个实例都是一个新的document，所以在查询的时候才能够独立进行查询，并且性能还不错，因为es底层会把整条数据存在同一个shard的lucene的sengment里面，缺点是更新的代价比较大，每一个子文档的更新都要重建整个结构体的索引，所以nested适合不经常update的嵌套多级关系的场景。**

**nested类型的数据，需要用其指定的查询和聚合方法才能生效，普通的es查询只能查询1级也就是root级的属性，嵌套的属性是不能查的，如果想要查，必须用嵌套查询或者聚合才行。**

**嵌套应用有两种模式：**

**第一种：嵌套查询**

**每个查询都是单个文档内生效，包括排序,**

**第二种：嵌套聚合或者过滤**

**对同一层级的所有文档都是全局生效，包括过滤排序**

**三，parent/children 父子关系**

**parent/children 模式与nested非常类似，但是应用场景侧重点有所不同。**

**在使用parent/children管理关联关系时，es会在每个shard的内存中维护一张关系表，在检索时，通过has\_parent和has\_child过滤器来得到关联的数据，这种模式下父文档与子文档也是独立的，查询性能会比nested模式稍低，因为父文档和子文档在插入的时候会通过route使得他们都分布在同一个shard里面，但并不保证在同一个lucene的sengment索引段里面，所以检索性能稍低，除此之外，每次检索es都需要从内存的关系表里面得到数据关联的信息，也需要花费一定的时间，相比nested的优势在于，父文档或者子文档的更新，并不影响其他的文档，所以对于更新频繁的多级关系，使用parent/children模式，最为合适不过。**

**父文档的mapping type：**

1. **{**
2. **"mappings":{**
3. **"person":{**
4. **"name":{**
5. **"type":"string"**
6. **}**
7. **}**
8. **}**
9. **}**

**子文档的mapping type：**

1. **{**
2. **"homes":{**
3. **"\_parent":{**
4. **"type" : "person"**
5. **},**
6. **"state" : {**
7. **"type" : "string"**
8. **}**
9. **}**
10. **}**

**插入数据时，需要先插入父文档：**

1. **curl -XPUT localhost:9200/test/person/zach/ -d*'***
2. **{**
3. **"name" : "Zach"**
4. **}**

**然后插入子文档时，需要加上路由字段：**

1. **$ curl -XPOST localhost:9200/homes?parent=zach -d*'***
2. **{**
3. **"state" : "Ohio"**
4. **}**
5. **$ curl -XPOST localhost:9200/test/homes?parent=zach -d*'***
6. **{**
7. **"state" : "South Carolina"**
8. **}**

**最终，父文档zach就关联上了两个子文档，在查询时候可以通过parent/children特定查询来获取数据。**

**总结：**

**方法一：**

**（1）简单，快速，性能较高**

**（2）对维护一对一的关系比较擅长**

**（3）不需要特殊的查询**

**方法二：**

**（1）由于底层存储在同一个lucene的sengment里，所以读取和查询性能对比方法三更快**

**（2）更新单个子文档，会重建整个数据结构，所以不适合更新频繁的嵌套场景**

**（3）可以维护一对多和多对多的存储关系**

**方法三：**

**（1）多个关系数据，存储完全独立，但是存在同一个shard里面，所以读取和查询性能比方法二稍低**

**（2）需要额外的内存，维护管理关系列表**

**（3）更新文档不影响其他的子文档，所以适合更新频繁的场景**

**（4）排序和评分操作比较麻烦，需要额外的脚本函数支持**

**参考文档：**

[**https://www.elastic.co/blog/managing-relations-inside-elasticsearch**](https://www.elastic.co/blog/managing-relations-inside-elasticsearch)