**Elasticsearch 如何处理存在关联关系的数据?**

**一，关系数据库的三大范式**

**本文基于Elasticsearch7.x**

**什么是范式? 范式就是数据建模的规则.**

**第一范式 : 确保每列保持原子性.**

**数据库表中的所有字段都是不可分割的原子值.**

**第二范式: 确保表中的每列都和主键相关.**

**一张数据库表中只能保存一种数据, 不可以把多种数据保存在同一张数据库表中. 比如订单相关的信息会设计三张表: 订单表, 订单项表, 商品表.**

**第三范式: 确保每列都和主键直接相关, 而不是间接相关.**

**比如一个订单表里只需要保存userId, 不需要保存整个用户信息.**

**关系数据库的三大范式简化了写操作, 但读操作性能不高(join操作非常耗费性能), 并且扩展性很差. 而反范式化设计在文档中保存冗余的数据, 无需处理join操作, 数据读取性能很好, 但反范式化设计不适合数据频繁修改的场景.**

**二 ，Elasticsearch 处理存在关联关系的数据**

**Elasticsearch使用的是非关系型的数据存储引擎, 即反范式化设计, 那Elasticsearch如何处理存在关联关系的数据呢? 有三种方法, 即三种数据类型.**

**对象类型(Object)**

**嵌套类型(Nested)**

**Join类型(Join)**

1. **对象类型(Object)**

**使用Object数据类型来将电影和演员信息存储到一个doc里.**

1. **定义Mapping**

**PUT /my\_movies**

**{**

**"mappings": {**

**"properties": {**

**"title": {**

**"type": "text",**

**"fields": {**

**"keyword": {**

**"type": "keyword",**

**"ignore\_above": 256**

**}**

**}**

**},**

**"actors": {**

**"properties": {**

**"first\_name": {**

**"type": "keyword"**

**},**

**"last\_name": {**

**"type": "keyword"**

**}**

**}**

**}**

**}**

**}**

**}**

**(2) 添加数据**

**PUT /my\_movies/\_doc/1**

**{**

**"title": "Speed",**

**"actors": [**

**{**

**"first\_name": "Keanu",**

**"last\_name": "Reeves"**

**},**

**{**

**"first\_name": "Dennis",**

**"last\_name": "Hopper"**

**}**

**]**

**}**

**(3) 搜索**

**GET /my\_movies/\_search**

**{**

**"query": {**

**"bool": {**

**"must": [**

**{**

**"match": {**

**"actors.first\_name": "Keanu"**

**}**

**},**

**{**

**"match": {**

**"actors.last\_name": "Hopper"**

**}**

**}**

**]**

**}**

**}**

**}**

**结果:**

**"hits" : [**

**{**

**"\_index" : "my\_movies",**

**"\_type" : "\_doc",**

**"\_id" : "1",**

**"\_score" : 0.723315,**

**"\_source" : {**

**"title" : "Speed",**

**"actors" : [**

**{**

**"first\_name" : "Keanu",**

**"last\_name" : "Reeves"**

**},**

**{**

**"first\_name" : "Dennis",**

**"last\_name" : "Hopper"**

**}**

**]**

**}**

**}**

**]**

**我们想要的搜索结果应该是返回为空, 但Elasticsearch却返回了一个结果, 为什么会这样呢? 这是因为对象数组被处理成了扁平式键值对的结构:**

**"title":"Speed"**

**"actors.first\_name":["Keanu","Dennis"]**

**"actors.last\_name":["Reeves","Hopper"]**

**1**

**2**

**3**

**所以在进行搜索时不能返回我们想要的结果. 即对象类型不适合处理关联关系.**

1. **嵌套类型(Nested)**

**从上文的实例我们知道, 对象数组在建立倒排索引时对象不是独立的, 最终导致结果不准确, 而Nested数据类型在为对象数组创建索引时, 每个对象都是独立的, 通过nested query就可以得到我们想要的结果.**

**(1) 定义Maping**

**PUT /my\_movies**

**{**

**"mappings": {**

**"properties": {**

**"title": {**

**"type": "text",**

**"fields": {**

**"keyword": {**

**"type": "keyword",**

**"ignore\_above": 256**

**}**

**}**

**},**

**"actors": {**

**"type": "nested",**

**"properties": {**

**"first\_name": {**

**"type": "keyword"**

**},**

**"last\_name": {**

**"type": "keyword"**

**}**

**}**

**}**

**}**

**}**

**}**

**(2) 添加数据**

**PUT /my\_movies/\_doc/1**

**{**

**"title": "Speed",**

**"actors": [**

**{**

**"first\_name": "Keanu",**

**"last\_name": "Reeves"**

**},**

**{**

**"first\_name": "Dennis",**

**"last\_name": "Hopper"**

**}**

**]**

**}**

**(3) 搜索**

**GET /my\_movies/\_search**

**{**

**"query": {**

**"bool": {**

**"must": [**

**{**

**"nested": {**

**"path": "actors",**

**"query": {**

**"bool": {**

**"must": [**

**{**

**"match": {**

**"actors.first\_name": "Keanu"**

**}**

**},**

**{**

**"match": {**

**"actors.last\_name": "Hopper"**

**}**

**}**

**]**

**}**

**}**

**}**

**}**

**]**

**}**

**}**

**}**

**结果:**

**"hits" : {**

**"total" : {**

**"value" : 0,**

**"relation" : "eq"**

**},**

**"max\_score" : null,**

**"hits" : [ ]**

**}**

1. **Join类型(Join)**

**嵌套类型处理关联关系有一个的局限性, 即每次更新都需要重新索引整个对象（包括根对象和嵌套对象）.**

**Elasticsearch 提供了类似关系型数据库中 Join 的实现, 即Join 数据类型. Join数据类型定义了文档之间的父子关系, 从而分离两个对象.**

**父文档和子文档是两个独立的文档.**

**更新父文档无需重新索引子文档.**

**子文档被添加, 更新或者删除也不会影响到父文档和其他的子文档.**

**我们来看一个博客和评论的实例.**

**(1) 定义Mapping**

**PUT /my\_blogs**

**{**

**"settings": {**

**"number\_of\_shards": 2**

**},**

**"mappings": {**

**"properties": {**

**"title": {**

**"type": "keyword"**

**},**

**"content": {**

**"type": "text"**

**},**

**"comment": {**

**"type": "text"**

**},**

**"username": {**

**"type": "keyword"**

**},**

**"blog\_comments\_relation" : {**

**"type": "join",**

**"relations": {**

**"blog": "comment"**

**}**

**}**

**}**

**}**

**}**

**注意这里把主分片数定义为2, blog和comment之间是父子关系.**

**(2) 添加数据**

**a. 添加博客数据**

**PUT /my\_blogs/\_doc/blog1**

**{**

**"title": "Learning Elasticsearch",**

**"content": "learning ELK @ tyshawn",**

**"blog\_comments\_relation": {**

**"name": "blog"**

**}**

**}**

**PUT /my\_blogs/\_doc/blog2**

**{**

**"title": "Learning Hadoop",**

**"content": "learning Hadoop @ tyshawn",**

**"blog\_comments\_relation": {**

**"name": "blog"**

**}**

**}**

**blog1和blog2是\_id, 要注意\_id不一定是数字.**

**b. 添加评论数据**

**PUT /my\_blogs/\_doc/comment1?routing=blog1**

**{**

**"comment": "I am learning ELK",**

**"username": "Jack",**

**"blog\_comments\_relation": {**

**"name": "comment",**

**"parent": "blog1"**

**}**

**}**

**PUT /my\_blogs/\_doc/comment2?routing=blog2**

**{**

**"comment": "I like Hadoop!!!!!",**

**"username": "Jack",**

**"blog\_comments\_relation": {**

**"name": "comment",**

**"parent": "blog2"**

**}**

**}**

**添加评论时指定了路由, 确保父子文档索引到相同的分片. 目的是确保join查询的性能.**

**(3) 查询**

**Join类型特有的查询:**

**parent\_id**

**通过对父文档id进行查询, 返回所有相关的子文档.**

**has\_child**

**对子文档进行查询, 返回拥有相关子文档的父文档. 父子文档在相同的分片上, 所以Join效率高.**

**has\_parent**

**对父文档进行查询, 返回所有相关的子文档.**

**a. parent\_id**

**GET /my\_blogs/\_search**

**{**

**"query": {**

**"parent\_id": {**

**"type": "comment",**

**"id": "blog2"**

**}**

**}**

**}**

**结果:**

**"hits" : [**

**{**

**"\_index" : "my\_blogs",**

**"\_type" : "\_doc",**

**"\_id" : "comment2",**

**"\_score" : 0.6931472,**

**"\_routing" : "blog2",**

**"\_source" : {**

**"comment" : "I like Hadoop!!!!!",**

**"username" : "Jack",**

**"blog\_comments\_relation" : {**

**"name" : "comment",**

**"parent" : "blog2"**

**}**

**}**

**}**

**]**

**b. has\_child**

**GET /my\_blogs/\_search**

**{**

**"query": {**

**"has\_child": {**

**"type": "comment",**

**"query": {**

**"match": {**

**"username": "Jack"**

**}**

**}**

**}**

**}**

**}**

**结果:**

**"hits" : [**

**{**

**"\_index" : "my\_blogs",**

**"\_type" : "\_doc",**

**"\_id" : "blog1",**

**"\_score" : 1.0,**

**"\_source" : {**

**"title" : "Learning Elasticsearch",**

**"content" : "learning ELK @ tyshawn",**

**"blog\_comments\_relation" : {**

**"name" : "blog"**

**}**

**}**

**},**

**{**

**"\_index" : "my\_blogs",**

**"\_type" : "\_doc",**

**"\_id" : "blog2",**

**"\_score" : 1.0,**

**"\_source" : {**

**"title" : "Learning Hadoop",**

**"content" : "learning Hadoop @ tyshawn",**

**"blog\_comments\_relation" : {**

**"name" : "blog"**

**}**

**}**

**}**

**]**

**c. has\_parent**

**GET /my\_blogs/\_search**

**{**

**"query": {**

**"has\_parent": {**

**"parent\_type": "blog",**

**"query": {**

**"match": {**

**"title": "Learning Hadoop"**

**}**

**}**

**}**

**}**

**}**

**结果:**

**"hits" : [**

**{**

**"\_index" : "my\_blogs",**

**"\_type" : "\_doc",**

**"\_id" : "comment2",**

**"\_score" : 1.0,**

**"\_routing" : "blog2",**

**"\_source" : {**

**"comment" : "I like Hadoop!!!!!",**

**"username" : "Jack",**

**"blog\_comments\_relation" : {**

**"name" : "comment",**

**"parent" : "blog2"**

**}**

**}**

**}**

**]**

**(4) 更新子文档**

**更新子文档不会影响到父文档.**

**POST /my\_blogs/\_update/comment2?routing=blog2**

**{**

**"doc": {**

**"comment": "Hello Hadoop??"**

**}**

**}**

**通过id和routing进行查询**

**GET /my\_blogs/\_doc/comment2?routing=blog2**

**1**

**结果:**

**{**

**"\_index" : "my\_blogs",**

**"\_type" : "\_doc",**

**"\_id" : "comment2",**

**"\_version" : 2,**

**"\_seq\_no" : 4,**

**"\_primary\_term" : 1,**

**"\_routing" : "blog2",**

**"found" : true,**

**"\_source" : {**

**"comment" : "Hello Hadoop??",**

**"username" : "Jack",**

**"blog\_comments\_relation" : {**

**"name" : "comment",**

**"parent" : "blog2"**

**}**

**}**

**}**

**Nested类型与Join类型对比**

**Object数据类型不适合处理具有关联关系的数据, 那Nested类型和Join类型分别适用于什么场景呢? 我们来看下两者之间的对比.**

**对比 Nested Join**

**优点 文档存储在⼀起, 读取性能⾼ 父子文档可以独立更新**

**缺点 更新嵌套的子文档时, 需要更新整个文档 需要额外的内存维护关系, 读取性能相对差**

**适用场景 子文档偶尔更新, 以查询为主 子文档更新频繁**

**其他处理方式**

**在实际开发中我们也可以不使用Nested和Join类型来处理具有关联关系的数据, 我们可以直接把数据库表和ES索引建立一对一关系, 然后通过ES查询出数据后, 在应用端处理关联关系. 或者直接把具有关联关系的数据表合并建立一个ES索引, 这种处理方式是最简单的.**