

目录

1 Mapping基本概念

2 Mapping与文档解析流程

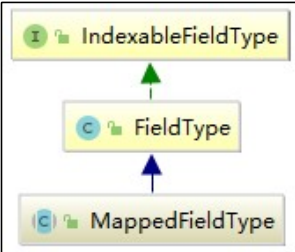
3 Bool、Nested等字段实现

4 Nested Docvalue方案

基本概念：FieldType

FieldType

Lucene用于存储字段的索引、存储设置



MappedFieldType

ES用于额外的字段设置
(字段名、索引和搜索分析器、字段元数据)

XXXXXFieldType

ES定义的各类具体字段类型

- Field datatypes
 - Alias
 - Arrays
 - Binary
 - Boolean
 - Date
 - Date nanoseconds
 - Dense vector

```
private boolean stored;
private boolean tokenized = true;
private boolean storeTermVectors;
private boolean storeTermVectorOffsets;
private boolean storeTermVectorPositions;
private boolean storeTermVectorPayloads;
private boolean omitNorms;
private IndexOptions indexOptions = IndexOptions.NONE;
private boolean frozen;
private DocValuesType docValueType = DocValuesType.NONE;
private int dimensionCount;
private int indexDimensionCount;
private int dimensionNumBytes;
private Map<String, String> attributes;
```

```
private String name;
private float boost;
// TODO: remove this docvalues flag and use docValues
private boolean docValues;
private NamedAnalyzer indexAnalyzer;
private NamedAnalyzer searchAnalyzer;
private NamedAnalyzer searchQuoteAnalyzer;
private SimilarityProvider similarity;
private Object nullValue;
private String nullValueAsString; // for search
private boolean eagerGlobalOrdinals;
private Map<String, String> meta;
```

- Mapping
- Removal of mapping type
 - + Field datatypes
 - + Meta-Fields
 - + Mapping parameters
 - + Dynamic Mapping

基本概念：Mapper

Mapper	ES使用Mapper将文档的字段解析为Lucene::IndexableField
Mapper::Builder	ES使用MapperBuilder来获取对应的Mapper
TypeParser	ES使用各个字段的TypeParser来获取对应的MapperBuilder

```
{
  "mappings": {
    "properties": {
      "is_published": {
        "type": "boolean"
      }
    }
  }
}
```

XContentParser处理

```
"is_published": {
  "type": "boolean"
}
```

TypeParser处理

得到各项类型参数

解析出IndexableField用于Lucene索引

构造BooleanFieldMapper

构造BooleanBuilder

```
{
  "is_published": "true"
}
```

基本概念：索引Mapper

DocumentMapperParser

用于根据Mapping获取DocumentMapper

DocumentMapper

用于根据文档json获取ParsedDocument

ParsedDocument

存储解析好的Document

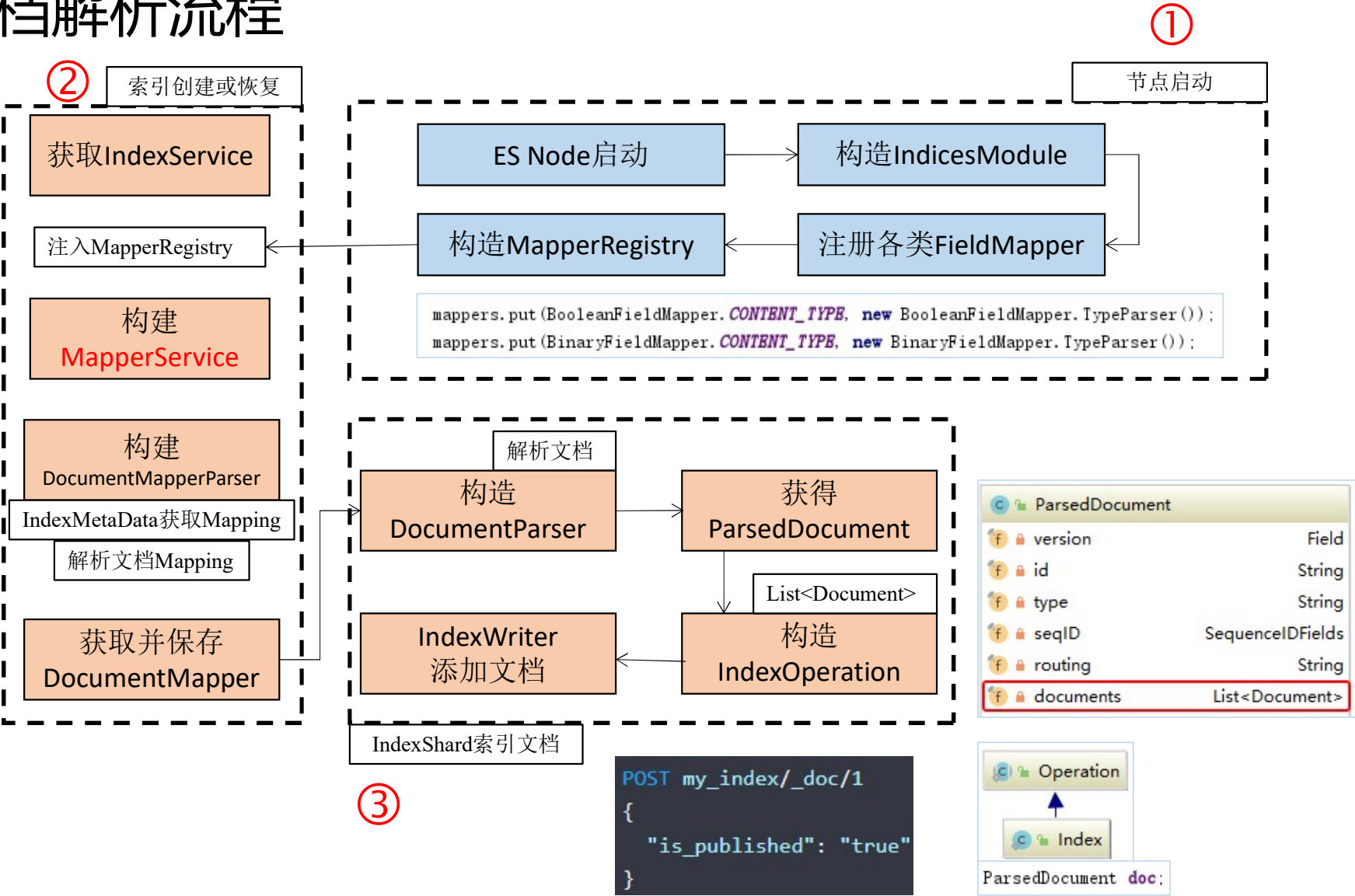
Document

ES对Lucene Document的扩展，增加了parent、path等信息

```
class Document implements Iterable<IndexableField>
```

Mapping与文档解析流程

```
PUT my_index
{
  "mappings": {
    "properties": {
      "is_published": {
        "type": "boolean"
      }
    }
  }
}
```



Mapping解析流程

从IndexMetaData获取
Compressed Mapping Source

```
{
  "_doc": {
    "properties": {
      "user": {
        "type": "nested",
        "properties": {
          "last": {
            "type": "keyword"
          },
          "first": {
            "type": "keyword"
          }
        }
      }
    }
  }
}
```

解析source获取
source mapping

```
mapping = {LinkedHashMap@10256} size = 1
0 = {LinkedHashMap$Entry@10259} "properties" -> " size = 1"
key = "properties"
value = {LinkedHashMap@10261} size = 1
0 = {LinkedHashMap$Entry@10264} "user" -> " size = 2"
key = "user"
value = {LinkedHashMap@10267} size = 2
0 = {LinkedHashMap$Entry@10270} "type" -> "nested"
1 = {LinkedHashMap$Entry@10271} "properties" -> " size = 2"
```

构造ParserContext用于存储解析
过程需要用到的工具与状态

```
parserContext = {Mapper$TypeParser$ParserContext}
similarityLookupService = {DocumentMapperParser}
mapperService = {MapperService@10223}
typeParsers = 所有type parser
arg$1 = {Collections$UnmodifiableMap@1023}
0 = {Collections$UnmodifiableMap$Unmodifiable}
key = "half_float"
value = {NumberFieldMapper$TypeParser}
type = {NumberFieldMapper$NumberFieldMapper}
1 = {Collections$UnmodifiableMap$Unmodifiable}
key = "float"
value = {NumberFieldMapper$TypeParser}
```

针对meta-fields属性

遍历source mapping, 逐一用
TypeParser处理每个field

```
Iterator<Map.Entry<String, Object>> iterator = mapping.entrySet().iterator();
// parse DocumentMapper
while(iterator.hasNext()) {
  Map.Entry<String, Object> entry = iterator.next();
  String fieldName = entry.getKey();
  Object fieldNode = entry.getValue();

  MetadataFieldMapper.TypeParser typeParser = rootTypeParsers.get(fieldName);
}
```

嵌套map表示的field

获取field对应的TypeParser

针对properties属性

使用RootObjectMapper根据
mapping构造Mapper::Builder

```
Mapper.TypeParser typeParser = parserContext.typeParser(type);
if (typeParser == null) {...}
String[] fieldNameParts = fieldName.split( regex: "\\." );
String realFieldName = fieldNameParts[fieldNameParts.length - 1];
Mapper.Builder<?, ?> fieldBuilder = typeParser.parse(realFieldName, propNode, parserContext);
for (int i = fieldNameParts.length - 2; i >= 0; --i) {...}
objBuilder.add(fieldBuilder);
propNode.remove( key: "type" );
DocumentMapperParser.checkNoRemainingFields(fieldName, propNode, parserContext.indexSettings);
```

构造出
DocumentMapper

Document解析流程

处理所有meta field

```
▼ p metadataFieldsMappers = {MetadataFieldMapper[9]@10897}
> 0 = {FieldNamesFieldMapper@10912}
> 1 = {IdFieldMapper@10913}
> 2 = {IgnoredFieldMapper@10916}
> 3 = {IndexFieldMapper@10917}
> 4 = {RoutingFieldMapper@10920}
> 5 = {SeqNoFieldMapper@10915}
> 6 = {SourceFieldMapper@10914}
> 7 = {TypeFieldMapper@10918}
> 8 = {VersionFieldMapper@10919}
```

```
for (MetadataFieldMapper metadataMapper : metadataFieldsMappers) {
    metadataMapper.preParse(context);
}
```

逐个处理文档每个字段

根据token类型采用对应处理方式

```
while (token != XContentParser.Token.END_OBJECT) {
    if (token == XContentParser.Token.FIELD_NAME) {...}
    else if (token == XContentParser.Token.START_OBJECT) {...}
    else if (token == XContentParser.Token.START_ARRAY) {...}
    else if (token == XContentParser.Token.VALUE_NULL) {...}
    else if (token == null) {...}
    else if (token.isValue()) {
        parseValue(context, mapper, currentFieldName, token, paths);
    }
    token = parser.nextToken();
}
```

FieldMapper解析流程: TextField

满足如下条件的字段才需要被处理（其他字段只会存于_source中）：
需要被索引、需要存储字段值

创建Lucene::Field

```
if (fieldType().indexOptions() != IndexOptions.NONE || fieldType().stored()) {  
    Field field = new Field(fieldType().name(), value, fieldType());  
    fields.add(field);  
}
```

将Field放入Document

```
for (IndexableField field : fields) {  
    context.doc().add(field);  
}
```

一个ES Field可能会根据设置创建多个Lucene Field

为了加速exist查询，若该字段值不为空，且没有存norm或docvalue信息，将该字段名存于_field_names字段中

index_prefixes

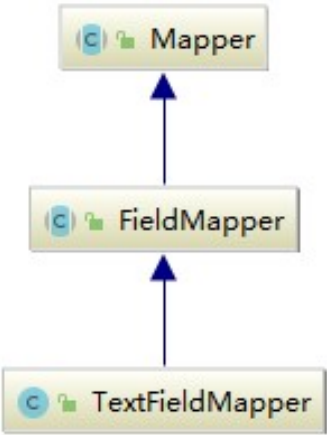
用于加速前缀搜索

If enabled, term prefixes of between 2 and 5 characters are indexed into a separate field. This allows prefix searches to run more efficiently, at the expense of a larger index.

index_phrases

用于加速Phrase搜索

If enabled, two-term word combinations (*shingles*) are indexed into a separate field. This allows exact phrase queries (no slop) to run more efficiently, at the expense of a larger index. Note



FieldMapper解析流程: BooleanField

满足如下条件的字段才需要被处理（其他字段只会存于_source中）：
需要被索引、需要存储字段值、或需要字段具有docValue

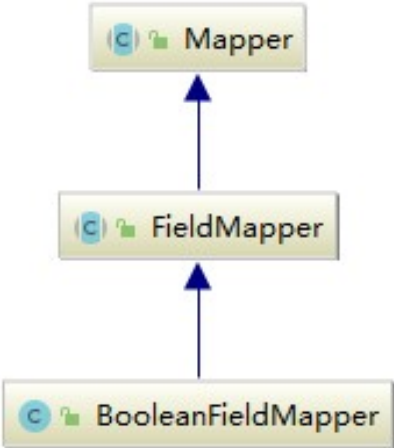
创建Lucene::Field

```
if (fieldType().indexOptions() != IndexOptions.NONE || fieldType().stored()) {  
    fields.add(new Field(fieldType().name(), value ? "T" : "F", fieldType()));  
}
```

若设置了docvalue

`doc_values` Should the field be stored on disk in a column-stride fashion, so that it can later be used for sorting, aggregations, or scripting? Accepts `true` (default) or `false`.

```
if (fieldType().hasDocValues()) {  
    fields.add(new SortedNumericDocValuesField(fieldType().name(), value ? 1 : 0));  
} else {  
    createFieldNamesField(context, fields); context: ParseContext$InternalParseCont  
}
```



FieldMapper解析流程: NestedField

创建NestedContext

获取nestedDoc与parentDoc

```
private final List<Document> documents;
```

parentDoc与nestedDoc在各自被创建时，依次添加进documents中

```
context = context.createNestedContext(mapper.fullPath());
ParseContext.Document nestedDoc = context.doc(); nestedDoc
ParseContext.Document parentDoc = nestedDoc.getParent();
```

设置nestedDoc的_id为parentDocId（方便统一删除）

```
// We just need to store the id as indexed field, so that IndexWriter#deleteDocuments(term) can then
// delete it when the root document is deleted too.
nestedDoc.add(new Field(IdFieldMapper.NAME, idField.binaryValue(), IdFieldMapper.Defaults.NESTED_FIELD_TYPE));
```

设置nestedDoc的_type字段为__XXX（以__开头，方便过滤）

```
nestedDoc.add(new Field(TypeFieldMapper.NAME, mapper.nestedTypePathAsString(), TypeFieldMapper.Defaults.FIELD_TYPE));
```

依次解析每个内部字段，生成对应Field并放入nestedDoc中

nestedDoc

```
document = {ParseContext$Document@11359}
> parent = {ParseContext$Document@11192}
> path = "user"
> prefix = "user."
> fields = {ArrayList@11385} size = 2
  0 = {Field@11389} "indexed,omitNorms,indexOptions=DOCS<_id:[fe 1f]>"
  1 = {Field@11390} "indexed,omitNorms,indexOptions=DOCS<_type:_user>"
keyedFields = null
```

```
"user": {
  "type": "nested",
  "properties": {
    "first": {
      "type": "keyword"
    },
    "last": {
      "type": "keyword"
    }
  }
}
```

```
p mapper = {ObjectMapper@10935}
> fullPath = "user"
> enabled = true
> nested = {ObjectMapper$Nested@11348}
> nestedTypePathAsString = "__user"
> nestedTypePathAsBytes = {BytesRef@11350} "[5f 5f 75 73 65 72]"
> nestedTypeFilter = {TermQuery@11351} "_type:_user"
> dynamic = null
> mappers = {CopyOnWriteHashMap@11352} size = 2
  0 = {AbstractMap$SimpleImmutableEntry@11374} "last" ->
    key = "last"
    value = {KeywordFieldMapper@11378}
  1 = {AbstractMap$SimpleImmutableEntry@11375} "first" ->
    key = "first"
    value = {KeywordFieldMapper@11378}
> simpleName = "user"
```

重排序，保证子文档出现在父文档前面

```
void postParse() {
    if (documents.size() > 1) {
        docsReversed = true;
        if (indexSettings.getIndexVersionCreated().onOrAfter(Version.V_6_5_0)) {
            /**
             * For indices created on or after {@link Version#V_6_5_0} we preserve the order
             * of the children while ensuring that parents appear after them.
             */
            List<Document> newDocs = reorderParent(documents);
            documents.clear();
            documents.addAll(newDocs);
        } else {
            // reverse the order of docs for nested docs support, parent should be last
            Collections.reverse(documents);
        }
    }
}
```

保留父文档、子文档之间的相对顺序

```
{
  "user": {
    "type": "nested",
    "properties": {
      "first": {
        "type": "keyword"
      },
      "last": {
        "type": "keyword"
      }
    }
  }
}
```

```
private List<Document> reorderParent(List<Document> docs) {
    List<Document> newDocs = new ArrayList<>(docs.size());
    LinkedList<Document> parents = new LinkedList<>();
    for (Document doc : docs) {
        while (parents.peek() != doc.getParent()) {
            newDocs.add(parents.poll());
        }
        parents.add(index: 0, doc);
    }
    newDocs.addAll(parents);
    return newDocs;
}
```

nestedDoc01

nestedDoc02

parentDoc

```
doc = {ParsedDocument@11777} "Document id[2] doc [[org.elasticsearch.index.mapper.Parse
> {f version = {NumericDocValuesField@11779} "docValuesType=NUMERIC<-1>"
> {f id = "2"
> {f type = "_doc"
> {f seqID = {SeqNoFieldMapper$SequenceIDFields@11781}
  {f routing = null
> {f documents = {ArrayList@11782} size = 3
  > 0 = {ParseContext$Document@11787}
    > {f parent = {ParseContext$Document@11789}
    > {f path = "user"
    > {f prefix = "user."
    > {f fields = {ArrayList@11791} size = 9
      {f keyedFields = null
  > 1 = {ParseContext$Document@11788}
    > {f parent = {ParseContext$Document@11789}
    > {f path = "user"
    > {f prefix = "user."
    > {f fields = {ArrayList@11793} size = 9
      {f keyedFields = null
  > 2 = {ParseContext$Document@11789}
    {f parent = null
    > {f path = ""
    > {f prefix = ""
    > {f fields = {ArrayList@11795} size = 16
      {f keyedFields = null
  > {f source = {ByteArray@11783}
  > {f xContentType = {XContentType$1@11049} "JSON"
  {f dynamicMappingsUpdate = null
```

KeywordField

获取keyword值

获取field normalizer

标准化keyword

添加Field至Document

添加docvalue至Document

```
final NamedAnalyzer normalizer = fieldType().normalizer();
if (normalizer != null) {
    try (TokenStream ts = normalizer.tokenStream(name(), value)) {
        final CharTermAttribute termAtt = ts.addAttribute(CharTermAttribute.class);
        ts.reset();
        if (ts.incrementToken() == false) {...}
        final String newValue = termAtt.toString();
        if (ts.incrementToken()) {...}
        ts.end();
        value = newValue;
    }
}

// convert to utf8 only once before feeding postings/dv/stored fields
final BytesRef binaryValue = new BytesRef(value);
if (fieldType().indexOptions() != IndexOptions.NONE || fieldType().stored()) {
    Field field = new Field(fieldType().name(), binaryValue, fieldType());
    fields.add(field);

    if (fieldType().hasDocValues() == false && fieldType().omitNorms()) {
        createFieldNamesField(context, fields);
    }
}

if (fieldType().hasDocValues()) {
    fields.add(new SortedSetDocValuesField(fieldType().name(), binaryValue));
}
```

KeywordField相关查询

判断字段是否为空

```
public Query existsQuery(QueryShardContext context) {
    if (hasDocValues()) {
        return new DocValuesFieldExistsQuery(name());
    } else if (omitNorms()) {
        return new TermQuery(new Term(FieldNamesFieldMapper.NAME, name()));
    } else {
        return new NormsFieldExistsQuery(name());
    }
}
```

模糊查询

```
failIfNotIndexed();
return new FuzzyQuery(new Term(name(), indexedValueForSearch(value)),
    fuzziness.asDistance(BytesRefs.toString(value)), prefixLength, maxExpansions, transpositions);
```

通配符查询

范围查询

Terms查询

```
public Query termsQuery(List<?> values, QueryShardContext context) {
    failIfNotIndexed();
    BytesRef[] bytesRefs = new BytesRef[values.size()];
    for (int i = 0; i < bytesRefs.length; i++) {
        bytesRefs[i] = indexedValueForSearch(values.get(i));
    }
    return new TermInSetQuery(name(), bytesRefs);
}
```

前缀查询

```
failIfNotIndexed();
PrefixQuery query = new PrefixQuery(new Term(name(), indexedValueForSearch(value)));
if (method != null) {
    query.setRewriteMethod(method);
}
return query;
```

正则查询

Term查询

Arrays

```
{
  "activebool": [true, false, true]
}
```

```
> document = {ParseContext$Document@10271}
  f parent = null
> f path = ""
> f prefix = ""
✓ f fields = {ArrayList@10345} size = 13
  > # 0 = {Field@10277} "stored,indexed,omitNorms,indexOptions=DOCS<_id:[fe 3f]>"
  > # 1 = {LongPoint@10347} "LongPoint <_seq_no:-2>"
  > # 2 = {NumericDocValuesField@10348} "docValuesType=NUMERIC<_seq_no:-2>"
  > # 3 = {NumericDocValuesField@10349} "docValuesType=NUMERIC<_primary_term:0>"
  > # 4 = {StoredField@10350} "stored<_recovery_source:[7b a 20 20 20 20 22 61 63 74 69 76 65 62 66 66 23 23]"
  > # 5 = {NumericDocValuesField@10351} "docValuesType=NUMERIC<_recovery_source:1>"
  > # 6 = {NumericDocValuesField@10339} "docValuesType=NUMERIC<_version:-1>"
  ✓ # 7 = {Field@10352} "indexed,omitNorms,indexOptions=DOCS<activebool:T>"
    > f type = {BooleanFieldMapper$BooleanFieldType@10325} "indexed,omitNorms,indexOptions=DOCS"
    > f name = "activebool"
    > f fieldsData = "T"
      f tokenStream = null
  ✓ # 8 = {SortedNumericDocValuesField@10353} "docValuesType=SORTED_NUMERIC<activebool:1>"
    > f type = {FieldType@10379} "docValuesType=SORTED_NUMERIC"
    > f name = "activebool"
    > f fieldsData = {Long@10380} 1
      f tokenStream = null
  > # 9 = {Field@10354} "indexed,omitNorms,indexOptions=DOCS<activebool:F>"
  > # 10 = {SortedNumericDocValuesField@10355} "docValuesType=SORTED_NUMERIC<activebool:0>"
  ✓ # 11 = {Field@10389} "indexed,omitNorms,indexOptions=DOCS<activebool:T>"
    > f type = {BooleanFieldMapper$BooleanFieldType@10325} "indexed,omitNorms,indexOptions=DOCS"
    > f name = "activebool"
    > f fieldsData = "T"
      f tokenStream = null
  ✓ # 12 = {SortedNumericDocValuesField@10390} "docValuesType=SORTED_NUMERIC<activebool:1>"
    > f type = {FieldType@10379} "docValuesType=SORTED_NUMERIC"
    > f name = "activebool"
    > f fieldsData = {Long@10380} 1
      f tokenStream = null
  f keyedFields = null
```

相当于“
“TF T”不
docvalue

即使单个值也会存docvalue (activebool)

相当于“TF T”与 docvalue[1, 0, 1]两个字段
“TF T”存于倒排索引，用于检索
docvalue[1, 0, 1]用于排序和聚类

即使单个值也会存docvalue (activebool:true)

Array与Object

```
{
  "activeobject": [
    {
      "offset": 0,
      "enable": true
    },
    {
      "offset": 10,
      "enable": false
    }
  ]
}
```

```
{
  "activeobject.offset": [0, 10],
  "activeobject.enable": [true, false]
}
```

```

▼ f documents = {ArrayList@10898} size = 1
  ▼ 0 = {ParseContext$Document@10897}
    f parent = null
    > f path = ""
    > f prefix = ""
    ▼ f fields = {ArrayList@10906} size = 15
      > 0 = {Field@10908} "stored,indexed,omitNorms,indexOptions=DOCS<_id:[fe 3f]>"
      > 1 = {LongPoint@10909} "LongPoint <_seq_no:-2>"
      > 2 = {NumericDocValuesField@10910} "docValueType=NUMERIC<_seq_no:-2>"
      > 3 = {NumericDocValuesField@10911} "docValueType=NUMERIC<_primary_term:0>"
      > 4 = {StoredField@10912} "stored<_recovery_source:[7b a 20 20 20 22 61 63 74 69 76 65 6f 62 6a 65 63 74 6e 6d 6c 6b 6a 69 68 67 66 65 64 63 62 61 60 5f 5e 5d 5c 5b 5a 59 58 57 56 55 54 53 52 51 50 4f 4e 4d 4c 4b 4a 49 48 47 46 45 44 43 42 41 40 3f 3e 3d 3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f 1e 1d 1c 1b 1a 19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01 00]>"
      > 5 = {NumericDocValuesField@10913} "docValueType=NUMERIC<_recovery_source:1>"
      > 6 = {NumericDocValuesField@10899} "docValueType=NUMERIC<_version:-1>"
      ▼ 7 = {LongPoint@10914} "LongPoint <activeobject.offset:0>"
        > f type = {FieldType@10937} "pointDimensionCount=1,pointIndexDimensionCount=1,pointNumBytes=8"
        > f name = "activeobject.offset"
        > f fieldsData = {BytesRef@10939} "[80 0 0 0 0 0 0 0]"
        f tokenStream = null
      ▼ 8 = {SortedNumericDocValuesField@10915} "docValueType=SORTED_NUMERIC<activeobject.offset:0>"
        > f type = {FieldType@10379} "docValueType=SORTED_NUMERIC"
        > f name = "activeobject.offset"
        > f fieldsData = {Long@10374} 0
        f tokenStream = null
      ▼ 9 = {Field@10916} "indexed,omitNorms,indexOptions=DOCS<activeobject.enable:T>"
        > f type = {BooleanFieldMapper$BooleanFieldType@10946} "indexed,omitNorms,indexOptions=DOCS"
        > f name = "activeobject.enable"
        > f fieldsData = "T"
        f tokenStream = null
      ▼ 10 = {SortedNumericDocValuesField@10917} "docValueType=SORTED_NUMERIC<activeobject.enable:1>"
        > f type = {FieldType@10379} "docValueType=SORTED_NUMERIC"
        > f name = "activeobject.enable"
        > f fieldsData = {Long@10380} 1
        f tokenStream = null
      > 11 = {LongPoint@10918} "LongPoint <activeobject.offset:10>"
      > 12 = {SortedNumericDocValuesField@10919} "docValueType=SORTED_NUMERIC<activeobject.offset:10>"
      > 13 = {Field@10920} "indexed,omitNorms,indexOptions=DOCS<activeobject.enable:F>"
      > 14 = {SortedNumericDocValuesField@10921} "docValueType=SORTED_NUMERIC<activeobject.enable:0>"
      f keyedFields = null

```

相当于4个lucene字段:

[0, 10] longpoint	--- kdtree
[0, 10] docvalue	--- docvalue
"T F" text	--- inverted index
[1, 0] docvalue	--- docvalue

可以根据term搜出文档，
但是数组字段间相对顺序无法保证

```
{
  "keyword": ["A", "B"]
}
```

```

f fields = {ArrayList@12933} size = 10
  > # 0 = {Field@12943} "stored,indexed,omitNorms,indexOptions=DOCS<_id:[fe 1f]>"
  > # 1 = {LongPoint@12944} "LongPoint <_seq_no:-2>"
  > # 2 = {NumericDocValuesField@12945} "docValuesType=NUMERIC<_seq_no:-2>"
  > # 3 = {NumericDocValuesField@12946} "docValuesType=NUMERIC<_primary_term:0>"
  > # 4 = {StoredField@12947} "stored<_source:[7b a 20 20 20 22 6b 65 79 77 6f 72 64 22 3a 20 5b 22 41 22 2c 20 22 42 22 5d a 7d]>"
  > # 5 = {NumericDocValuesField@12937} "docValuesType=NUMERIC<_version:-1>"
  v # 6 = {Field@12948} "indexed,omitNorms,indexOptions=DOCS<keyword:[41]>"
    > f type = {KeywordFieldMapper$KeywordFieldType@12381} "indexed,omitNorms,indexOptions=DOCS"
    > f name = "keyword"
    > f fieldsData = {BytesRef@12962} "[41]"
    f tokenStream = null
    v # 7 = {SortedSetDocValuesField@12949} "docValuesType=SORTED_SET<keyword:[41]>"
      > f type = {FieldType@12965} "docValuesType=SORTED_SET"
      > f name = "keyword"
      > f fieldsData = {BytesRef@12962} "[41]"
      f tokenStream = null
    > # 8 = {Field@12950} "indexed,omitNorms,indexOptions=DOCS<keyword:[42]>"
    > # 9 = {SortedSetDocValuesField@12951} "docValuesType=SORTED_SET<keyword:[42]>"
  f keyedFields = null

```

```

f stored = false
f tokenized = false
f storeTermVectors = false
f storeTermVectorOffsets = false
f storeTermVectorPositions = false
f storeTermVectorPayloads = false
f omitNorms = true
> f indexOptions = {IndexOptions@12387} "DOCS"
  f frozen = true
> f docValuesType = {DocValuesType@12817} "NONE"
  f dimensionCount = 0
  f indexDimensionCount = 0
  f dimensionNumBytes = 0

```

```

# 7 = {SortedSetDocValuesField@12949} "docValuesType=SORTED_SET"
  v f type = {FieldType@12965} "docValuesType=SORTED_SET"
    f stored = false
    f tokenized = true
    f storeTermVectors = false
    f storeTermVectorOffsets = false
    f storeTermVectorPositions = false
    f storeTermVectorPayloads = false
    f omitNorms = false
  > f indexOptions = {IndexOptions@12972} "NONE"
    f frozen = true
  > f docValuesType = {DocValuesType@12481} "SORTED_SET"
    f dimensionCount = 0
    f indexDimensionCount = 0

```

Nested Docvalue方案

需要建立字段间的连接关系，用于检索

保留docvalue，用于排序、聚类

需要建立字段间的连接关系，用于对象检索

```
{
  "tagkey": [
    {
      "value": "football",
      "weight": 0.2
    },
    {
      "value": "running",
      "weight": 0.1
    }
  ]
}
```

ES存储方式:

tagkey.value: “football running” <inverted>

tagkey.value: [football running] <docvalue>

tagkey.weight: [0.2, 0.1] <floatpoint>

tagkey.weight: [0.2, 0.1] <docvalue>

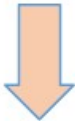
无法独立查找每个子对象

Nested改造

```
{
  "tagkey": {
    "type": "nested",
    "properties": {
      "value": {
        "type": "keyword"
      },
      "weight": {
        "type": "double"
      }
    }
  }
}
```

```
{
  "tagkey": {
    "properties": {
      "value": {
        "type": "multiKeyword"
      },
      "weight": {
        "type": "double"
      }
    }
  }
}
```

docid	Nested.value	Nested.weight	NotNested
100	football	0.2	
101	running	0.1	
102			xxx



docid	Nested.value	Nested.weight	NotNested
100	Football Running	0.2	xxx

方案一：DocValue过滤

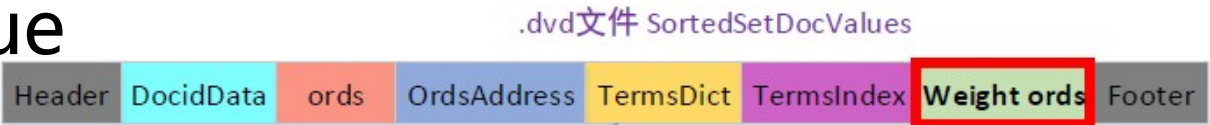
mapping
<pre>{ "tagkey": { "type": "relation_object", "relations": ["value", "weight", "enable"], "properties": { "value": { "type": "keyword" }, "weight": { "type": "double" }, "enable": { "type": "boolean" } } } }</pre>

_source
<pre>{ "tagkey": [{ "value": "football", "weight": 0.2, "enable": true }, { "value": "running", "weight": 0.1, "enable": false }] }</pre>

fields
<pre>tagkey.value: "football running" <inverted> IndexOption<doc> tagkey.value: ["football", "running"] <docvalue> tagkey.weight: [0.2, 0.1] <floatpoint> tagkey.weight: [0.2, 0.2] <docvalue> tagkey.enable: "T F" <inverted> IndexOption<doc> tagkey.enable: [1, 0] <docvalue> tagkey.relations: ["football#0.2#T, running#0.1#F"] <docvalue></pre>

根据value、weight查找出对应文档（此时包含部分不严格匹配的文档）
Lucene层提供RelationObjectCollector，根据relations字段对文档进行过滤

方案二：带权DocValue



TermsDict	
football	0
running	1
basketball	2

```
mapping
{
  "tagkey": {
    "type": "relation_object",
    "properties": {
      "value": {
        "type": "weighted_keyword",
      },
      "enable": {
        "type": "boolean"
      }
    }
  }
}
```

fields	
tagkey.value: "football running"	<inverted> IndexOption<doc>
tagkey.value: ["football",0.2], ["running",0.1]	<weighted_docvalue>
tagkey.enable: "T F"	<inverted> IndexOption<doc>
tagkey.enable: [1, 0]	<docvalue>

Lucene层提供带权DocValue实现和对应Collector，根据权重进行文档过滤

```
_source
{
  "tagkey": [
    {
      "value": {
        "value": "football",
        "weight": 0.2
      },
      "enable": true
    },
    {
      "value": {
        "value": "running",
        "weight": 0.1
      },
      "enable": false
    }
  ]
}
```

方案三：ES层实现文档过滤

mapping

```
{
  "tagkey": {
    "type": "object",
    "properties": {
      "value": {
        "type": "keyword"
      },
      "weight": {
        "type": "double"
      },
      "enable": {
        "type": "boolean"
      }
    }
  }
}
```

_source

```
{
  "tagkey": [
    {
      "value": {
        "value": "football",
        "weight": 0.2
      },
      "enable": true
    },
    {
      "value": {
        "value": "running",
        "weight": 0.1
      },
      "enable": false
    }
  ]
}
```

query

```
{
  "query": {
    "bool": {
      "must": {
        "term": { "value": "football" }
      },
      "must": {
        "range": {
          "weight": { "gte": 0.1, "lte": 0.3 }
        }
      }
    },
    "relation_filter": {
      "fields": ["value", "weight"]
    }
  }
}
```

先根据must查出文档，再使用relation_filter过滤

转移具体实现到Lucene

方法一：在dvd文件中增加Weight ords数组，与ord数组一一对应，标识每个ord的权重

在ES层完成设计实现

方法二：将权重也保存成docvalue多值，SortedSetDocValuesField原生是文档内部排序，可能会乱序，要考虑其对应关系

```
doc = new Document();
doc.add(new SortedSetDocValuesField("Field", new BytesRef("Football")));
doc.add(new SortedSetDocValuesField("Field", new BytesRef("Running")));
doc.add(new SortedSetDocValuesField("Weight", new BytesRef("0.2")));
doc.add(new SortedSetDocValuesField("Weight", new BytesRef("0.1")));
indexWriter.addDocument(doc);
```

转移具体实现到Lucene

方法三：将权重和列值一致保存成多值

```
doc = new Document();
doc.add(new SortedSetDocValuesField("Field", new BytesRef("Football#0.2")));
doc.add(new SortedSetDocValuesField("Field", new BytesRef("Running#0.1")));
indexWriter.addDocument(doc);
```

```
public class BinaryDocValuesField extends Field {

    /**
     * Type for straight bytes DocValues.
     */
    public static final FieldType TYPE = new FieldType();
    static {
        TYPE.setDocValuesType(DocValuesType.BINARY);
        TYPE.freeze();
    }
}

public class NumericDocValuesField extends Field {

    /**
     * Type for numeric DocValues.
     */
    public static final FieldType TYPE = new FieldType();
    static {
        TYPE.setDocValuesType(DocValuesType.NUMERIC);
        TYPE.freeze();
    }
}
```

```

public enum DocValueType {
    /**
     * No doc values for this field.
     */
    NONE,
    /**
     * A per-document Number
     */
    NUMERIC,
    /**
     * A per-document byte[]. Values may be larger than
     * 32766 bytes, but different codecs may enforce their own limits.
     */
    BINARY,
    /**
     * A pre-sorted byte[]. Fields with this type only store distinct byte values
     * and store an additional offset pointer per document to dereference the shared
     * byte[]. The stored byte[] is presorted and allows access via document id,
     * ordinal and by-value. Values must be {@code <= 32766} bytes.
     */
    SORTED,
    /**
     * A pre-sorted Number[]. Fields with this type store numeric values in sorted
     * order according to {@link Long#compare(long, long)}.
     */
    SORTED_NUMERIC,
    /**
     * A pre-sorted Set<byte[]>. Fields with this type only store distinct byte values
     * and store additional offset pointers per document to dereference the shared
     * byte[]s. The stored byte[] is presorted and allows access via document id,
     * ordinal and by-value. Values must be {@code <= 32766} bytes.
     */
    SORTED_SET,
}

```

保留写入顺序?