BooleanQuery介绍

BooleanQuery常用来对实现多个Query子类对象的进行组合，这些Query子类对象会组成一个Cluase实现组合查询。每一个Query都有四种可选，分别描述了匹配的文档需要满足的要求，定义在BooleanClause类中，如下：

```
public static enum Occur {
    /** Use this operator for clauses that <i>must</i> appear in the matching documents. */
    MUST     { @Override public String toString() { return "+"; } },
    /** Like {@link #MUST} except that these clauses do not participate in scoring. */
    FILTER   { @Override public String toString() { return "#"; } },
    /** Use this operator for clauses that <i>should</i> appear in the
     * matching documents. For a BooleanQuery with no <code>MUST</code>
     * clauses one or more <code>SHOULD</code> clauses must match a document
     * for the BooleanQuery to match.
     * @see BooleanQuery.Builder#setMinimumNumberShouldMatch
     */
    SHOULD   { @Override public String toString() { return "";  } },
    /** Use this operator for clauses that <i>must not</i> appear in the matching documents.
     * Note that it is not possible to search for queries that only consist
     * of a <code>MUST_NOT</code> clause. These clauses do not contribute to the
     * score of documents. */
    MUST_NOT { @Override public String toString() { return "-"; } };
 }
```

MUST (+)

满足查询要求的文档中必须包含查询的关键字。

SHOULD (" ")

满足查询要求的文档中包含一个或多个查询的关键字。

FILTER (#)

满足查询要求的文档中必须包含查询的关键字，但是这个Query不会参与文档的打分。

MUST_NOT (-)

满足查询要求的文档中必须不能包含查询的关键字。

组合查询

```
例子：“+a b -c d”
```

转化为代码如下：

```
BooleanQuery.Builder query = new BooleanQuery.Builder();
query.add(new TermQuery(new Term("content", "a")),
BooleanClause.Occur.MUST);
query.add(new TermQuery(new Term("content", "b")),
BooleanClause.Occur.SHOULD);
query.add(new TermQuery(new Term("content", "c")),
BooleanClause.Occur.MUST_NOT);
query.add(new TermQuery(new Term("content", "d")),
BooleanClause.Occur.SHOULD);
```

满足查询要求的文档必须包含 "a",，不能包含“c”,，可以包含“b”，“d”中一个或者多个，包含的越多，文档的分数越高

BooleanQuery的方法

设置minimumNumberShouldMatch

```
 public Builder setMinimumNumberShouldMatch(int min) {
     ......
    }
```

当查询有多个SHOULD的Query对象时，满足查询要求的文档中必须包含minimumNumberShouldMatch个Query的关键字

构建CreateWeight对象

```
public Weight createWeight(IndexSearcher searcher, boolean needsScores,
float boost) throws IOException {
    ...
    return new BooleanWeight(query, searcher, needsScores, boost);
  }
```

Query对象的子类都会重写这个方法。对于BooleanQuery的createWeight(...)实现，只是调用了对象组合中的所有Query子类的createWeight(...)方法分别生成Weight对象，然后将这些对象封装到BooleanWeight对象中。TermQuery的createWeight()的具体实现看博客的文章 TermQuery。

重写Query

```
public Query rewrite(IndexReader reader) throws IOException {
    ...
  }
```

BooleanQuery的rewrite(..)跟createWeight(...)相同的是都是调用对象组合中所有Query子类的rewrite(...)方法，但是并不是所有的Query都需要重写。比如TermQuery，他就没有重写父类的rewrite(...)方法，而对于PrefixQuery(前缀查询)，则必须要重写, 重写后的PrefixQuery会生成多个TermQuery，最后组合成BooleanQuery。

```
例子：前缀查询关键字  "ca*", 重写后，会变成 "car", "cat", ...每一个关键字作为TermQuery，组合成BooleanQuery进行查询，所以一般都禁用PreFixQuery，容易抛出TooManyClause的异常。
```

BooleanQuery的rewrite(...)实现中一共有9个逻辑(下面的会对每一种逻辑进行标注，比如说 逻辑一), 根据BooleanQuery中的不同的组合(MUST, SHOULD, MUST_NOT, FILTER的任意组合), 会至少执行1个多个重写逻辑，我们对最常用的组合来描述重写的过程。

只有一个SHOULD或MUST的TermQuery

重写第一步

直接返回。。。不需要重写。

```
// 逻辑一
if (clauses.size() == 1) {
      BooleanClause c = clauses.get(0);
      Query query = c.getQuery();
      if (minimumNumberShouldMatch == 1 && c.getOccur() == Occur.SHOULD) {
        return query;
      } else if (minimumNumberShouldMatch == 0) {
        switch (c.getOccur()) {
          case SHOULD:
          case MUST:
          // 直接返回原Query。
            return query;
          case FILTER:
            // no scoring clauses, so return a score of 0
            return new BoostQuery(new ConstantScoreQuery(query), 0);
          case MUST_NOT:
            // no positive clauses
            return new MatchNoDocsQuery("pure negative BooleanQuery");
          default:
            throw new AssertionError();
        }
      }
```

```
    }
```

多个SHOULD的TermQuery

重写第一步

首先遍历BooleanQuery中的所有Query对象，调用他们自身的重写方法，由于TermQuery不需要重写，所以直接返回自身。

```
// 逻辑二
{
    // 重新生成一个BooleanQuery的构建器，准备对重写后的Query进行组合。
    BooleanQuery.Builder builder = new BooleanQuery.Builder();
    // 设置一样的MinimumNumberShouldMatch。
    builder.setMinimumNumberShouldMatch(getMinimumNumberShouldMatch());
    boolean actuallyRewritten = false;
    for (BooleanClause clause : this) {
      Query query = clause.getQuery();
      // 调用Query子类的rewrite(...)方法
      // 我们的例子中都是TermQuery，所以直接返回自身this。
      Query rewritten = query.rewrite(reader);
      if (rewritten != query) {
        actuallyRewritten = true;
      }
      builder.add(rewritten, clause.getOccur());
    }
    // 由于我们例子中的的BooleanQuery的Query子类都是TermQuery，不需要重写，所以就不用生成新的BooleanQuery对象
    if (actuallyRewritten) {
      return builder.build();
    }
}
```

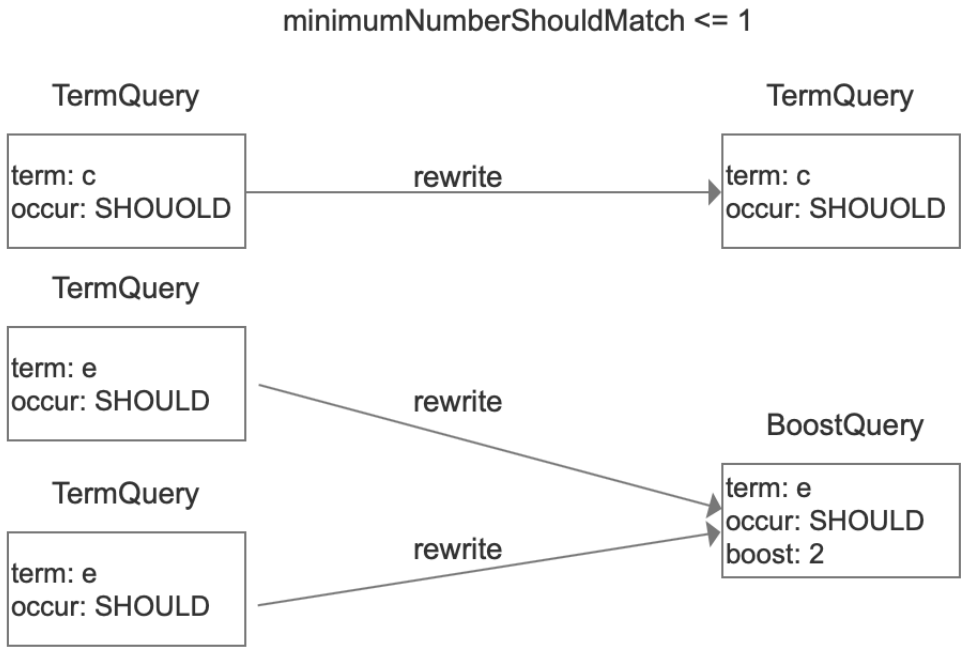
重写第二步(可选)

如果minimumNumberShouldMatch的值 <= 1那么需要执行第二步。 当有多个相同的TermQuery，并且是SHOULD，会将这些相同的TermQuery封住成一个BoostQuery，增加boost的值。

```
// 逻辑七
// 这段代码的逻辑跟 逻辑八 一毛一样，往下找一找^-^,就不赘述了
if (clauseSets.get(Occur.SHOULD).size() > 0 && minimumNumberShouldMatch <= 1) {
      Map<Query, Double> shouldClauses = new HashMap<>();
      for (Query query : clauseSets.get(Occur.SHOULD)) {
        double boost = 1;
        while (query instanceof BoostQuery) {
          BoostQuery bq = (BoostQuery) query;
          boost *= bq.getBoost();
```

```
10            query = bq.getQuery();
11          }
12          shouldClauses.put(query, shouldClauses.getOrDefault(query, 0d) +
boost);
13        }
14        if (shouldClauses.size() != clauseSets.get(Occur.SHOULD).size()) {
15          BooleanQuery.Builder builder = new BooleanQuery.Builder()
16              .setMinimumNumberShouldMatch(minimumNumberShouldMatch);
17          for (Map.Entry<Query,Double> entry : shouldClauses.entrySet()) {
18            Query query = entry.getKey();
19            float boost = entry.getValue().floatValue();
20            if (boost != 1f) {
21              query = new BoostQuery(query, boost);
22            }
23            builder.add(query, Occur.SHOULD);
24          }
25          for (BooleanClause clause : clauses) {
26            if (clause.getOccur() != Occur.SHOULD) {
27              builder.add(clause);
28            }
29          }
30          return builder.build();
31        }
32      }
```

下图中左边是重写前的BooleanQuery，右边是重写后的BooleanQuery。



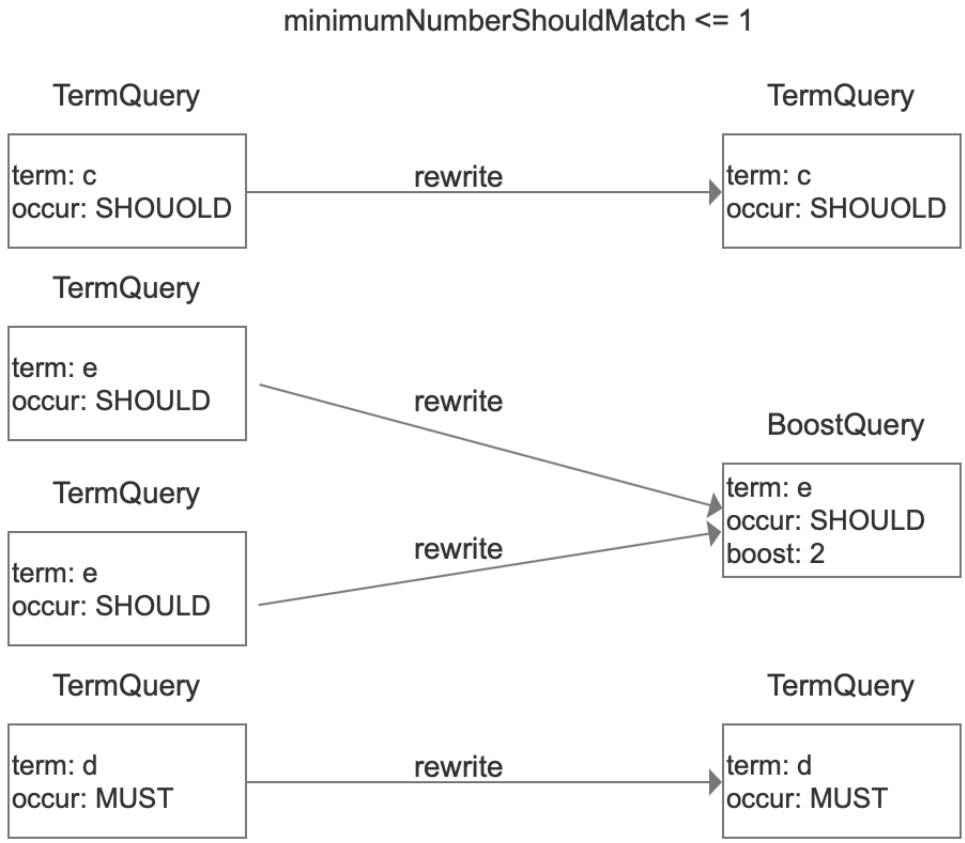
SHOULD(至少一个)和MUST(至少一个)的TermQuery

重写第一步

同样先要执行 逻辑二，不赘述。

重写第二步(可选)

同样先要执行 逻辑七，不赘述。 下图中左边是重写前的BooleanQuery，右边是重写后的BooleanQuery。



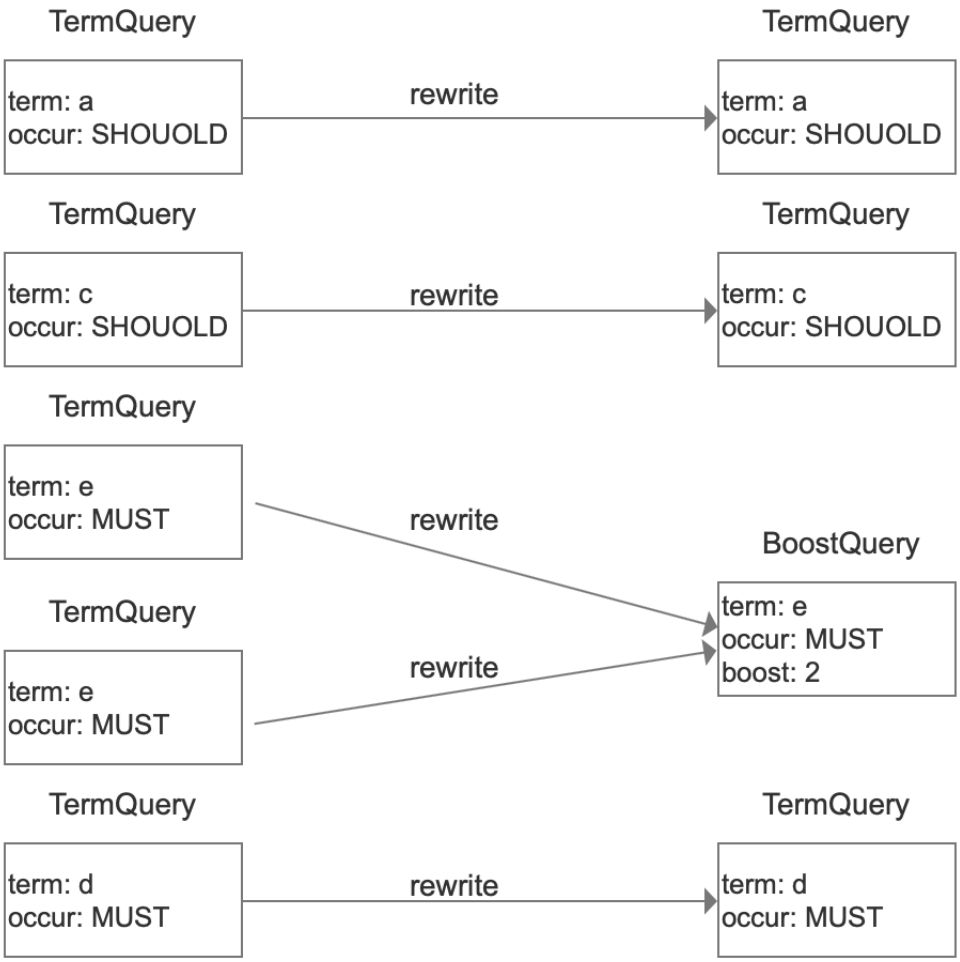
重写第三步

当有多个相同的TermQuery，并且是MUST，会将这些相同的TermQuery封住成一个BoostQuery，增加boost的值。

```
// 逻辑八
if (clauseSets.get(Occur.MUST).size() > 0) {
      Map<Query, Double> mustClauses = new HashMap<>();
      // 这里遍历所有的MUST的Clause，如果有重复的Clause，boost值就加1，描述了这个关键字的重要性
      for (Query query : clauseSets.get(Occur.MUST)) {
        double boost = 1;
        while (query instanceof BoostQuery) {
          BoostQuery bq = (BoostQuery) query;
          boost *= bq.getBoost();
          query = bq.getQuery();
        }
        // 调用getOrDefault()查看是否有相同的clause，如果有，那么取出boost，然后对boost进行+1后，覆盖已经存在的clause。
```

```
13          mustClauses.put(query, mustClauses.getOrDefault(query, 0d) +
boost);
14        }
15        // 运行至此，如果BooleanQuery有相同的query，并且是MUST，那么将这些MUST的
query合并为一个query，并且增加boost的值。
16        // if语句为true：说明有重复的clause(MUST)，那么需要对boost不等于1的query重
写，然后跟其他的query一起写到新的BooleanQuery中。
17        if (mustClauses.size() != clauseSets.get(Occur.MUST).size()) {
18          BooleanQuery.Builder builder = new BooleanQuery.Builder()
19                  .setMinimumNumberShouldMatch(minimumNumberShouldMatch);
20          // 这个for循环是将那些boost值不等于1的query重写为BoostQuery。
21          for (Map.Entry<Query,Double> entry : mustClauses.entrySet()) {
22            Query query = entry.getKey();
23            float boost = entry.getValue().floatValue();
24            if (boost != 1f) {
25              // 重写为BoostQuery。
26              query = new BoostQuery(query, boost);
27            }
28            builder.add(query, Occur.MUST);
29          }
30          // 把其他不是MUST的clause重写添加到新的BooleanQuery中。
31          for (BooleanClause clause : clauses) {
32            if (clause.getOccur() != Occur.MUST) {
33              builder.add(clause);
34            }
35          }
36          return builder.build();
37        }
38      }
```

下图中左边是重写前的BooleanQuery，右边是重写后的BooleanQuery。



多个MUST的TermQuery

重写第一步

同样先要执行 逻辑二，不赘述。

重写第二步

当有多个相同的TermQuery，并且是MUST，会将这些相同的TermQuery封住成一个BoostQuery，增加boost的值。然后执行逻辑八，已说明，不赘述。

MUST(至少一个)和MUST_NOT(至少一个)的TermQuery

重写第一步

同样先要执行 逻辑二，不赘述。

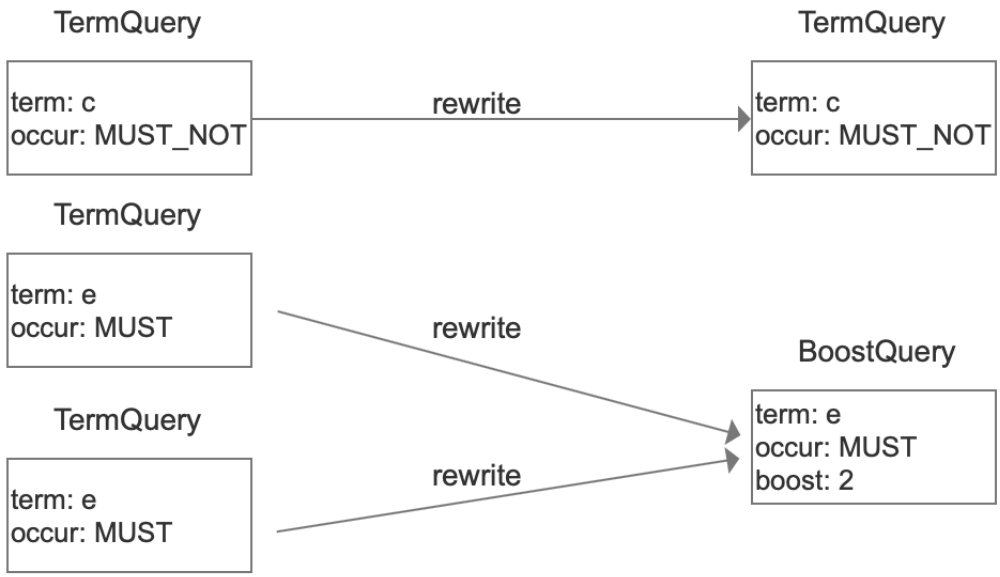
重写第二步

如果在这个逻辑中返回了，那么就会返回一个MatchNoDocsQuery对象，也就是不会搜索到任何结果。

```
// 逻辑四
 final Collection<Query> mustNotClauses = clauseSets.get(Occur.MUST_NOT);
    if (!mustNotClauses.isEmpty()) {
      final Predicate<Query> p = clauseSets.get(Occur.MUST)::contains;
      // 判断是否MUST_NOT跟MUST或FILTER是否有相同的term
      if (mustNotClauses.stream().anyMatch(p.or(clauseSets.get(Occur.FILTER)::contains)))     {
        return new MatchNoDocsQuery("FILTER or MUST clause also in MUST_NOT");
      }
       // 判断是否有MatchAllDocsQuery的Query
      if (mustNotClauses.contains(new MatchAllDocsQuery())) {
        return new MatchNoDocsQuery("MUST_NOT clause is MatchAllDocsQuery");
      }
    }
```

重写第三步

当有多个相同的TermQuery，并且是MUST，会将这些相同的TermQuery封住成一个BoostQuery，增加boost的值。然后执行逻辑八，已说明，不赘述。 下图中左边是重写前的BooleanQuery，右边是重写后的BooleanQuery。



SHOULD(至少一个)和MUST_NOT(至少一个)的TermQuery

重写第一步

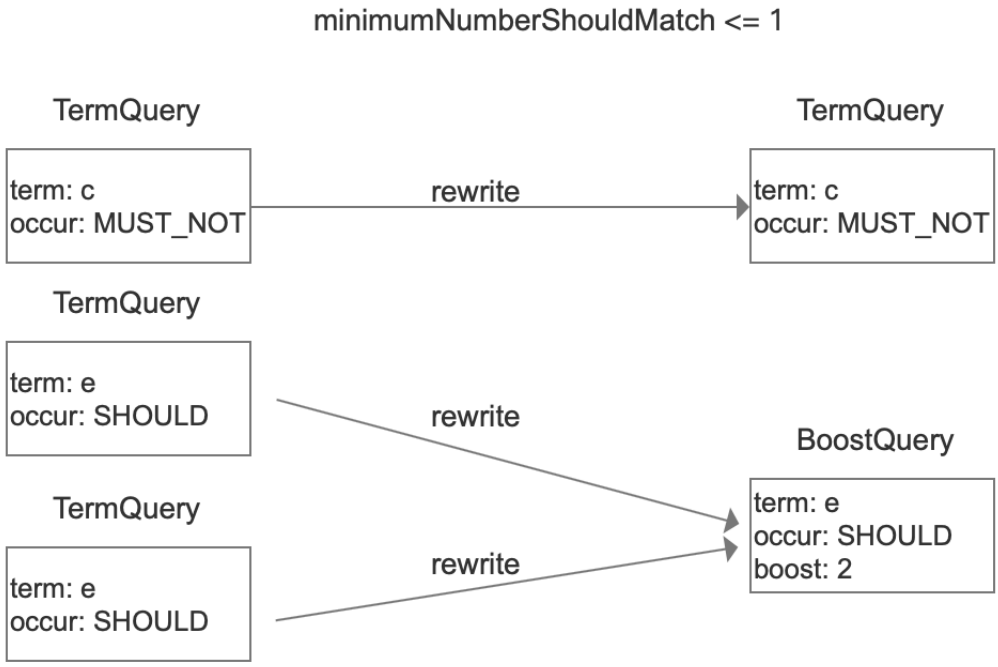
同样先要执行 逻辑二，不赘述。

重写第二步

执行逻辑四，不赘述。

重写第三步(可选)

执行逻辑七，不赘述。 下图中左边是重写前的BooleanQuery，右边是重写后的BooleanQuery。



SHOULD(至少一个)和MUST(至少一个)和MUST_NOT(至少一个)的TermQuery

重写第一步

同样先要执行 逻辑二，不赘述。

重写第二步

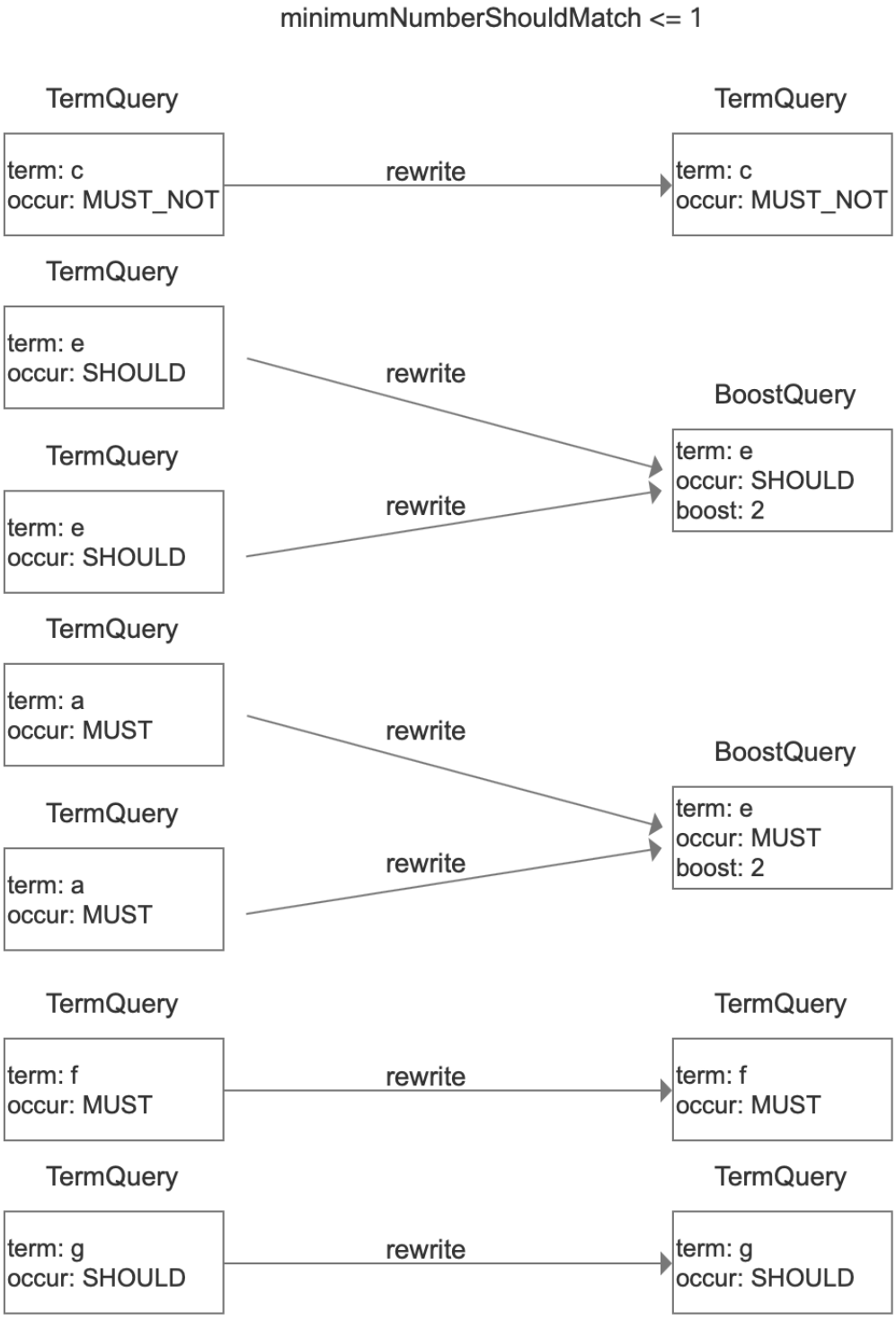
执行逻辑四，不赘述。

重写第三步(可选)

执行逻辑七，不赘述。

重写第四步

执行逻辑八，不赘述。 下图中左边是重写前的BooleanQuery，右边是重写后的BooleanQuery。



SHOULD(至少一个)和FILTER(至少一个)的TermQuery

重写第一步

同样先要执行 逻辑二，不赘述。

重写第二步

因为对于FILTER的Query中的term，他只是不参与打分，但是搜索结果必须包含这个term，如果SHOULD的Query中也有这个term，那么将这个Query的SHOULD改为MUST, 然后minShouldMatch的值就必须少一个，注意的是FILTER的这个Query没有放到新的BooleanQuery中。

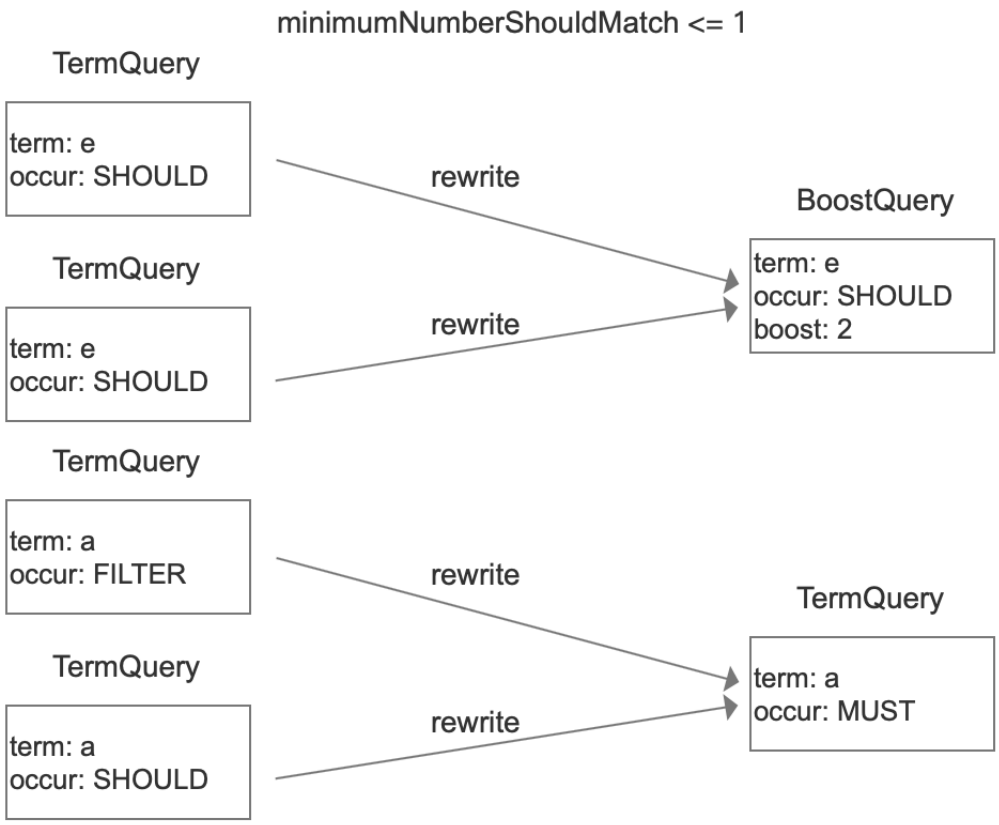
```
// 逻辑六
if (clauseSets.get(Occur.SHOULD).size() > 0 &&
clauseSets.get(Occur.FILTER).size() > 0) {
      final Collection<Query> filters = clauseSets.get(Occur.FILTER);
      final Collection<Query> shoulds = clauseSets.get(Occur.SHOULD);

      Set<Query> intersection = new HashSet<>(filters);
      // 在intersection中保留 FILTER跟SHOUL有相同的term的Query
      intersection.retainAll(shoulds);

      // if语句为真：说明至少有一个term，他即有FILTER又有SHOULD的Query
      if (intersection.isEmpty() == false) {
        // 需要重新生成一个BooleanQuery
        BooleanQuery.Builder builder = new BooleanQuery.Builder();
        int minShouldMatch = getMinimumNumberShouldMatch();

        for (BooleanClause clause : clauses) {
          if (intersection.contains(clause.getQuery())) {
            if (clause.getOccur() == Occur.SHOULD) {
            // 将SHOULD 改为 MUST
              builder.add(new BooleanClause(clause.getQuery(),
Occur.MUST));
              // 对minShouldMatch的值减一，因为这个SHOULD的Query的term，同样是
FILTER的term，满足匹配要求的文档必须包含这个term
              minShouldMatch--;
            }
          } else {
            builder.add(clause);
          }
        }
        // 更新minShouldMatch
        builder.setMinimumNumberShouldMatch(Math.max(0, minShouldMatch));
        return builder.build();
      }
    }
```

重写第三步(可选)

执行逻辑七，不赘述。

重写第四步

执行逻辑八，不赘述。 下图中左边是重写前的BooleanQuery，右边是重写后的BooleanQuery。



MUST(至少一个)和FILTER(至少一个)的TermQuery

重写第一步

同样先要执行 逻辑二，不赘述。

重写第二步

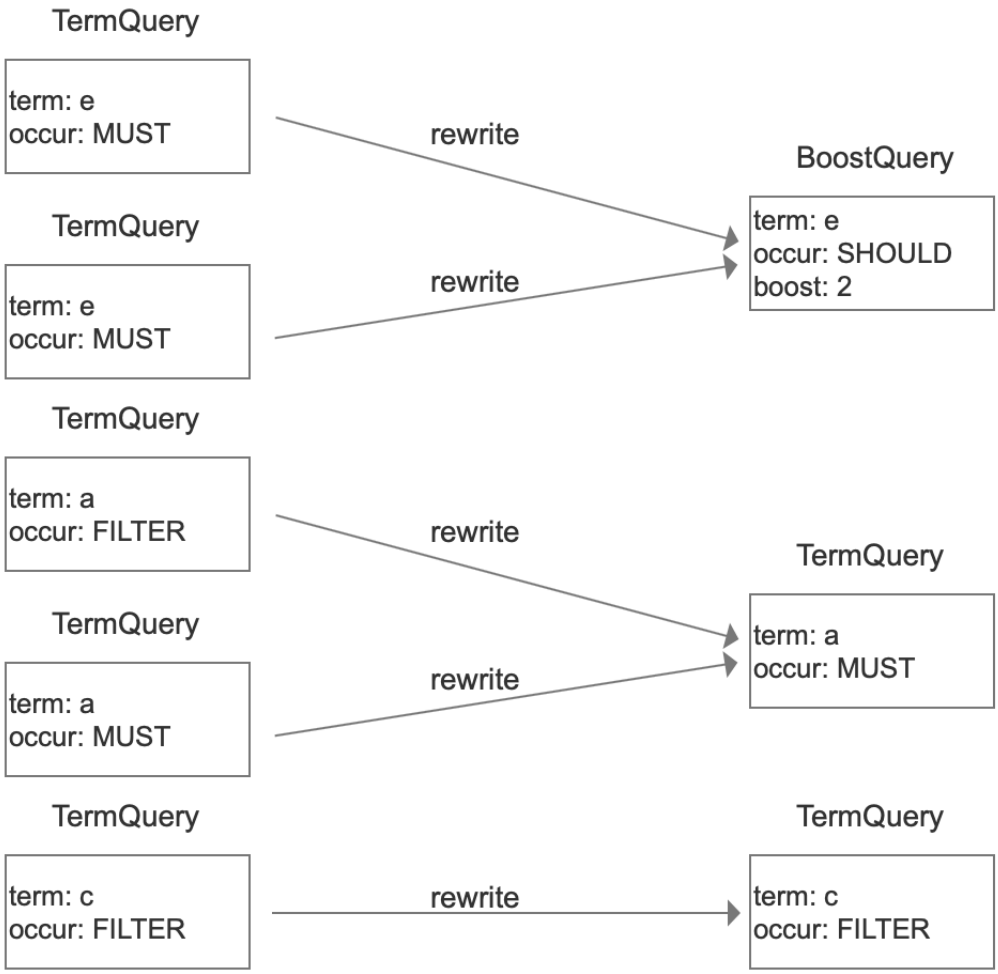
判断是否存在一个term对应的Query即是 MUST又是FILTER，如果存在，那么移除FILTER的Query。

```
// 逻辑六
if (clauseSets.get(Occur.MUST).size() > 0 &&
clauseSets.get(Occur.FILTER).size() > 0) {
      // 获得所有的FILTER的Query
      final Set<Query> filters = new HashSet<Query>
(clauseSets.get(Occur.FILTER));
      boolean modified = filters.remove(new MatchAllDocsQuery());
      // 从filters中移除既是FILTER又是MUST的Query
      modified |= filters.removeAll(clauseSets.get(Occur.MUST));
      if (modified) {
        BooleanQuery.Builder builder = new BooleanQuery.Builder();

 builder.setMinimumNumberShouldMatch(getMinimumNumberShouldMatch());
        for (BooleanClause clause : clauses) {
          if (clause.getOccur() != Occur.FILTER) {
            builder.add(clause);
```

```
14            }
15          }
16          for (Query filter : filters) {
17            builder.add(filter, Occur.FILTER);
18          }
19          return builder.build();
20        }
21      }
```

重写第三步

执行逻辑八，不赘述。 下图中左边是重写前的BooleanQuery，右边是重写后的BooleanQuery。



结语

BooleanQuery类中最重要的方法就是rewrite()方法，上面的例子中列举了最常用的几种BooleanQuery的情况，MUST，SHOULD，FILTER，MUST_NOT的他们之间不同数量的组合有着不一样的rewrite过程，无法一一详细列出。BooleanQuery中的rewrite一共有9个逻辑，都在关键处给出了注释，大家可以到我们的GitHub看这个类的源码https://github.com/luxugang/Lucene-7.5.0/blob/master/solr-7.5.0/lucene/core/src/java/org/apache/lucene/search/BooleanQuery.java。

点击下载Markdown文件