<https://www.cnblogs.com/cuihongyu3503319/p/9454149.html>

<https://www.cnblogs.com/netuml/p/5695751.html>

<https://www.cnblogs.com/cuihongyu3503319/category/1257523.html>

<https://www.cnblogs.com/HD/p/4023960.html>

[**Solr学习总结（六）solr的函数查询Function Queries**](https://www.cnblogs.com/cuihongyu3503319/p/9454149.html)

**摘要：** 函数查询允许你使用一个或多个数字字段的真实值生成一个相关性分数，函数查询在standard，DisMax，eDisMax下都能使用。   
查询函数可以是常量，字段或者其他函数的组合。使用函数可以影响结果的排序。本文整理自Solr官方文档：[https://lucene.apache.org/solr/guide/6\_6/function-queries.html](https://lucene.apache.org/solr/guide/6_6/function-queries.html" \t "_blank)

**使用Function Query**

1.直接向QParser指定函数参数，如func或frange：

q={!func}div{popularity,price}&fq={!frange l=1000}cus

tomer\_ratings

2.在排序时使用：

sort=div(popularity,price)desc,score desc

3.在查询结果中预处理一些输出字段

&fl=sum(x,y),id,a,b,c,score

4.指定函数的一个参数：在dDisMax中指定boost参数，在DisMax中指定bf参数

q=dismax&bf=”ord(popularity)^0.5 recip(rord(price),1,1000,1000)^0.3”

5.在Lucene的QParser中使用\_val\_关键字指定函数：

q=\_val\_:mynumericfield \_val\_:”recip(rord(myfield),1,2,3)”

**Solr中的可用函数**

| **函数** | **说明** | **举例** |
| --- | --- | --- |
| abs(x) | 返回绝对值 | abs(-5) |
| “constant” | 指定一个浮点数 | 1.5 |
| def(“field”,value) | 默认值，当指定字段不存在时，返回默认值 | def(rationg,5) |
| div(x,y) | 除法，x除以y | div(1,5) |
| dist | 计算两点之间的距离 | dis(2,x,y,0,0) |
| docfreq(field,val) | 返回某值在某字段出现的次数 | docfreq(title,’solr’) |
| field(“field”) | 返回该field的索引数量 | field(‘title’) |
| hsin | 曲面圆弧上两点之间的距离 | hsin(2,true,x,y,0,0) |
| idf | Inverse document frequency 倒排文档频率 | idf(“field”,’solr’) |
| if | if(test,value1,value2) | if(termfreq(title,’solr’),popularity,42) |
| linear(x,m,c) | 就是m\*x+c,等同于sum(product(m,x),c) | linear(1,2,4)=1x2+4=6 |
| log(x) | 以10为底，x的对数 | log(sum(x,100)) |
| map(x,min,max,target) | 如果x在min和max之间,x=target,否则x=x | map(x,0,0,1) |
| max(x,y,…) | 返回最大值 | max(2,3,0) |
| maxdoc | 返回索引的个数，查看有多少文档，包括被标记为删除状态的文档 | maxdoc() |
| min(x,y,…) | 返回最小值 | min(2,4,0) |
| ms | 返回两个参数间毫秒级的差别 | ms(datefield1,2000-01-01T00:00:00Z) |
| norm(field) | 返回该字段索引值的范数 | norm(title) |
| numdocs | 返回索引的个数，查看有多少文档，不包括被标记为删除状态的文档 | numdocs() |
| ord | 根据顺序索引发货结果 | ord(title) |
| pow(x,y) | 返回x的y次方 | pow(x,log(y)) |
| product(x,y) | 返回多个值得乘积 | product(x,2) |
| query | 返回给定的子查询的得分，或者文档不匹配的默认值值 | query(subquery,default) |
| recip(x,m,a,b) | 相当于a/(m\*x+b),a,m,b是常量，x是变量 | recip(myfield,m,a,b) |
| rord | 按ord的结果反序返回 |  |
| scale | 返回一个在最大值和最小值之间的值 | scale(x,1,3) |
| sqedist | 平方欧氏距离计算 | sqedist(x\_td,y\_td,0,0) |
| sqrt | 返回指定值得平方根 | sqrt(x)sqrt(100) |
| strdist | 计算两个字符串之间的距离 | strdist(“SOLR”,id,edit) |
| sub | 返回x-y | sub(field1,field2) |
| sum(x,y) | 返回指定值的和 | sum(x,y,…) |
| sumtotaltermfreq | 返回所有totaltermfreq的和 |  |
| termfreq | 词出现的次数 | termfreq(title,’sorl’) |
| tf | 词频 | tf(text,’solr’) |
| top | 功能类似于ord |  |
| totaltermfreq | 返回这个词在该字段出现的次数 | ttf(title,’memory’) |
| and | 返回true值当且仅当它的所有操作为true | and(not(exists(popularity)),exists(price)) |
| or | 返回true值当有一个操作为true | or(value1,value2) |
| xor | 返回false值如果所有操作都为真 | xor(field1,field2) |
| not | 排除操作 | not(exists(title)) |
| exists | 如果字段存在返回真 | exists(title) |
| gt,gte,lt,lte,eq | 比较函数 | 2 gt 1 |

[**Solr高亮与Field权重**](https://www.cnblogs.com/Leo_wl/p/3979017.html)

**阅读目录**

* [**Solr高亮与Field权重**](https://www.cnblogs.com/Leo_wl/p/3979017.html#_label0)
* [**Solr高亮**](https://www.cnblogs.com/Leo_wl/p/3979017.html#_label1)
* [**Solr权重**](https://www.cnblogs.com/Leo_wl/p/3979017.html#_label3)

[**回到目录**](https://www.cnblogs.com/Leo_wl/p/3979017.html#_labelTop)

# [Solr高亮与Field权重](http://www.cnblogs.com/edwinchen/p/3977366.html)

[**回到目录**](https://www.cnblogs.com/Leo_wl/p/3979017.html#_labelTop)

# Solr高亮

**原理**

做搜索时，高亮是很常见的需求，那么Solr肯定也为高亮提供了支持。先解释下Solr高亮的原理，在我们设置了需要高亮显示的Field之后，查询得到的返回结果会多出来下面的内容：

**[复制代码](javascript:void(0);)**

复制代码

"highlighting": {

"519": {

"Artist\_Name": [

"<em>周杰伦</em>"

]

}

}

复制代码

**[复制代码](javascript:void(0);)**

其实就是多了highlighting的字段，并没有改变原来返回的字段内容。

Json串是使用 Unique Field ：{高亮显示的内容}的形式。

SolrJ有三种高亮类型：

如果要对某field做高亮显示，必须对该field设置stored=true

Standard Highlighter，根据查询的docIdSet，获取Documents，并获取当前document的需要高亮的field的value，根据query的term和该field的value做匹配算法  
FastVector Highlighter，效率比普通的高亮显示要高；需要定义termvector(占用空间和IO)，包括position和offset，根据query term的termvector到field value中做快速的定位标记，进而实现快速的高亮显示  
Postings Highlighter，由于高亮显示需要对field设置为store=true，所有对于单节点数据量比较大并且该字段比较大的话，会消耗大量的IO操作，那么可以把该字段存储在另外的地方，比如Hbase，在外部做高亮显示的匹配。

其中推荐使用的是Standard Highlighter，下面也是针对Standard Highlighter，

**配置**

下面介绍两种配置方式：

　　1.SolrJ配置:

**[复制代码](javascript:void(0);)**

复制代码

//设置高亮

songQuery.setHighlight(true); // 开启高亮组件或用query.setParam("hl", "true"); songQuery.addHighlightField("Song\_Name,Song\_SingerName");// 高亮字段

songQuery.set("hl.highlightMultiTerm","true");//启用多字段高亮

songQuery.setHighlightSimplePre("<font style=\"color:#A7D043;font-weight:bold;\">"); //标记，高亮关键字前缀

songQuery.setHighlightSimplePost("</font>");//后缀

复制代码

**[复制代码](javascript:void(0);)**

　　2.solrConfig.xml配置：

**[复制代码](javascript:void(0);)**

复制代码

<requestHandler name="search" class="solr.SearchHandler" default="true">

<!-- default values for query parameters can be specified, these

will be overridden by parameters in the request

-->

<lst name="defaults">

<str name="echoParams">explicit</str>

<int name="rows">10</int>

<str name="df">text</str>

<str name="hl">true</str>

<str name="hl.fl">content</str>

<str name="f.name.hl.fragsize">50</str>

<str name="hl.simple.pre">&lt;font color=&quot;red&gt;</str>

<str name="hl.simple.post">&lt;/font&gt;</str>

</lst>

</requestHandler>

复制代码

**[复制代码](javascript:void(0);)**

详细的RequestHandler配置请参看博客：五、SolrJ、Request Handler

其实这两种配置并没有本质上的区别。我个人习惯使用SolrJ配置。

**解析**

获取highlighting是非常简单的，一条语句搞定：

Map<String,Map<String,List<String>>> tempMap = response.getHighlighting();

相信大家也注意到了，虽然接受结果简单，但是如果想遍历就比较复杂了，因为接受到的结果是嵌套了很多层的类型Map<String,Map<String,List<String>>>

那么我这边把我解析的方法分享下：

**[复制代码](javascript:void(0);)**

复制代码

Map<String,Map<String,List<String>>> tempMap = songHighlight.getHighlighting();

for(Map.Entry<String, Map<String,List<String>>> entry : tempMap.entrySet()) {

if(Integer.parseInt(entry.getKey()) == song.getSong\_SongID()) {

for(Map.Entry<String, List<String>> entryLayer2 : entry.getValue().entrySet()) {

if(null != entryLayer2.getKey() && "Song\_Name".equals(entryLayer2.getKey())) {

//your Operation

}

if(null != entryLayer2.getKey() && "Song\_SingerName".equals(entryLayer2.getKey())) {

//your Operation

}

}

}

}

复制代码

**[复制代码](javascript:void(0);)**

这个方法比Iterator和foreach效率稍高。

我设置了两个字段需要高亮，所以在循环中判断了高亮是属于哪个字段，之后进行相应的操作。

因为我做的时候，一首歌可能有几个Song\_SingerName，在数据库中用"/"分隔，所以这种情况更加复杂，我首先是把后缀中的/换成了出现概率很小的@

songQuery.setHighlightSimplePost("<@font>");//后缀

然后再用split（"@"）分隔出不同的Song\_SingerName，但是这样就会有一个问题，就是我不知道高亮的歌手到底是哪一个歌手，所以这个时候，我还需要从分割后的String[]中提取所有的中文字符，比对后，存入另一个变量，最后再用"/"替换掉"@"。

[**回到目录**](https://www.cnblogs.com/Leo_wl/p/3979017.html#_labelTop)

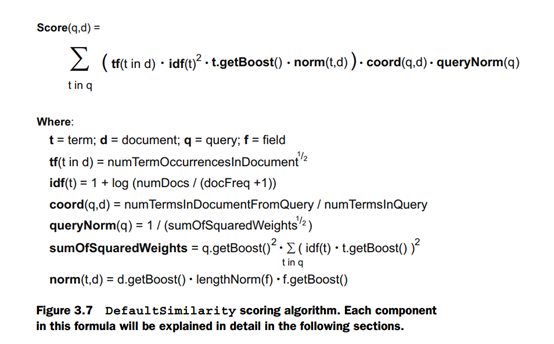
# 

[**回到目录**](https://www.cnblogs.com/Leo_wl/p/3979017.html#_labelTop)

# Solr权重

**概念**

Solr底层依然用的是Lucene的权重算法，也就是通过一个公式计算每个Documents的得分，然后按得分高低排序，公式如下：



简单解释下这个公式中包含的一些因子：

Tf：Term frequency，就是条目出现的次数。

Idf: Inverse document frequency，就是用来描述在一个搜索关键字中，不同字词的稀有程度。比如搜索The Cat in the Hat，那么很明显The和in远没有Cat和Hat重要。

Boosting：这个使我们设置权重的重点，比如搜索歌手名，那么在一个document中还有歌手的ID、歌曲的清晰度、歌曲上传时间，而boosting是不同的Filed有不同权重，之后根据公式计算得分。所以可以看到，我们并不能直接影响solr搜索结果的排序，需要改变权重，进而改变不同Document的得分，从而影响排序。

其中还有很多因子和公式的解释，有兴趣的同学可以参考Solr in action这本书，里面有比较详细的解释。

因为我们只需简单的根据某一Filed的权重影响结果的排序，所以我们需要改变Document的Boosting，那么就需要用到Dismax，Dismax是一个查询解析器（Query parser），查询解析器的概念就是提供了一系列查询的参数，一旦我们在查询url中设置了相应的参数，那么查询解析器将会解析查询信息，从而得到搜索结果，其实完全也可以把查询解析器理解为一个Api，就是提供了相应的方法，我们设置，之后Solr根据我们设置的参数进行查询，只不过不同的Query Parser提供了不同的参数而已。

一共提供了三种Query Parser

Standard：最常用的，并且是默认

Dismax：

Extended Dismax

功能从上至下是逐渐递增的，在大部分情况下，Standard已经可以完全满足需求，但是因为要使用权重排序，那么需要用到Dismax，具体提供的参数请查看wiki：

[**https://cwiki.apache.org/confluence/display/solr/The+Standard+Query+Parser**](https://cwiki.apache.org/confluence/display/solr/The+Standard+Query+Parser)

那么首先需要设置Query Parser为Dismax：

songQuery.set("defType","dismax");

之后设置需要查询的Field：

songQuery.set("qf","Song\_Name^2 Song\_SingerName^0.2");

比如我这里就需要根据用户输入的关键字查询歌手名和歌曲名，之后返回这两个Field命中的结果、多个Query Filed中可以设置不同的权重，比如Song\_Name的权重就为2，必须注意，在Solr权重的设置中，所有权重标准为1，意思是当权重设置大于1时，代表这个字段的权重变大，如果权重设置小于1并且大于0的时候，代表这个字段权重变小。

之后设置其它Field的权重：

songQuery.set("bf", "sum(div(Song\_Quality,0.01),if(exists(Song\_FileMV),20000,0),recip(ms(NOW,Song\_CreateTimeForNew),1,10000,1))");

这里面用到了很多Function Query，比如div,代表相除、exists代表如果Song\_FileMV如果不为空那么设置它的权重为20000，为空则为0。记住最后要sum起来，因为从上面的公式可以看出来，boosting是一个变量，所以最好要有一个和值。相关的函数请参考wiki：

[**http://wiki.apache.org/solr/FunctionQuery**](http://wiki.apache.org/solr/FunctionQuery)

这样搜索结果就会按照有MV的优先显示、最近上传的优先显示、清晰度高的优先显示。

### 1、字段

**Filed**:<field name="\_id" type="text\_ik" indexed="true" stored="true" required="true" multiValued="false" />

　　　　name:字段名

　　　　type：字段类型，影响分词结果

　　　　indexed：是否需要索引

　　　　stored：是否需要存储

　　　　required：是否唯一

　　　　multiValued：是否多值，对于copyField字段有效

**dynamicField**：动态字段 <dynamicField name="bd01010\*" type="text\_ik"    indexed="true"  stored="true"  multiValued="true"/>

　　　　如：{"\_id":"change.me","bd01003":"美好的生活，未来","bd01010.0.text":"你好，计算机能力","bd01010.1.text":"生活","bd01007":"清华大学出版社"}

　　　　建立索引之后，如下

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 {

2 "responseHeader": {

3 "status": 0,

4 "QTime": 3,

5 "params": {

6 "lowercaseOperators": "true",

7 "bq": "bd01test^2",

8 "fl": "score,\_id,bd01003,bd01010\*,bd01007,bd01keyword",

9 "indent": "true",

10 "qf": "bd01keyword",

11 "\_": "1469434764614",

12 "q.alt": "bd01keyword:计算机",

13 "stopwords": "true",

14 "wt": "json",

15 "defType": "edismax"

16 }

17 },

18 "response": {

19 "numFound": 2,

20 "start": 0,

21 "maxScore": 0.09362577,

22 "docs": [

23 {

24 "\_id": "change.me2",

25 "bd01003": "计算机",

26 "bd01010.0.text": [

27 "你好，能力"

28 ],

29 "bd01010.1.text": [

30 "生活"

31 ],

32 "bd01007": "清华大学出版社",

33 "score": 0.09362577

34 },

35 {

36 "\_id": "change.me",

37 "bd01003": "美好的生活，未来",

38 "bd01010.0.text": [

39 "你好，计算机能力"

40 ],

41 "bd01010.1.text": [

42 "生活"

43 ],

44 "bd01007": "清华大学出版社",

45 "score": 0.06553804

46 }

47 ]

48 }

49 }

[复制代码](javascript:void(0);)

**copyField**:字段组合 ，如下

1 <copyField source="bd01010\*" dest="bd01test"/>

2 <copyField source="bd01test" dest="bd01keyword"/>

3 <copyField source="bd01003" dest="bd01keyword"/>

4 <copyField source="bd01007" dest="bd01keyword"/

　　如果想对上面的**dynamicField**字段进行查询，====》bd01test:计算机

　　如果想对上面的**dynamicField**字段+其它字段进行查询，====》bd01keyword:计算机

### 2、edismax

**q.alt**: 当q字段为空时，用于设置缺省的query，通常设置q.alt为\*:\*。 如q.alt = title:计算机

**qf**:query fields，指定solr从哪些field中搜索。 如qf=title

**mm**: Minimum ‘Should’ Match。 Solr支持三种查询clause，即“必须出现”， “不能出现”和“可以出现”，分别对应于AND, -, OR

**pf**:**boosting phrases over words。用于指定一组field，当query完全匹配pf指定的某一个field时，来进行boost，给搜索匹配到的字段打分  如pf =字段1^0.5 字段2^0.2**

**ps**: Phrase Slop. 短语坡度。短语查询的坡度量用在pf字段，影响boost。

**qs**:Query Phrase Slop。查询短语坡度。查询短语坡度是指短语查询明确包含用户查询的字符串(在qf字段，影响匹配)。

**tie**：tie breaker。float值作为决胜局中DisjunctionMaxQueries使用（应该是远小于1）。

**bq**: Boost Query。对**某个field**的value进行boost，例如brand:xq^5.0。

**bf :Boost Functions。用函数的方式计算boost。**

**uf:**User Fields。用户字段。制定模式的字段可以被用户显示的查询。此参数支持通配符。

**pf2**：Phrase bigram fields。短语两字母字段。e.g. “the brown fox jumped” is queried as “the brown” “brown fox” “fox jumped”。

**pf3**：Phrase trigram fields。短语三字母字段。e.g. “the brown fox jumped” is queried as “the brown fox” “brown fox jumped”。

**ps2**：短语两字母坡度。如果未指定，将使用”ps”。

**ps3**：短语三字母坡度。如果未指定，将使用”ps”。

**boost**：**Boost Function, multiplicative。作为bf，score=bf\*score。bf =sum(div(字段,100),1),**

**stopwords**：单词停用，true 或false。

**lowercaseOperators**:此参数用于控制小写单词作为布尔运算符，如”and” and “or”。设置与lowercaseOperators= true来允许此。默认为true。

### 3、solr 函数

**bf、boost 里都可以写权重的函数**

**div**(字段,100)：表示 字段的值/100 得到一个新值

**pow**(字段,2)：表示 字段的值的2次方   得到一个新值

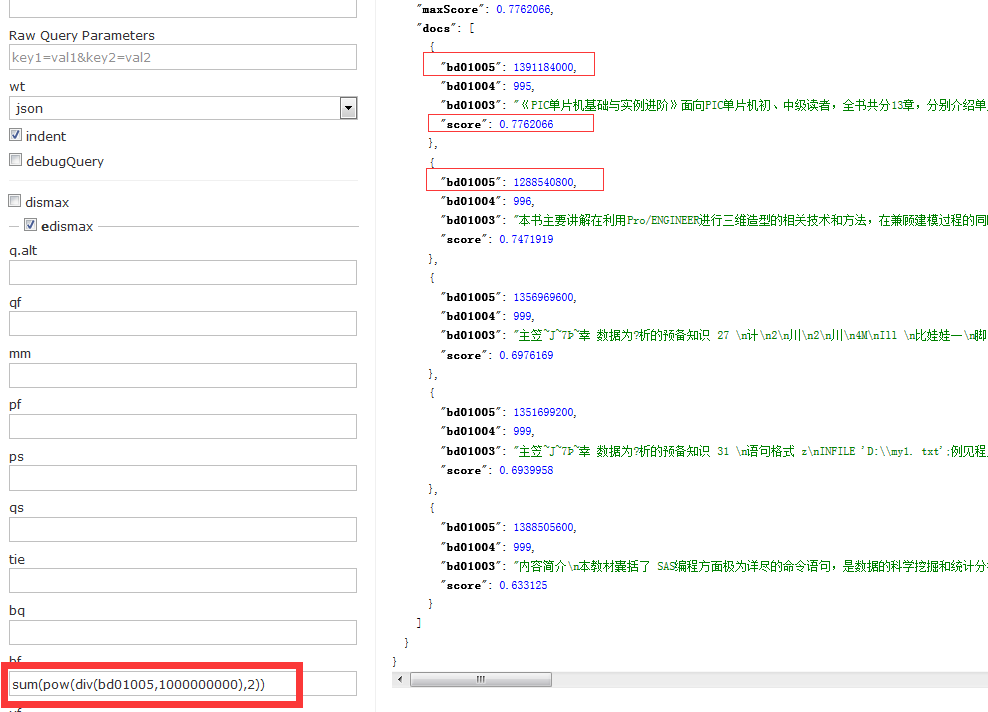
**linear(x,2,4)** 表示  2\*x+4

**sqrt(x)** 表示 X的平方根

　  boost = sum(div(字段,100),1), 设 a=字段值  ，则 权重 =权重\*（a/100+1）；



图：未使用权重的结果



　　　　　　　　　　　　图：使用了权重之后的结果

1、constant：支持有小数点的常量，

例如，1.5，查询表达式就是：\_val\_:1.5

2、fieldvalue，这个函数将会返回numeric field的值，这个域必须是indexd的，非multiValued的。格式很简单，就是该域的名字。如果这个域中没有这样的值，那么将会返回0。

3、ord，对于一个域，它所有的值都将会按照字典顺序排列，这个函数返回你要查询的那个特定的值在这个顺序中的排名。这个域，必须是非multiValued的，当没有值存在的时候，将返回0。

例如：某个特定的域只能去三个值，“apple”、“banana”、“pear”，那么ord（“apple”）=1，ord（“banana”）=2，ord（“pear”）=3.

需要注意的是，ord（）这个函数，依赖于值在索引中的位置，所以当有文档被删除、或者添加的时候，ord（）的值就会发生变化。当你使用MultiSearcher的时候，这个值也就是不定的了。

4、rord，这个函数将会返回与ord相对应的倒排序的排名。

格式: rord(myIndexedField).

5、sum，这个函数的意思就显而易见啦，它就是表示“和”。

格式：sum(x,1)

        sum(x,y)

        sum(sqrt(x),log(y),z,0.5)

6、product，product(x,y,...)将会返回多个函数的乘积。

格式：product(x,2)

        product(x,y)

7、div，div(x,y)表示x除以y的值

格式：div（1,x）

        div(sum(x,100),max(y,1))

8、pow，pow表示幂值。pow(x,y) =x^y。

格式：pow(x,0.5) 表示开方

            pow(x,log(y))

9、abs，abs(x)将返回表达式的绝对值

格式：abs(-5)

        abs(x)

10、log，log(x)将会返回基数为10，x的对数

格式： log(x)

         log(sum(x,100))

11、sqrt，sqrt(x) 返回 一个数的平方根

格式：sqrt（2）

          sqrt(sum(x,100))

12、map，如果 min<=x<=max，那么map(x,min,max,target)=target，如果 x不在[min,max]这个区间内，那么map(x,min,max,target)=x.

格式：map(x,0,0,1)

13、scale，scale(x,minTarget,maxTarget) 这个函数将会把x的值限制在[minTarget,maxTarget]范围内。

14、query，query(subquery,default)将会返回给定subquery的分数，如果subquery与文档不匹配，那么将会返回默认值。任何的查询类型都是受支持的。可以通过引用的方式，也可以直接指定查询串。

 例子：q=product(popularity, query({!dismax v='solr rocks'})) 将会返回popularity和通过dismax 查询得到的分数的乘积。

         q=product(popularity, query($qq)&qq={!dismax}solr rocks) 跟上一个例子的效果是一样的。不过这里使用的是引用的方式

         q=product(popularity, query($qq,0.1)&qq={!dismax}solr rocks) 在前一个例子的基础上又加了一个默认值。

15、linear，linear(x,m,c)表示 m\*x+c ,其中m和c都是常量，x是一个变量也可以是一个函数。

例如：linear(x,2,4)=2\*x+4.

16、recip，recip(x,m,a,b)=a/(m\*x+b)，其中，m、a、b是常量，x是变量或者一个函数。当a=b，并且x>=0的时候，这个函数的最大值是1，值的大小随着x的增大而减小。

例如：recip(rord(creationDate),1,1000,1000)

17、max，max(x,c)将会返回一个函数和一个常量之间的最大值。

例如：max(myfield,0)

### 4、autoCommint 和autoSoftCommint

 　　在很多情况下，我们并不想自己主动触发COMMIT相应的XML给SOLR，这样带来很多的不便（例如频繁commit，会带来性能上较大的消耗），恰 好，SOLR通过配置文件可以自行在满足指定的条件下自动的COMMIT索引，同时，让前端检索实例可以检索到最新生成的数据，而不需要人为干预。

[复制代码](javascript:void(0);)

1 <requestHandler name="/update" class="solr.XmlUpdateRequestHandler">

2 <maxPendingDeletes>10000</maxPendingDeletes>

3 <autoCommit>

4 <maxDocs>20</maxDocs>

5 <maxTime>${solr.autoCommit.maxTime:15000}</maxTime>   
6　　　　　　 　<openSearcher>true</openSearcher>

7 </autoCommit>

8 </requestHandler>

[复制代码](javascript:void(0);)

　　上 面autoCommit中的maxDocs指定的pending多少个文档后执行一次commit，而maxTime指定了多长时间间隔进行一次 commit，一般这两个选项只需要配置一个即可满足需要。另外，每次commit会将最近的更新生效，但是如果一次commit操作尚未完成，又达到了 下一次commit的时刻，这样做会严重影响索引的吞吐量。

　　自动硬提交方式:maxTime：设置多长时间提交一次maxDocs：设置达到多少文档提交一次openSearcher：文档提交后是否开启新的searcher，  
如果false，文档只是提交到index索引库，搜索结果中搜不到此次提交的文档；如果true，既提交到index索引库，也能在搜索结果中搜到此次提交的内容。

在Solr 4.0将会实现一种基于“软自动提交”（soft auto commit）的功能，它会根据当前的系统上下文决定是否提交（简单的情况就是，确保每次commit完成，也就是最近的索引数据更新已经更新同步到磁盘上之后再自动执行下一次commit）

1 <autoSoftCommit>

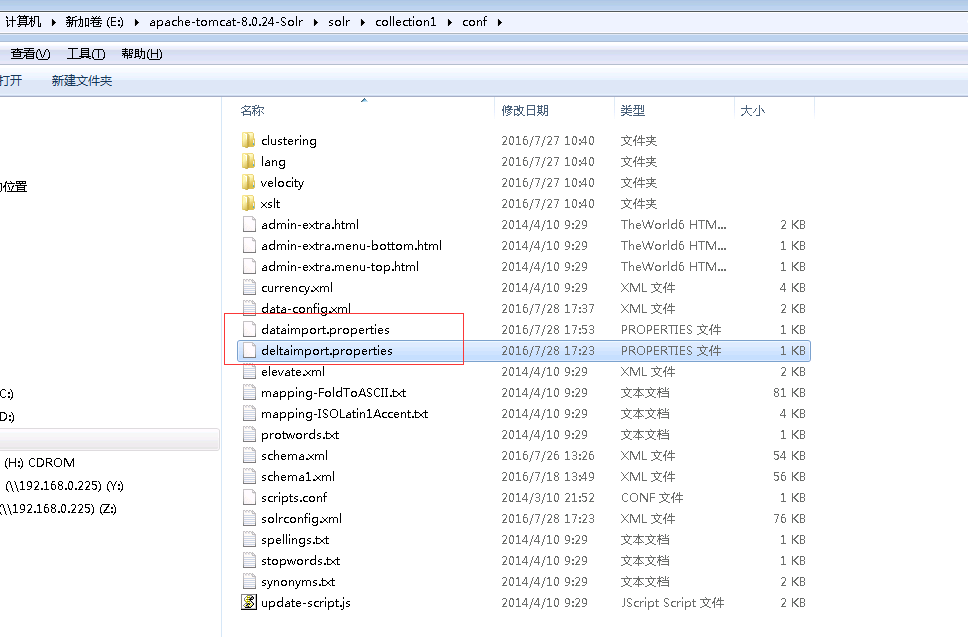
2 <maxTime>${solr.autoSoftCommit.maxTime:1800000}</maxTime>

3 <maxDocs>100</maxDocs>

4 </autoSoftCommit>

### 5、Solr与mysql 增量索引同步

　1.在进行增量索引前，首先要弄懂几个必要的属性，以及数据库建表事项，和 deltaimport.properties,dataimport.properties, data-config.xml里面的数据   
  　　 transformer 格式转化：HTMLStripTransformer 索引中忽略HTML标签  
   　　query:查询数据库表符合记录数据    
         deltaQuery:增量索引查询主键ID     注意这个只能返回ID字段   
   　　deltaImportQuery:增量索引查询导入数据    
   　　deletedPkQuery:增量索引删除主键ID查询  ---> 注意这个只能返回ID字段   
         数据库配置注意事项   
　　　　1.如果只涉及添加，与修改业务，那么数据库里只需额外有一个timpstamp字段   
　　　　　　就可以了，默认值为当前系统时间，CURRENT\_TIMESTAMP（笔者的数据为mysql的）   
　　　　2.如果还涉及删除业务，那么数据里就需额外再多添加一个字段isdelete，int类型的   
　　　　　　用0,1来标识，此条记录是否被删除，当然也可以用其他字段标识，ture或false都可以

　　　deltaimport.properties,dataimport.properties  
　　　　　这个配置文件很重要，它是用来记录当前时间与上一次修改时间的，通过它能够找出，那些，新添加的，修改的，或删除的记录

　　clean:选择是否要在索引开始构建之前删除之前的索引，默认为true

　　commit:选择是否在索引完成之后提交。默认为true

　　optimize:是否在索引完成之后对索引进行优化。默认为true

　　debug:是否以调试模式运行，适用于交互式开发（interactive development mode）之中。

　新增collections1/conf/data-config.xml文件

[复制代码](javascript:void(0);)

1 <dataConfig>

2 <dataSource type="JdbcDataSource" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://192.168.0.121:3306/test" user="root" password="ttestO" batchSize="-1" />

3 <document name="content">

4 <entity pk="bd01001" name="bd01"

5 query="select bd01001,bd01002,bd01003,bd01004,bd01005,bd01006 from `bd01`"

6 deltaQuery="select bd01001 from bd01 where bd01996 > '${dataimporter.last\_index\_time}'"

7 deletedPkQuery="select bd01001 from bd01 where bd01997=1"

8 deltaImportQuery="select bd01001,bd01002,bd01003,bd01004,bd01005,bd01006 from bd01 where bd01001='${dataimporter.delta.bd01001}'">

9 <field column="bd01001" name="\_id"/>

10 <field column="bd01002" name="bd01002" />

11 <field column="bd01003" name="bd01003" />

12 <field column="bd01004" name="bd01004" />

13 <field column="bd01005" name="bd01005"/>

14 <field column="bd01006" name="bd01006" />

15 </entity>

16 </document>

17 </dataConfig>

[复制代码](javascript:void(0);)

　　下面分别来介绍属性:

　　 entity.pk="bd01001"  数据表的主键ID

　　 全量导入：

　　 　　query="select \* from bd01 where isdelete=0  查询是指 查询出表里所有未删除的数据进行全量导入建索引

　　 增量索引

　　　　 添加、修改 索引

　　　　 deltaQuery="select bd01001 from bd01 where isDelete = 0 and  my\_date > '${dataimporter.last\_index\_time}'"

　　　　 deltaQuery的意思是，查询出所有经过修改的记录的ID 可能是修改操作，添加操作，删除操作产生的 (此查询只对增量导入起作用，而且只能返回ID值)

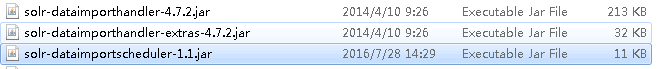
　　　　 deltaImportQuery="select \* from bd01 where ID='${dataimporter.delta.bd01001}'" 次查询是获取以上两步的ID，然后把其全部数据获取，根据获取的数据 对索引库进行更新操作，可能是添加，修改 (此查询只对增量导入起作用)

　　　　 删除索引

　　       deletedPkQuery="select ID from bd01 where isdelete=1" 此操作值查询那些数据库里伪删除的数据的ID（即isdelete标识为1的数据） solr通过它来删除索引里面对应的数据 (此查询只对增量导入起作用，而且只能返回ID值)

### 6、增量索引定时同步更新

  　　　　1. 将 apache-solr-dataimportscheduler-1.0.jar 和solr自带的 apache-solr-dataimporthandler-.jar, apache-solr-dataimporthandler-extras-.jar 放到solr.war的lib目录下面



   　　　  2.修改solr.war中WEB-INF/web.xml, 在servlet节点前面增加:

1 <listener>

2 <listener-class>

3 　　　　　　　　org.apache.solr.handler.dataimport.scheduler.ApplicationListener

4 </listener-class>

5 </listener>

 　　　　3.将apache-solr-dataimportscheduler-.jar 中 dataimport.properties 取出并根据实际情况修改,然后放到 solr.home/conf (不是solr.home/core/conf) 目录下面

[复制代码](javascript:void(0);)

1 #################################################

2 # #

3 # dataimport scheduler properties #

4 # #

5 #################################################

6

7 # to sync or not to sync

8 # 1 - active; anything else - inactive

9 syncEnabled=1

10

11 # which cores to schedule

12 # in a multi-core environment you can decide which cores you want syncronized

13 # leave empty or comment it out if using single-core deployment

14 syncCores=collection1

15

16 # solr server name or IP address

17 # [defaults to localhost if empty]

18 server=localhost

19

20 # solr server port

21 # [defaults to 80 if empty]

22 port=8080

23

24 # application name/context

25 # [defaults to current ServletContextListener's context (app) name]

26 webapp=solr

27

28 # URL params [mandatory]

29 # remainder of URL

30 #增量

31 params=/dataimport?command=delta-import&clean=false&commit=true&wt=json&indent=true&optimize=false

32

33 # schedule interval

34 # number of minutes between two runs

35 # [defaults to 30 if empty]

36 interval=1

37

38 # 重做索引的时间间隔，单位分钟，默认7200，即1天;

39 # 为空,为0,或者注释掉:表示永不重做索引

40 reBuildIndexInterval=0

41

42 # 重做索引的参数

43 #reBuildIndexParams=/dataimportUserFull?command=full-import&clean=true&commit=true&optimize=true&wt=json&indent=true&entity=zpxx&verbose=false&debug=false

44 reBuildIndexParams=/dataimport?command=full-import&indent=true&wt=json&\_=1469693722233423

45 # 重做索引时间间隔的计时开始时间，第一次真正执行的时间=reBuildIndexBeginTime+reBuildIndexInterval\*60\*1000；

46 # 两种格式：2012-04-11 03:10:00 或者 03:10:00，后一种会自动补全日期部分为服务启动时的日期

47 reBuildIndexBeginTime=16:06:30

[复制代码](javascript:void(0);)