关于solr schema.xml 和solrconfig.xml的解释 - CONAN ZONE - BlogJava

笔记本： 技术资料
创建时间： 2015/2/2 14:15
URL： http://www.blogjava.net/conans/articles/379545.html

关于solr schema.xml 和solrconfig.xml的解释

一、字段配置（schema）

schema.xml位于solr/conf/目录下，类似于数据表配置文件，

定义了加入索引的数据的数据类型，主要包括type、fields和其他的一些缺省设置。

1、先来看下type节点，这里面定义FieldType子节点，包括name,class,positionIncrementGap等一些参数。

- name：就是这个FieldType的名称。
- class：指向org.apache.solr.analysis包里面对应的class名称，用来定义这个类型的行为。

```
1. < schema   name = "example"   version = "1.2" >
2.   < types >
3.     < fieldType   name = "string"   class = "solr.StrField"   sortMissingLast = "true"
   omitNorms = "true" />
4.     < fieldType   name = "boolean"   class = "solr.BoolField"   sortMissingLast = "true"
   omitNorms = "true" />
5.     < fieldtype   name = "binary"   class = "solr.BinaryField" />
6.     < fieldType   name = "int"   class = "solr.TrieIntField"   precisionStep = "0"
   omitNorms = "true"
7.                                                 positionIncrementGap = "0" />
8.     < fieldType   name = "float"   class = "solr.TrieFloatField"   precisionStep = "0"
   omitNorms = "true"
9.                                                 positionIncrementGap = "0" />
10.    < fieldType   name = "long"   class = "solr.TrieLongField"   precisionStep = "0"
   omitNorms = "true"
11.                                                positionIncrementGap = "0" />
12.    < fieldType   name = "double"   class = "solr.TrieDoubleField"   precisionStep = "0"
   omitNorms = "true"
13.                                                positionIncrementGap = "0" />
14.   ...
15.   </ types >
16.   ...
17. </ schema >
```

必要的时候fieldType还需要自己定义这个类型的数据在建立索引和进行查询的时候要使用的分析器analyzer，包括分词和过滤，如下：

view plain print ?

```
1. < fieldType   name = "text_ws"   class = "solr.TextField"   positionIncrementGap =
   "100" >
2.   < analyzer >
3.     < tokenizer   class = "solr.WhitespaceTokenizerFactory" />
4.   </ analyzer >
5. </ fieldType >
6. < fieldType   name = "text"   class = "solr.TextField"   positionIncrementGap = "100"
   >
7.   < analyzer   type = "index" >
8.     <!--这个分词包是空格分词，在向索引库添加text类型的索引时，Solr会首先用空格
   进行分词
9.         然后把分词结果依次使用指定的过滤器进行过滤，最后剩下的结果，才会加入到
   索引库中以备查询。
10.      注意:Solr的analysis包并没有带支持中文的包，需要自己添加中文分词器，google
   下。
11.     -->
12.    < tokenizer   class = "solr.WhitespaceTokenizerFactory" />
13.       <!-- in this example, we will only use synonyms at query time
14.       < filter   class = "solr.SynonymFilterFactory"   synonyms = "index_synonyms.txt"
15.                                       ignoreCase = "true"   expand = "false" />
16.       -->
17.       <!-- Case insensitive stop word removal.
18.         add enablePositionIncrements = true  in both the index and query
19.         analyzers to leave a 'gap' for more accurate phrase queries.
20.       -->
21.      < filter   class = "solr.StopFilterFactory"
22.               ignoreCase = "true"
23.               words = "stopwords.txt"
24.               enablePositionIncrements = "true"
25.               />
26.      < filter   class = "solr.WordDelimiterFilterFactory"   generateWordParts = "1"
27.            generateNumberParts = "1"   catenateWords = "1"   catenateNumbers = "1"
28.            catenateAll = "0"   splitOnCaseChange = "1" />
29.      < filter   class = "solr.LowerCaseFilterFactory" />
30.      < filter   class = "solr.SnowballPorterFilterFactory"   language = "English"
31.                                          protected = "protwords.txt" />
32.    </ analyzer >
33.    < analyzer   type = "query" >
34.      < tokenizer   class = "solr.WhitespaceTokenizerFactory" />
35.        < filter   class = "solr.SynonymFilterFactory"   synonyms = "synonyms.txt"
   ignoreCase = "true"
36.                                                   expand = "true" />
```

```
37.         < filter   class = "solr.StopFilterFactory"
38.                 ignoreCase = "true"
39.                 words = "stopwords.txt"
40.                 enablePositionIncrements = "true"
41.                 />
42.         < filter   class = "solr.WordDelimiterFilterFactory"   generateWordParts = "1"
43.                 generateNumberParts = "1"   catenateWords = "0"   catenateNumbers =
    "0"
44.                                     catenateAll = "0"   splitOnCaseChange = "1" />
45.         < filter   class = "solr.LowerCaseFilterFactory" />
46.         < filter   class = "solr.SnowballPorterFilterFactory"   language = "English"
47.                                                 protected = "protwords.txt" />
48.       </ analyzer >
49.  </ fieldType >
```

2、再来看下fields节点内定义具体的字段（类似数据库的字段），含有以下属性：

- name：字段名
- type：之前定义过的各种FieldType
- indexed：是否被索引
- stored：是否被存储（如果不需要存储相应字段值，尽量设为false）
- multiValued：是否有多个值（对可能存在多值的字段尽量设置为true，避免建索引时抛出错误）

view plain print ?

```
1.  < fields >
2.     < field   name = "id"   type = "integer"   indexed = "true"   stored = "true"
    required = "true"   />
3.     < field   name = "name"   type = "text"   indexed = "true"   stored = "true"   />
4.     < field   name = "summary"   type = "text"   indexed = "true"   stored = "true"   />
5.     < field   name = "author"   type = "string"   indexed = "true"   stored = "true"   />
6.     < field   name = "date"   type = "date"   indexed = "false"   stored = "true"   />
7.     < field   name = "content"   type = "text"   indexed = "true"   stored = "false"   />
8.     < field   name = "keywords"   type = "keyword_text"   indexed = "true"   stored =
    "false"   multiValued = "true"   />
9.     <!--拷贝字段-->
10.    < field   name = "all"   type = "text"   indexed = "true"   stored = "false"
    multiValued = "true" />
11. </ fields >
```

3、建议建立一个拷贝字段，将所有的 全文本 字段复制到一个字段中，以便进行统一的检索：

以下是拷贝设置：

view plain print ?

```
1. < copyField   source = "name"   dest = "all" />
2. < copyField   source = "summary"   dest = "all" />
```

4、动态字段，没有具体名称的字段，用dynamicField字段

如：name为*_i，定义它的type为int，那么在使用这个字段的时候，任务以_i结果的字段都被认为符合这个定义。如name_i, school_i

view plain print ?

```
1. < dynamicField   name = "*_i"    type = "int"      indexed = "true"    stored = "true"
   />
2. < dynamicField   name = "*_s"    type = "string"    indexed = "true"    stored = "true"
   />
3. < dynamicField   name = "*_l"    type = "long"     indexed = "true"    stored = "true"
   />
4. < dynamicField   name = "*_t"    type = "text"      indexed = "true"    stored = "true"
   />
5. < dynamicField   name = "*_b"    type = "boolean"   indexed = "true"    stored =
   "true" />
6. < dynamicField   name = "*_f"    type = "float"    indexed = "true"    stored = "true"
   />
7. < dynamicField   name = "*_d"    type = "double"   indexed = "true"    stored = "true"
   />
8. < dynamicField   name = "*_dt"   type = "date"      indexed = "true"    stored = "true"
   />
```

schema.xml文档注释中的信息：

1、为了改进性能，可以采取以下几种措施：

- 将所有只用于搜索的，而不需要作为结果的field（特别是一些比较大的field）的stored设置为false
- 将不需要被用于搜索的，而只是作为结果返回的field的indexed设置为false
- 删除所有不必要的copyField声明
- 为了索引字段的最小化和搜索的效率，将所有的 text fields的index都设置成field，然后使用copyField将他们都复制到一个总的 text field上，然后对他进行搜索。
- 为了最大化搜索效率，使用java编写的客户端与solr交互（使用流通信）

- 在服务器端运行JVM（省去网络通信），使用尽可能高的Log输出等级，减少日志量。

2、< schema name =" **example** " version =" **1.2** " >

- name：标识这个schema的名字
- version：现在版本是1.2

3、filedType

< fieldType name =" **string** " class =" **solr.StrField** " sortMissingLast =" **true** " omitNorms =" **true** " />

- name：标识而已。
- class和其他属性决定了这个fieldType的实际行为。（class以solr开始的，都是在org.appache.solr.analysis包下）

可选的属性：

- sortMissingLast和sortMissingFirst两个属性是用在可以内在使用String排序的类型上（包括：string,boolean,sint,slong,sfloat,sdouble,pdate）。
- sortMissingLast="true"，没有该field的数据排在有该field的数据之后，而不管请求时的排序规则。
- sortMissingFirst="true"，跟上面倒过来呗。
- 2个值默认是设置成false

StrField类型不被分析，而是被逐字地索引/存储。

StrField和TextField都有一个可选的属性“compressThreshold”，保证压缩到不小于一个大小（单位：char）

< fieldType name =" **text** " class =" **solr.TextField** " positionIncrementGap =" **100** " >

solr.TextField 允许用户通过分析器来定制索引和查询，分析器包括一个分词器（tokenizer）和多个过滤器（filter）

- positionIncrementGap：可选属性，定义在同一个文档中此类型数据的空白间隔，避免短语匹配错误。

name: 字段类型名
class: java类名
indexed: 缺省true。 说明这个数据应被搜索和排序，如果数据没有indexed，则stored应是true。
stored: 缺省true。说明这个字段被包含在搜索结果中是合适的。如果数据没有stored,则

indexed应是true。
sortMissingLast: 指没有该指定字段数据的document排在有该指定字段数据的document的后面
sortMissingFirst: 指没有该指定字段数据的document排在有该指定字段数据的document的前面
omitNorms: 字段的长度不影响得分和在索引时不做boost时，设置它为true。一般文本字段不设置为true。
termVectors: 如果字段被用来做more like this 和highlight的特性时应设置为true。
compressed: 字段是压缩的。这可能导致索引和搜索变慢，但会减少存储空间，只有StrField和TextField是可以压缩，这通常适合字段的长度超过200个字符。
multiValued: 字段多于一个值的时候，可设置为true。
positionIncrementGap: 和multiValued一起使用，设置多个值之间的虚拟空白的数量

< tokenizer class =" **solr.WhitespaceTokenizerFactory** " />

空格分词，精确匹配。

< filter class =" **solr.WordDelimiterFilterFactory** " generateWordParts =" **1** " generateNumberParts =" **1** " catenateWords =" **1** " catenateNumbers =" **1** " catenateAll =" **0** " splitOnCaseChange =" **1** " />

在分词和匹配时，考虑 "-"连字符，字母数字的界限，非字母数字字符，这样 "wifi"或"wi fi"都能匹配"Wi-Fi"。

< filter class =" **solr.SynonymFilterFactory** " synonyms =" **synonyms.txt** " ignoreCase =" **true** " expand =" **true** " />

同义词

< filter class =" **solr.StopFilterFactory** " ignoreCase =" **true** " words =" **stopwords.txt** " enablePositionIncrements =" **true** " />

在禁用字（stopword）删除后，在短语间增加间隔

stopword：即在建立索引过程中（建立索引和搜索）被忽略的词，比如is this等常用词。在conf/stopwords.txt维护。

4、fields

< field name =" **id** " type =" **string** " indexed =" **true** " stored =" **true** " required =" **true** " />

- name：标识而已。
- type：先前定义的类型。
- indexed：是否被用来建立索引（关系到搜索和排序）
- stored：是否储存
- compressed：[false]，是否使用gzip压缩（只有TextField和StrField可以压缩）
- mutiValued：是否包含多个值
- omitNorms：是否忽略掉Norm，可以节省内存空间，只有全文本field和need an index-time boost的field需要norm。（具体没看懂，注释里有矛盾）
- termVectors：[false]，当设置true，会存储 term vector。当使用MoreLikeThis，用来作为相似词的field应该存储起来。
- termPositions：存储 term vector中的地址信息，会消耗存储开销。
- termOffsets：存储 term vector 的偏移量，会消耗存储开销。
- default：如果没有属性需要修改，就可以用这个标识下。

< field name =" **text** " type =" **text** " indexed =" **true** " stored =" **false** " multiValued =" **true** " />

包罗万象（有点夸张）的field，包含所有可搜索的text fields，通过copyField实现。

< copyField source =" **cat** " dest =" **text** " />

< copyField source =" **name** " dest =" **text** " />
< copyField source =" **manu** " dest =" **text** " />
< copyField source =" **features** " dest =" **text** " />
< copyField source =" **includes** " dest =" **text** " />

在添加索引时，将所有被拷贝field（如cat）中的数据拷贝到text field中

作用：

- 将多个field的数据放在一起同时搜索，提供速度
- 将一个field的数据拷贝到另一个，可以用2种不同的方式来建立索引。

< dynamicField name =" ***_i** " type =" **int** " indexed =" **true** " stored =" **true** " />

如果一个field的名字没有匹配到，那么就会用动态field试图匹配定义的各种模式。

- "*"只能出现在模式的最前和最后
- 较长的模式会被先去做匹配
- 如果2个模式同时匹配上，最先定义的优先

< dynamicField name =" * " type =" **ignored** " multiValued=" **true** " />

如果通过上面的匹配都没找到，可以定义这个，然后定义个type，当String处理。（一般不会发生）

但若不定义，找不到匹配会报错。

5、其他一些标签

< uniqueKey > **id** </ uniqueKey >

文档的唯一标识， 必须填写这个field（除非该field被标记required="false"），否则solr建立索引报错。

< defaultSearchField > **text** </ defaultSearchField >

如果搜索参数中没有指定具体的field，那么这是默认的域。

< solrQueryParser defaultOperator =" **OR** " />

配置搜索参数短语间的逻辑，可以是"AND|OR"。

二、solrconfig.xml

1、索引配置

mainIndex 标记段定义了控制Solr索引处理的一些因素.

- useCompoundFile：通过将很多 Lucene 内部文件整合到单一一个文件来减少使用中的文件的数量。这可有助于减少 Solr 使用的文件句柄数目，代价是降低了性能。除非是应用程序用完了文件句柄，否则 `false` 的默认值应该就已经足够。
- useCompoundFile：通过将很多Lucene内部文件整合到一个文件，来减少使用中的文件的数量。这可有助于减少Solr使用的文件句柄的数目，代价是降低了性能。除非是应用程序用完了文件句柄，否则false的默认值应该就已经足够了。
- mergeFacor：决定Lucene段被合并的频率。较小的值（最小为2）使用的内存较少但导致的索引时间也更慢。较大的值可使索引时间变快但会牺牲较多的内存。（典型的时间与空间 的平衡配置）
- maxBufferedDocs：在合并内存中文档和创建新段之前，定义所需索引的最小文档数。段是用来存储索引信息的Lucene文件。较大的值可使索引时间变快但会牺牲较多内

存。

- maxMergeDocs：控制可由Solr合并的 Document 的最大数。较小的值（<10,000）最适合于具有大量更新的应用程序。
- maxFieldLength：对于给定的Document，控制可添加到Field的最大条目数，进而阶段该文档。如果文档可能会很大，就需要增加这个数值。然后，若将这个值设置得过高会导致内存不足错误。
- unlockOnStartup：告知Solr忽略在多线程环境中用来保护索引的锁定机制。在某些情况下，索引可能会由于不正确的关机或其他错误而一直处于锁定，这就妨碍了添加和更新。将其设置为true可以禁用启动索引，进而允许进行添加和更新。（锁机制）

2、查询处理配置

query标记段中以下一些与缓存无关的特性：

- maxBooleanClauses：定义可组合在一起形成以个查询的字句数量的上限。正常情况1024已经足够。如果应用程序大量使用了通配符或范围查询，增加这个限制将能避免当值超出时，抛出TooMangClausesException。
- enableLazyFieldLoading：如果应用程序只会检索Document上少数几个Field，那么可以将这个属性设置为 true。懒散加载的一个常见场景大都发生在应用程序返回一些列搜索结果的时候，用户常常会单击其中的一个来查看存储在此索引中的原始文档。初始的现实常常只需要现实很短的一段信息。若是检索大型的Document，除非必需，否则就应该避免加载整个文档。

query部分负责定义与在Solr中发生的时间相关的几个选项：

概念：Solr（实际上是Lucene）使用称为Searcher的Java类来处理Query实例。Searcher将索引内容相关的数据加载到内存中。根据索引、CPU已经可用内存的大小，这个过程可能需要较长的一段时间。要改进这一设计和显著提高性能，Solr引入了一张“温暖”策略，即把这些新的Searcher联机以便为现场用户提供查询服务之前，先对它们进行“热身”。

- newSearcher和firstSearcher事件，可以使用这些事件来制定实例化新Searcher或第一个Searcher时，应该执行哪些查询。如果应用程序期望请求某些特定的查询，那么在创建新Searcher或第一个Searcher时就应该反注释这些部分并执行适当的查询。

query中的智能缓存：

- filterCache：通过存储一个匹配给定查询的文档 id 的无序集，过滤器让 Solr 能够有效提高查询的性能。缓存这些过滤器意味着对Solr的重复调用可以导致结果集的快速查找。更常见的场景是缓存一个过滤器，然后再发起后续的精炼查询，这种查询能使用过滤器来限制要搜索的文档数。

- queryResultCache：为查询、排序条件和所请求文档的数量缓存文档 id 的有序集合。
- documentCache：缓存Lucene Document，使用内部Lucene文档id（以便不与Solr唯一id相混淆）。由于Lucene的内部Document id 可以因索引操作而更改，这种缓存不能自热。
- Named caches：命名缓存是用户定义的缓存，可被 Solr定制插件 所使用。

其中filterCache、queryResultCache、Named caches（如果实现了org.apache.solr.search.CacheRegenerator）可以自热。

每个缓存声明都接受最多四个属性：

- class：是缓存实现的Java名
- size：是最大的条目数
- initialSize：是缓存的初始大小
- autoWarmCount：是取自旧缓存以预热新缓存的条目数。如果条目很多，就意味着缓存的hit会更多，只不过需要花更长的预热时间。

对于所有缓存模式而言，在设置缓存参数时，都有必要在内存、cpu和磁盘访问之间进行均衡。统计信息管理页（管理员界面的Statistics）对于分析缓存的 hit-to-miss 比例以及微调缓存大小的统计数据都非常有用。而且，并非所有应用程序都会从缓存受益。实际上，一些应用程序反而会由于需要将某个永远也用不到的条目存储在缓存中这一额外步骤而受到影响。

posted on 2012-05-30 14:18 CONAN 阅读(10838) 评论(0) 编辑 收藏 所属分类: Solr