# solrconfig.xml

solrconfig.xml配置文件主要定义了SOLR的一些处理规则，包括索引数据的存放位置，更新，删除，查询的一些规则配置。

## datadir节点

*<dataDir>${solr.data.dir:d:/Server/Solr/data}</dataDir>*  定义了索引数据和日志文件的存放位置

## luceneMatchVersion

*<luceneMatchVersion>4.10.1</luceneMatchVersion>* 表示solr底层使用的是lucene4.8

## lib

*<lib dir="../../../contrib/extraction/lib"regex=".\*\.jar"/>* 表示solr引用包的位置,当dir对应的目录不存在时候，会忽略此属性

## directoryFactory

索引存储方案，共有以下存储方案

1. solr.StandardDirectoryFactory,这是一个基于文件系统存储目录的工厂，它会试图选择最好的实现基于你当前的操作系统和Java虚拟机版本。
2. solr.SimpleFSDirectoryFactory,适用于小型应用程序，不支持大数据和多线程。
3. solr.NIOFSDirectoryFactory,适用于多线程环境，但是不适用在windows平台（很慢），是因为JVM还存在bug。
4. solr.MMapDirectoryFactory,这个是solr3.1到4.0版本在linux64位系统下默认的实现。它是通过使用虚拟内存和内核特性调用mmap去访问存储在磁盘中的索引文件。它允许lucene或solr直接访问I/O缓存。如果不需要近实时搜索功能，使用此工厂是个不错的方案。
5. solr.NRTCachingDirectoryFactory,此工厂设计目的是存储部分索引在内存中，从而加快了近实时搜索的速度。
6. solr.RAMDirectoryFactory,这是一个内存存储方案，不能持久化存储，在系统重启或服务器crash时数据会丢失。且不支持索引复制

<directoryFactory class="${solr.directoryFactory:solr.NRTCachingDirectoryFactory}" name="DirectoryFactory">

<str name="solr.hdfs.home">${solr.hdfs.home:}</str>

<str name="solr.hdfs.confdir">${solr.hdfs.confdir:}</str>

<str name="solr.hdfs.blockcache.enabled">${solr.hdfs.blockcache.enabled:true}</str>

<str name="solr.hdfs.blockcache.global">${solr.hdfs.blockcache.global:true}</str>

</directoryFactory>

## codecFactory

编解码工厂允许使用自定义的编解码器。例如：如果想启动per-field DocValues格式, 可以在solrconfig.xml里面设置SchemaCodecFactory：

docValuesFormat="Lucene42": 这是默认设置，所有数据会被加载到堆内存中。

docValuesFormat="Disk": 这是另外一个实现，将部分数据存储在磁盘上。

docValuesFormat="SimpleText": 文本格式，非常慢，用于学习。

<codecFactory class="solr.SchemaCodecFactory"/>

<schemaFactory class="ClassicIndexSchemaFactory"/>

## indexconfig

用于设置索引的低级别的属性

1、<filter class="solr.LimitTokenCountFilterFactory" maxTokenCount="10000"/>//限制token最大长度

2、<writeLockTimeout>1000</writeLockTimeout>//IndexWriter等待解锁的最长时间（毫秒）。

3、<maxIndexingThreads>8</maxIndexingThreads>//

4、<useCompoundFile>false</useCompoundFile>//solr默认为false。如果为true,索引文件减少,检索性能降低,追求平衡。

5、<ramBufferSizeMB>100</ramBufferSizeMB>//缓存

6、<maxBufferedDocs>1000</maxBufferedDocs>//同上。两个同时定义时命中较低的那个。

7、<mergePolicy class="org.apache.lucene.index.TieredMergePolicy">

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

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

</mergePolicy>

//合并策略。

8、<mergeFactor>10</mergeFactor>//合并因子,每次合并多少个segments。

9、<mergeScheduler class="org.apache.lucene.index.ConcurrentMergeScheduler"/>//合并调度器。

10、<lockType>${solr.lock.type:native}</lockType>//锁工厂。

11、<unlockOnStartup>false</unlockOnStartup>//是否启动时先解锁。

12、<termIndexInterval>128</termIndexInterval>//Lucene loads terms into memory 间隔

13、<reopenReaders>true</reopenReaders>//重新打开,替代先关闭-再打开。

14、<deletionPolicy class="solr.SolrDeletionPolicy">//提交删除策略,必须实现org.apache.lucene.index.IndexDeletionPolicy

15、<str name="maxCommitsToKeep">1</str>

16、<str name="maxOptimizedCommitsToKeep">0</str>

17、<str name="maxCommitAge">30MINUTES</str> OR <str name="maxCommitAge">1DAY</str><br>

18、 <infoStream file="INFOSTREAM.txt">false</infoStream>//相当于把创建索引时的日志输出。

<lockType>${solr.lock.type:native}</lockType>

设置索引库的锁方式，主要有三种：

1. single：适用于只读的索引库，即索引库是定死的，不会再更改
2. native：使用本地操作系统的文件锁方式，不能用于多个solr服务共用同一个索引库。Solr3.6 及后期版本使用的默认锁机制。
3. simple：使用简单的文件锁机制

## updateHandler

<updateLog>

<str name="dir">${solr.ulog.dir:}</str>

</updateLog>

设置索引库更新日志，默认路径为solr home下面的data/tlog。随着索引库的频繁更新，tlog文件会越来越大，所以建议提交索引时采用硬提交方式<autoCommit>，即批量提交。

<autoCommit>

<maxTime>15000</maxTime>

<maxDocs>10000</maxDocs>

<openSearcher>false</openSearcher>

</autoCommit>

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。

<updateHandler class="solr.DirectUpdateHandler2">

<!-- 允许事务日志 -->

<updateLog>

<str name="dir">${solr.ulog.dir:}</str>

</updateLog>

<!--

在满足一定条件时自动提交。maxDocs/maxTime/openSearcher

-->

<autoCommit>

<maxTime>15000</maxTime>

<openSearcher>false</openSearcher>

</autoCommit>

<!-- 软提交VS硬提交 -->

<!--

<autoSoftCommit>

<maxTime>1000</maxTime>

</autoSoftCommit>

-->

<!--

更新相关事件监听器

postCommit - fired after every commit or optimize command

postOptimize - fired after every optimize command

-->

<!-- The RunExecutableListener executes an external command from a

hook such as postCommit or postOptimize.

exe - the name of the executable to run

dir - dir to use as the current working directory. (default=".")

wait - the calling thread waits until the executable returns.

(default="true")

args - the arguments to pass to the program. (default is none)

env - environment variables to set. (default is none)

-->

<!--

<listener event="postCommit" class="solr.RunExecutableListener">

<str name="exe">solr/bin/snapshooter</str>

<str name="dir">.</str>

<bool name="wait">true</bool>

<arr name="args"> <str>arg1</str> <str>arg2</str> </arr>

<arr name="env"> <str>MYVAR=val1</str> </arr>

</listener>

-->

</updateHandler>

## Query 查询

<maxBooleanClauses>1024</maxBooleanClauses>

设置boolean 查询中，最大条件数。在范围搜索或者前缀搜索时，会产生大量的 boolean 条件，如果条件数达到这个数值时，将抛出异常，限制这个条件数，可以防止条件过多查询等待时间过长。

<filterCache class="solr.FastLRUCache" size="512" initialSize="512" autowarmCount="0"/>

<queryResultCache class="solr.LRUCache" size="512" initialSize="512" autowarmCount="0"/>

<documentCache class="solr.LRUCache" size="512" initialSize="512" autowarmCount="0"/>

<queryResultMaxDocsCached>200</queryResultMaxDocsCached>

<maxWarmingSearchers>2</maxWarmingSearchers>

## Request Dispatcher请求转发器

<!-- Request Dispatcher

主要是介绍当有请求访问SolrCore时SolrDispatchFilter如何处理。

handleSelect是一个以前版本中遗留下来的属性,会影响请求的对应行为（比如/select?qt=XXX）。

当handleSelect="true"时导致SolrDispatchFilter将请求转发给qt指定的处理器（前提是/select已经注册）。

当handleSelect="false"时会直接访问/select,若/select未注册则为404。

-->

<requestDispatcher handleSelect="false" >

<!-- Request Parsing：请求解析

这些设置说明Solr Requests如何被解析,以及对ContentStreams有什么限制。

enableRemoteStreaming - 是否允许使用stream.file和stream.url参数来指定远程streams。

multipartUploadLimitInKB - 指定多文件上传时Solr允许的最大的size。

formdataUploadLimitInKB - 表单通过POST请求发送的最大size

-->

<requestParsers enableRemoteStreaming="true"

multipartUploadLimitInKB="2048000"

formdataUploadLimitInKB="2048"/>

<!-- HTTP Caching

设置HTTP缓存的相关参数。

-->

<httpCaching never304="true" />

<!--

<httpCaching never304="true" >

<cacheControl>max-age=30, public</cacheControl>

</httpCaching>

-->

<!--

<httpCaching lastModifiedFrom="openTime"

etagSeed="Solr">

<cacheControl>max-age=30, public</cacheControl>

</httpCaching>

-->

</requestDispatcher>

# schema.xml

schema.xml文件里面主要定义了索引数据类型，索引字段等信息。

## fieldtype

fieldtype节点主要用来定义数据类型。

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

*<!-- boolean type: "true" or "false" -->*

*<fieldType name="boolean" sortMissingLast="true" class="solr.BoolField"/>*

name指定的是节点定义的名称

class指向org.apache.solr.analysis中定义的类型名称

fieldtype还可以自己定义当前类型建立索引和查询数据的时候使用的查询分析器。

tokenizer指定分词器

filter指定过滤器

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

<analyzer type="index">

<tokenizer class="solr.StandardTokenizerFactory"/>

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

<filter class="solr.LowerCaseFilterFactory"/>

</analyzer>

<analyzer type="query">

<tokenizer class="solr.StandardTokenizerFactory"/>

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

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

<filter class="solr.LowerCaseFilterFactory"/>

</analyzer>

</fieldType>

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。

positionIncrementGap=100 只对 multiValue = true 的fieldType有意义。

StrField类型不被分析，而是被逐字地索引/存储

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）

## field

field节点指定建立索引和查询数据的字段。

name代表数据字段名称

type代表数据类型，也就是之前定义的fieldtype

indexed代表是否被索引

stored代表是否被存储

multiValued是否有多个值，如果字段可能有多个值，尽可能设为true

\_version\_节点和\_root\_节点是必须保留的，不能删除

<field name="\_version\_" stored="true" indexed="true" type="long"/>

<field name="\_root\_" stored="false" indexed="true" type="string"/>

<field name="ProductCode" stored="true" indexed="true" type="string" multiValued="false" required="true"/>

<field name="ProductName" stored="true" indexed="true" type="text\_general"/>

## copyfield

通过这个节点，可以把一个字段的值复制到另一个字段中，也可以把多个字段的值同时复制到另一个字段中，这样搜索的时候都可以根据一个字段来进行搜索。

<copyField source="ProductName" dest="text"/>

<copyField source="ProductCode" dest="text"/>

## dynamicField

dynamicField 表示动态字段，可以动态定义一个字段，只要符合规则的字段都可以。

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

\*\_i只要以\_i结尾的字段都满足这个定义。

## uniquekey

<uniqueKey>id</uniqueKey>

uniquekey节点是文档的唯一标示，相当于主键，每次更新，删除的时候都根据这个字段来进行操作。必须填写。

## defaultSearchField

<defaultSearchField>text</defaultSearchField>

defaultSearchField指定搜索的时候默认搜索字段的值。

## solrQueryParser

<solrQueryParser defaultOperator="OR"/>

solrQueryParser指定搜索时多个词之间的关系，可以是or,and两种。

## 性能优化

1. 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；
2. 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；
3. 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；
4. 将所有的 text fields的index都设置成false，然后使用copyField将他们都复制到一个总的 text field上，然后进行搜索。