全文检索开发说明文档

---- solr

目录

[全文检索开发说明文档 1](#_Toc301861221)

[1. Solr概述 3](#_Toc301861222)

[1.1 solr简介 3](#_Toc301861223)

[1.2 Solr的特性 3](#_Toc301861224)

[1.3 源码结构 3](#_Toc301861225)

[1.3.1 目录结构说明 3](#_Toc301861226)

[1.3.2 Solr home说明 5](#_Toc301861227)

[2. 安装准备及环境和版本介绍 5](#_Toc301861228)

[3. 安装搭建solr服务器 6](#_Toc301861229)

[3.1 搭建基本服务步骤 6](#_Toc301861230)

[3.2 搭建多核服务步骤 8](#_Toc301861231)

[3.2.1 MultiCore的配置方法 8](#_Toc301861232)

[3.2.2 为何使用多core ？ 10](#_Toc301861233)

[3.3 一些关键配置文件解析说明 10](#_Toc301861234)

[3.3.1 schema.xml 10](#_Toc301861235)

[3.3.2  solrconfig.xml 13](#_Toc301861236)

[4. 安装配置中文分词-IK 20](#_Toc301861237)

[4.1 Solr中配置IK分词步骤 20](#_Toc301861238)

[4.2 IK中停止词的配置 21](#_Toc301861239)

[5. 数据源的配置 23](#_Toc301861240)

[5.1 基于Mysql数据库的索引数据源配置。 23](#_Toc301861241)

[5.1.1 全量索引 23](#_Toc301861242)

[5.1.2 增量索引 26](#_Toc301861243)

[5.1.3 多数据源 27](#_Toc301861244)

[5.2 基于程序动态添加索引的配置。 28](#_Toc301861245)

[6. Solr的应用分析说明。 29](#_Toc301861246)

[6.1 搜索引擎的规划设计 30](#_Toc301861247)

[6.1.1 定义业务模型 30](#_Toc301861248)

[6.1.2 定制索引服务 32](#_Toc301861249)

[6.1.3 定制搜索服务 33](#_Toc301861250)

[6.2 搜索引擎配置 33](#_Toc301861251)

[6.2.1 Solr Schema 设计(如何定制索引的结构?) 33](#_Toc301861252)

[7. Solrj的基本使用。 36](#_Toc301861253)

[7.1 获得CommonsHttpSolrServer。 36](#_Toc301861254)

[7.2 建立索引。 36](#_Toc301861255)

[7.3 执行查询。 36](#_Toc301861256)

[7.4 删除索引。 37](#_Toc301861257)

[7.5 设置权重 37](#_Toc301861258)

[7.6 设置查询到的条数（分页时用） 37](#_Toc301861259)

[7.7 设置排序字段 37](#_Toc301861260)

[7.8 设置高亮 37](#_Toc301861261)

[7.9 读取结果 38](#_Toc301861262)

[6.9.1 读取结果—基本 38](#_Toc301861263)

[6.9.2 读取结果—对象 38](#_Toc301861264)

[6.9.3 读取高亮结果—对象 38](#_Toc301861265)

[8. 附录---Solr的实际应用测试报告 38](#_Toc301861266)

[8.1 线下压力测试报告 38](#_Toc301861267)

[8.2 线上环境运行报告 39](#_Toc301861268)

[9. 附录---solr性能调优 39](#_Toc301861269)

[9.1 Schema Design Considerations 39](#_Toc301861270)

[indexed fields 39](#_Toc301861271)

[stored fields 40](#_Toc301861272)

[9.2 Configuration Considerations 40](#_Toc301861273)

[mergeFactor 40](#_Toc301861274)

[mergeFactor Tradeoffs 40](#_Toc301861275)

[9.3 Cache autoWarm Count Considerations 41](#_Toc301861276)

[9.4 Cache hit rate（缓存命中率） 41](#_Toc301861277)

[9.5 Explicit Warming of Sort Fields 41](#_Toc301861278)

[9.6  Optimization Considerations 41](#_Toc301861279)

[9.7 Updates and Commit Frequency Tradeoffs 42](#_Toc301861280)

[9.8 Query Response Compression 43](#_Toc301861281)

[9.9 Embedded vs HTTP Post 43](#_Toc301861282)

[9.10  RAM Usage Considerations（内存方面的考虑） 43](#_Toc301861283)

[OutOfMemoryErrors 43](#_Toc301861284)

[Memory allocated to the Java VM 43](#_Toc301861285)

# Solr概述

## 1.1 solr简介

Solr 是一个开源的企业级搜索服务器，底层使用易于扩展和修改的Java 来实现，Solr 底层的核心技术是使用Apache Lucene 来实现的。

基于Solr实现站内搜索。封装及扩展性较好，提供了较为完备的解决方案，因此在门户社区中采用此方案。

## 1.2 Solr的特性

Solr 主要特性有：强大的全文检索功能，高亮显示检索结果，动态集群，数据库接口和电子文档（Word ，PDF 等）的处理。而且Solr 具有高度的可扩展，支持分布搜索和索引的复制。

## 1.3 源码结构

### 1.3.1 目录结构说明

我们下载的Solr包后，进入Solr所在的目录，我们可以看到以下几个目录：build、client、dist、example、lib、site、src。下面分别对其进行介绍。

1. **build：**该目录是在ant build过程中生成的，其中包含了未被打包成jar或是war的class文件以及一些文档文件。
2. **client：**该目录包含了特定语言的Solr客户端API，使得使用其他语言的用户能通过HTTP用XML与Solr进行通话。现在该目录里面虽然包含javascript、python、ruby三个子目录，但是到目前为止只包含一部分的ruby的代码，其他语言仍是空的。另外，Solr的Java客户端称为SolrJ，其代码位于src/solrj目录下面。在之后的文章中我会详细介绍Solr客户端的使用。
3. **dist：**该目录包含build过程中产生的war和jar文件，以及相关的依赖文件。还记得上一篇文章中，我们在build 1.4版本的Solr源代码后需要部署example吗？其实就是将该目录下面的apache-solr-1.4.war部署到Jetty上面去，并重命名为solr.war。
4. **example：**这个目录实际上是Jetty的安装目录。其中包含了一些样例数据和一些Solr的配置。

其中一些子目录也比较重要，这里也对它们稍作介绍。

* example/etc：该目录包含了Jetty的配置，在这里我们可以将Jetty的默认端口从8983改为80端口。
* 将其中的8983端口换成80端口。注意更改端口后启动Jetty可能会提示你没有权限，你需要使用sudo java -jar start.jar来运行。
* example/multicore：该目录包含了在Solr的multicore中设置的多个home目录。在之后的文章中我会对其进行介绍。
* example/solr：该目录是一个包含了默认配置信息的Solr的home目录。

详见下面的“[solr home说明](file:///E:\全文检索\solr\开源企业搜索引擎SOLR的应用教程.doc#_Solr_home说明)”

* example/webapps：Jetty的webapps目录，该目录通常用来放置Java的Web应用程序。在Solr中，前面提到的solr.war文件就部署在这里。

1. lib：该目录包含了所有Solr的API所依赖的库文件。其中包括Lucene，Apache commons utilities和用来处理XML的Stax库。
2. site：该目录仅仅包含了Solr的官网的网页内容，以及一些教程的PDF文档。
3. src：该目录包含了Solr项目的整个源代码。这里对其各个子目录也做相应的介绍。

* src/java：该目录存放的是Solr使用Java编写的源代码。
* src/scripts：该目录存放的是配置Solr服务器的Unix Bash Shell脚本，在后面介绍多服务器配置中将会有重要的作用。
* src/solrj：前面提到过该目录存放的是Solr的Java版本的客户端代码。
* src/test：该目录存放的是测试程序的源代码和测试文件。
* src/webapp：该目录存放的是管理Solr的Web页面，包括Servlet和JSP文件，其构成了前面提到的WAR文件。管理Solr的JSP页面在web/admin目录下面，如果你有兴趣折腾Solr可以找到相应的JSP的页面对其进行设置

### 1.3.2 Solr home说明

所谓的Solr home目录实际上是一个运行的Solr实例所对应的配置和数据（Lucene索引）。在上一篇文章中我提到过在Solr的example/solr目录就是一个Solr用做示例的默认配置home目录。实际上example/multicore也是一个合法的Solr home目录，只不过是用来做mult-core设置的。那么我们来看看example/solr这个目录里面都有些什么。

example/solr目录下主要有以下一些目录和文件：

1. **bin：**如果你需要对Solr进行更高级的配置，该目录建议用来存放Solr的复制脚本。
2. **conf ：**该目录下面包含了各种配置文件，下面列出了两个最为重要的配置文件。其余的.txt和.xml文件被这两个文件所引用，如用来对文本进行特殊的处理。

* **conf/schema.xml：**该文件是索引的schema，包含了域类型的定义以及相关联的analyzer链。
* **conf/solrconfig.xml：**该文件是Solr的主配置文件。
* **conf/xslt：**该目录包含了各种XSLT文件，能将Solr的查询响应转换成不同的格式，如：Atom/RSS等。

1. **data：**包含了Lucene的二进制索引文件。
2. **lib：**该目录是可选的。用来放置附加的Java JAR文件，Solr在启动时会自动加载该目录下的JAR文件。这就使得用户可以对Solr的发布版本（solr.war）进行扩展。如果你的扩展并不对Solr本身进行修改，那么就可以将你的修改部署到JAR文件中放到这里。

# 安装准备及环境和版本介绍

|  |  |
| --- | --- |
| 项目 | 版本号 |
| Tomcat | Tomcat 7 |
| Solr  Jdk | apache-solr-3.3.0  Jdk 1.6 |
| IK | IKAnalyzer3.2.5Stable.jar |

说明：solr下载：<http://www.apache.org/dyn/closer.cgi/lucene/solr/>



选择3.3.0版本下载

# 安装搭建solr服务器

## 搭建基本服务步骤

1. 准备tomcat7. 修改Tomcat 的编码设置，修改为UTF-8。

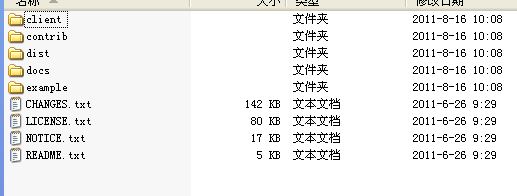
配置文件conf\server.xml，修改为：

<Connector port="8983" protocol="HTTP/1.1" connectionTimeout="20000"

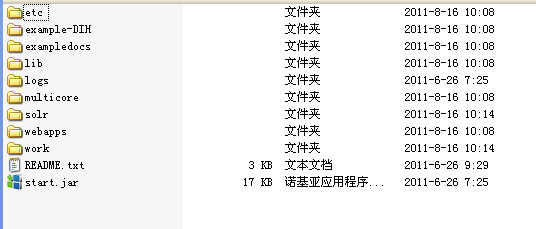
redirectPort="8443" URIEncoding="UTF-8"/>

注：如不添加，中文检索时因为乱码导致搜索不到。

1. 新建solr主目录。如d:/solr/home
2. 增加环境变量，**name**=solr.home；**value**= d:/solr/home
3. 解压下载好的apache-solr-3.3.0.jar.得到文件夹路径为：



1. 展开eamples。



1. 复制example/solr 到主目录（d:/solr/home）下。
2. 复制webapps/solr.war 到tomcat的安装目录webapps下
3. 启动tomcat，solr.war自动解压。修改web.xml

修改加红的地方为solr主目录。

<env-entry>

<env-entry-name>solr/home</env-entry-name>

<env-entry-value>D:\studio\Server\solrhome\solr</env-entry-value>

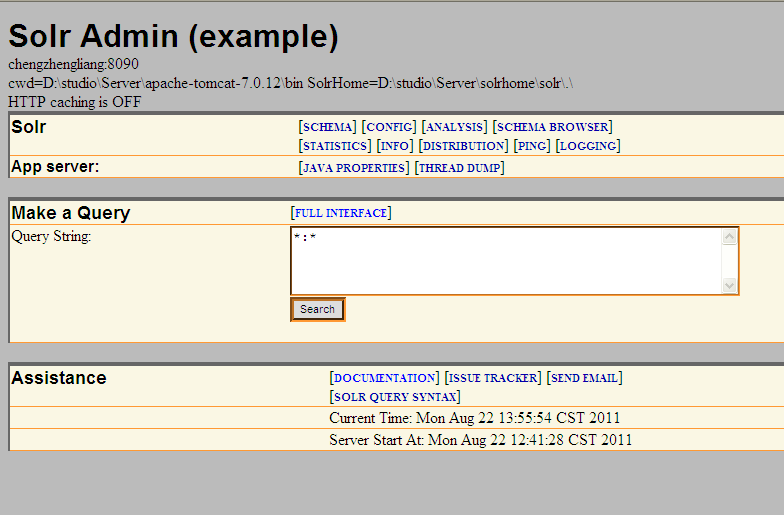
<env-entry-type>java.lang.String</env-entry-type>

</env-entry>

1. 重启tomcat，<http://localhost:8090/solr/> 点击进去看到页面



1. 点击去,或者<http://localhost:8090/solr/admin/> 进入页面：



11．可以输入Query String 中查询，进行查询分析。

## 搭建多核服务步骤

### 3.2.1 MultiCore的配置方法

1、找到solr下载包中的example文件夹，在它的下面有个multicore文件夹，将这个文件夹下面的core0、core1和solr.xml拷贝到 c:\solr-tomcat\solr下面。

注意：有一个 solr.xml（这只是默认文件，当然也可以指定别的文件），如：

<?xml version="1.0" encoding="UTF-8" ?>

<solr persistent="false">

<cores adminPath="/admin/cores">

<core name="core0" instanceDir="core0" />

<core name="core1" instanceDir="core1" />

</cores>

</solr>

这个文件是告诉solr应该加载哪些core，<cores>……</cores>里有 core0、core1。core0（可以类比以前的solr.home）/conf目录下有schema.xml与solrconfig.xml，可以把实际应用的/solr/conf/schema.xml复制过来（注意：solrconfig.xml不要复制）。

2、启动tomcat，访问应用，就可以看到有 Admin core0 和 Admin core1

<http://192.168.10.85:18080/solr/>

<http://192.168.10.85:18080/solr/core0/admin/>

<http://192.168.10.85:18080/solr/core1/admin/>

<http://192.168.10.85:18080/solr/core1/admin/>[analysis.jsp](http://localhost:8080/solr/admin/analysis.jsp)

3、采用上面的默认solr.xml，索引文件将存放在同一个目录下面，在这里将存放在C:\solr-tomcat\solr\data，如果你想更改目录，或者两个应用存放在不同的目录，请参见下面的xml。

<core name="core0" instanceDir="core0">

<property name="dataDir" value="/opt/solr-tomcat/solr/data/core0" />

</core>

<core name="core1" instanceDir="core1">

<property name="dataDir" value="/opt/solr-tomcat/solr/data/core1" />

</core>

一些关键的配置值是：

1.Persistent="false"指明运行时的任何修改我们不做保存。如拷贝。如果你想保存从启动起的一些改动，那就把 persistent设置为true。如果你的index策略是完成建index到一个纯净的core中然后交换到活动core 那么你绝对应该设为true。

sharedLib="lib"指明了所有core的jar文件的lib目录。如果你有一个core有自己需要的jar文件，那么你可以把他们置入到core/lib目录。例如：karaoke core 使用 Solr Cell来索引化富文本内容，因此那些用来解析和抽取富文本的jar文件被放到./examples/cores/karaoke/lib/.

### 3.2.2 为何使用多core ？

Solr实例支持多core比启用多index要好（do more）。多core同时解决了在生产环境下的一些关键需求：

1.重建索引

2.测试配置变更

3.合并索引

4.运行时重命名core

为何多core不是默认的？

多core是1.3版本中才加的，1.4后更成熟。我们强烈建议你使用多core，既是你现在的solr.xml只配置了一个core，虽然会比单个索引稍复杂，但可以带来管理core上的好处。或许一天单个core可能最终RELOAD and STATUS命令，又或许单个core最终会被废禁。多个core会是Solr将来支持大规模分布式索引的关键。因此，以后可以期待更多。

你可以得到更多的关于Solr的资料：<http://wiki.apache.org/solr/CoreAdmin>。

## 一些关键配置文件解析说明

### 3.3.1 schema.xml

schema.xml，这个相当于数据表配置文件，它定义了加入索引的数据的数据类型的。主要包括types、fields和其他的一些缺省设置。

注:schema.xml里有一个uniqueKey,的配置，这里将id字段作为索引文档的唯一标识符，非常重要。

<uniqueKey>id</uniqueKey>

#### FieldType（类型）

首先需要在types结点内定义一个FieldType子结点，包括name,class,positionIncrementGap等等一些参数，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。在FieldType定义的时候最重要的就是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。例如：

|  |
| --- |
| <fieldType name="text" class="solr.TextField" positionIncrementGap="100">  <analyzer type="index">  <tokenizer class="solr.WhitespaceTokenizerFactory"/>  <!-- in this example, we will only use synonyms at query time  <filter class="solr.SynonymFilterFactory" synonyms="index\_synonyms.txt" ignoreCase="true" expand="false"/>  -->  <!-- Case insensitive stop word removal.  enablePositionIncrements=true ensures that a 'gap' is left to  allow for accurate phrase queries.  -->  <filter class="solr.StopFilterFactory"  ignoreCase="true"  words="stopwords.txt"  enablePositionIncrements="true"  />  <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0" splitOnCaseChange="1"/>  <filter class="solr.LowerCaseFilterFactory"/>  <filter class="solr.EnglishPorterFilterFactory" protected="protwords.txt"/>  <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>  </analyzer>  ……  </fieldType> |

在index的analyzer中使用 solr.WhitespaceTokenizerFactory这个分词包，就是空格分词，然后使用 solr.StopFilterFactory，solr.WordDelimiterFilterFactory，solr.LowerCaseFilterFactory，solr.EnglishPorterFilterFactory，solr.RemoveDuplicatesTokenFilterFactory 这几个过滤器。在向索引库中添加text类型的索引的时候，Solr会首先用空格进行分词，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。Solr的analysis包并没有带支持中文分词的包

#### Fields（字段）

接下来的工作就是在fields结点内定义具体的字段（类似数据库中的字段），就是filed，filed定义包括name,type（为之前定义过的各种FieldType）,indexed（是否被索引）,stored（是否被储存），multiValued（是否有多个值）等等。

例：

<fields>

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

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

<field name="summary" type="text" indexed="true" stored="true" />

<field name="author" type="string" indexed="true" stored="true" />

<field name="date" type="date" indexed="false" stored="true" />

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

<field name="keywords" type="keyword\_text" indexed="true" stored="false" multiValued="true" />

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

</fields>

field的定义相当重要，有几个技巧需注意一下，对可能存在多值得字段尽量设置 multiValued属性为true，避免建索引是抛出错误；如果不需要存储相应字段值，尽量将stored属性设为false。

#### copyField（复制字段）

建议建立了一个拷贝字段，将所有的全文字段复制到一个字段中，以便进行统一的检索：

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

并在拷贝字段结点处完成拷贝设置：

<copyField source="name" dest="all"/>

<copyField source="summary" dest="all"/>

说明 ：“拷贝字段”就是查询的时候不用再输入：userName:张三 and userProfile:张三的个人简介。直接可以输入"张三"就可以将“名字”含“张三”或者“简介”中含“张三”的又或者“名字”和“简介”都含有“张三”的查询出来。他将需要查询的内容放在了一个字段中，并且默认查询该字段设为该字段就行了。

#### dynamicField（动态字段）

除此之外，还可以定义动态字段，所谓动态字段就是不用指定具体的名称，只要定义字段名称的规则，例如定义一个 dynamicField，name 为\*\_i，定义它的type为text，那么在使用这个字段的时候，任何以\_i结尾的字段都被认为是符合这个定义的，例如：name\_i，gender\_i，school\_i等。

schema.xml配置文件大体上就是这样，更多细节请参见solr wiki：

<http://wiki.apache.org/solr/SchemaXml>

### 3.3.2  solrconfig.xml

在配置方面，solrconfig.xml 文件不仅指定了 Solr 如何处理索引、突出显示、分类、搜索以及其他请求，还指定了用于指定缓存的处理方法的属性，以及用于指定 Lucene 管理索引的方法的属性。配置取决于模式，但模式不取决于配置。

solrconfig.xml文件包含了大部分的参数用来配置Solr本身的。

* **dataDir parameter：**<dataDir>/var/data/solr</dataDir>

用来指定一个替换原先在Solr目录下默认存放所有的索引数据，可以在Solr目录以外的任意目录中。如果复制使用后应该符合该参数。如果这个目录不是绝对路径的话，那么应该以当前的容器为相对路径。

* **mainIndex ： 这个参数的值用来控制合并多个索引段。**

<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。

mergeFactor：决定低水平的 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。

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的内存。

* **maxMergeDocs：** 控制可由 Solr ,000) 最适合于具有合并的 Document 的最大数。较小的值 (< 10大量更新的应用程序。该参数不允许lucene在任何索引段里包含比这个值更多的文档，但是，多余的文档可以创建一个新的索引段进行替换。
* **maxFieldLength：** 对于给定的 Document，控制可添加到 Field 的最大条目数，进而截断该文档。如果文档可能会很大，就需要增加这个数值。然而，若将这个值设置得过高会导致内存不足错误。
* **unlockOnStartup：** 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 可以禁用启动锁定，进而允许进行添加和更新。

<mainIndex>

<!-- lucene options specific to the main on-disk lucene index -->

    <useCompoundFile>false</useCompoundFile>

    <mergeFactor>10</mergeFactor>

    <maxBufferedDocs>1000</maxBufferedDocs>

    <maxMergeDocs>2147483647</maxMergeDocs>

    <maxFieldLength>10000</maxFieldLength>

  </mainIndex>

* **updateHandler：**这个更新处理器主要涉及底层的关于如何更新处理内部的信息。（此参数不能跟高层次的配置参数Request Handlers对处理发自客户端的更新相混淆）。

<updateHandler class="solr.DirectUpdateHandler2">

<!-- Limit the number of deletions Solr will buffer during doc updating.

         Setting this lower can help bound memory use during indexing.

    -->   
缓冲更新这么多的数目，设置如下比较低的值，可以约束索引时候所用的内存   
    <maxPendingDeletes>100000</maxPendingDeletes>   
等待文档满足一定的标准后将自动提交，未来版本可以扩展现有的标准   
    <!-- autocommit pending docs if certain criteria are met.  Future versions may expand the available   
     criteria -->   
    <autoCommit>    
      <maxDocs>10000</maxDocs> <!-- maximum uncommited docs before autocommit triggered -->   
触发自动提交前最多可以等待提交的文档数量   
      <maxTime>86000</maxTime> <!-- maximum time (in MS) after adding a doc before an autocommit is triggered -->   
在添加了一个文档之后，触发自动提交之前所最大的等待时间   
    </autoCommit>

这个参数用来配置执行外部的命令。   
一个postCommit的事件被触发当每一个提交之后   
<listener event="postCommit" class="solr.RunExecutableListener">   
      <str name="exe">snapshooter</str>   
      <str name="dir">solr/bin</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>   
exe--可执行的文件类型   
dir--可以用该目录做为当前的工作目录。默认为"."   
wait--调用线程要等到可执行的返回值   
args--传递给程序的参数 默认nothing   
env--环境变量的设置 默认nothing   
  
<query>   
    <!-- Maximum number of clauses in a boolean query... can affect range   
         or wildcard queries that expand to big boolean queries.    
一次布尔查询的最大数量，可以影响查询的范围或者进行通配符的查询，借此来扩展一个更强大的查询。   
         An exception is thrown if exceeded.   
    -->   
    <maxBooleanClauses>1024</maxBooleanClauses>   
  
<query>:   
控制跟查询相关的一切东东。   
  
Caching：修改这个参数可以做为索引的增长和变化。   
  
<!-- Cache used by SolrIndexSearcher for filters (DocSets),   
         unordered sets of \*all\* documents that match a query.   
在过滤器中过滤文档集合的时候，或者是一个无序的所有的文档集合中将在在SolrIndexSearcher中使用缓存来匹配符合查询的所有文档。   
         When a new searcher is opened, its caches may be prepopulated   
         or "autowarmed" using data from caches in the old searcher.   
当一次搜索被打开，它可以自动的或者预先从旧的搜索中使用缓存数据。   
         autowarmCount is the number of items to prepopulate.   
autowarmCount这个值是预先设置的数值项。   
  For LRUCache,   
         the autowarmed items will be the most recently accessed items.   
在LRUCache中，这个autowarmed 项中保存的是最近访问的项。   
       Parameters: 参数选项   
         class - the SolrCache implementation (currently only LRUCache)实现SolrCache接口的类 当前仅有LRUCache   
              
         size - the maximum number of entries in the cache   
在cache中最大的上限值   
         initialSize - the initial capacity (number of entries) of   
           the cache.  (seel java.util.HashMap)   
在cache中初始化的数量   
         autowarmCount - the number of entries to prepopulate from   
           and old cache.   
从旧的缓存中预先设置的项数。   
         -->   
    <filterCache   
      class="solr.LRUCache"   
      size="512"   
      initialSize="512"   
      autowarmCount="256"/>   
  
   <!-- queryResultCache caches results of searches - ordered lists of   
         document ids (DocList) based on a query, a sort, and the range   
         of documents requested.  -->   
查询结果缓存   
    <queryResultCache   
      class="solr.LRUCache"   
      size="512"   
      initialSize="512"   
      autowarmCount="256"/>   
  
  <!-- documentCache caches Lucene Document objects (the stored fields for each document).   
documentCache缓存Lucene的文档对象（存储领域的每一个文件）   
       Since Lucene internal document ids are transient, this cache will not be autowarmed.  -->   
由于Lucene的内部文档ID标识（文档名称）是短暂的，所以这种缓存不会被自动warmed。   
    <documentCache   
      class="solr.LRUCache"   
      size="512"   
      initialSize="512"   
      autowarmCount="0"/>   
  
    <!-- Example of a generic cache.    
一个通用缓存的列子。   
These caches may be accessed by name   
         through SolrIndexSearcher.getCache().cacheLookup(), and cacheInsert().   
这些缓存可以通过在SolrIndexSearcher.getCache().cacheLookup()和cacheInsert()中利用缓存名称访问得到。   
         The purpose is to enable easy caching of user/application level data.   
这样做的目的就是很方便的缓存用户级或应用程序级的数据。   
         The regenerator argument should be specified as an implementation   
         of solr.search.CacheRegenerator if autowarming is desired.  -->   
这么做的的关键就是应该明确规定实现solr.search.CacheRegenerator接口如果autowarming是比较理想化的设置。   
    <!--   
    <cache name="myUserCache"   
      class="solr.LRUCache"   
      size="4096"   
      initialSize="1024"   
      autowarmCount="1024"   
      regenerator="org.mycompany.mypackage.MyRegenerator"   
      />   
    -->   
  
    <!-- An optimization that attempts to use a filter to satisfy a search.   
一种优化方式就是利用一个过滤器，以满足搜索需求。   
         If the requested sort does not include a score,   
如果请求的不是要求包括得分的类型，filterCache 这种过滤器将检查与过滤器相匹配的结果。如果找到，过滤器将被用来作为文档的来源识别码，并在这个基础上进行排序。

then the filterCache   
         will be checked for a filter matching the query.  If found, the filter   
         will be used as the source of document ids, and then the sort will be   
         applied to that.   
      -->   
    <useFilterForSortedQuery>true</useFilterForSortedQuery>

    <!-- An optimization for use with the queryResultCache.  When a search   
         is requested, a superset of the requested number of document ids   
         are collected.  For example, of a search for a particular query   
         requests matching documents 10 through 19, and queryWindowSize is 50,   
         then documents 0 through 50 will be collected and cached. Any further   
         requests in that range can be satisfied via the cache.   
    -->

一种优化用于queryResultCache，当一个搜索被请求，也会收集一定数量的文档ID做为一个超集。举个例子，一个特定的查询请求匹配的文档是10到19，此时，queryWindowSize是50，这样，文档从0到50都会被收集并缓存。这样，任何更多的在这个范围内的请求都会通过缓存来满足查询。

    <queryResultWindowSize>50</queryResultWindowSize>

  <!-- This entry enables an int hash representation for filters (DocSets)

         when the number of items in the set is less than maxSize. For smaller

         sets, this representation is more memory efficient, more efficient to

         iterate over, and faster to take intersections.

     -->

    <HashDocSet maxSize="3000" loadFactor="0.75"/>

    <!-- boolToFilterOptimizer converts boolean clauses with zero boost

     cached filters if the number of docs selected by the clause exceeds the   
         threshold (represented as a fraction of the total index)

    -->

<boolTofilterOptimizer enabled="true" cacheSize="32" threshold=".05"/>   
  
    <!-- Lazy field loading will attempt to read only parts of documents on disk that are   
         requested.  Enabling should be faster if you aren't retrieving all stored fields.   
    -->   
    <enableLazyFieldLoading>false</enableLazyFieldLoading>

# 安装配置中文分词-IK

## Solr中配置IK分词步骤

1. 下载 IKAnalyzer3.2.5Stable.jar 分词器，并将 IKAnalyzer3.2.5Stable.jar 复制到tomcat的webapps\solr\WEB-INF\lib 下。
2. 修改 schema.xml 配置文件。Solr主目录下，将

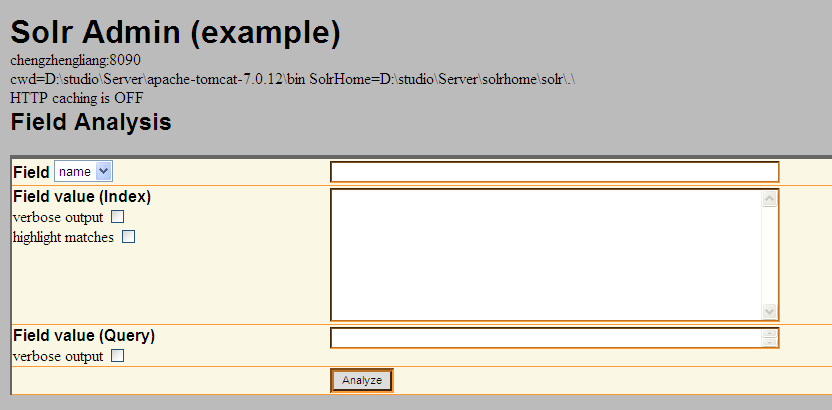
<tokenizer class="solr.WhitespaceTokenizerFactory"/>

修改为<tokenizer class="org.wltea.analyzer.solr.IKTokenizerFactory14"

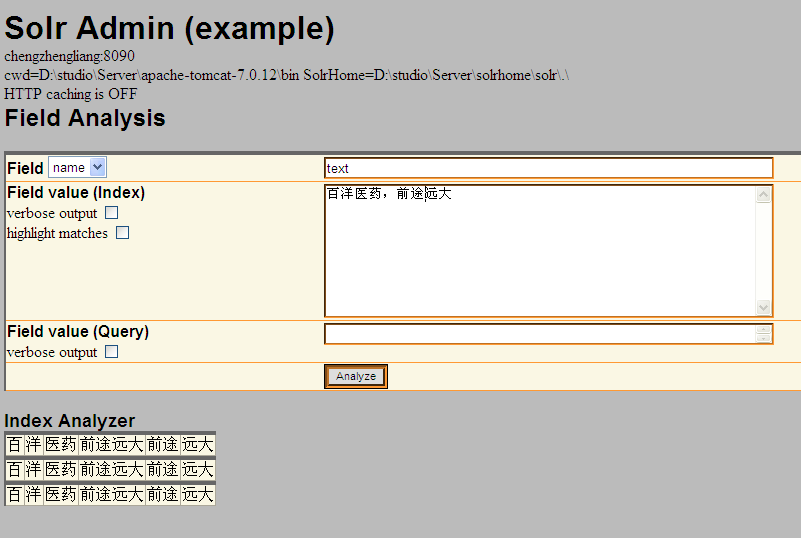
isMaxWordLength="true"/>

**注：要将index、query标签下的solr.WhitespaceTokenizerFactory都替换。**

1. 重启tomcat，打开浏览器进入<http://localhost:8090/solr/admin/analysis.jsp> 页面



1. Name ：text ，文本域处输入：百洋医药，前途远大



1. 可以看到分次效果，这是IK默认的分次效果。将“百洋医药，前途远大。”分解成了

“百、洋、医药、前途、前途远大、远大”。

## IK中停止词的配置

上一篇：“百洋医药，前途远大。”分解成了

“百、洋、医药、前途、前途远大、远大”。“百洋”并没有当做一个词。IK分解词与组成词的设置下面详细设置

1. 在tomcat安装目录webapps\solr\WEB-INF下新建 classes。
2. 在classes下新建3个文件：

IKAnalyzer.cfg.xml：内容为：



配置扩展词文件为：mydict.dic

配置停止词文件为：ext\_stopword.dic

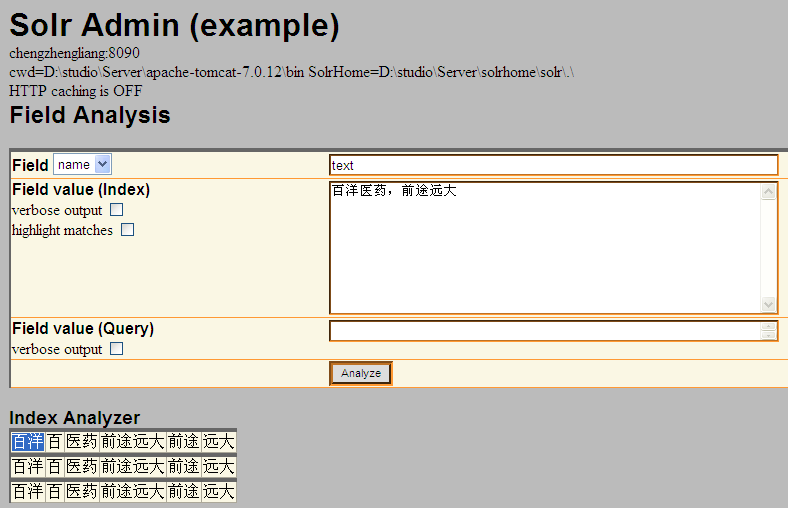
**ext\_stopword.dic：**

停止词配置。

**mydict.dic：**

扩展词配置。

1. 重启tomcat，输入分词，查看效果。
2. 进入 <http://localhost:8090/solr/admin/analysis.jsp>：



1. 看到效果，百洋已经当做一个词语了。
2. 停止词设置：每行代表一个停止词；如设置停止词为：“色鬼”，则搜索时输入“色鬼将不能搜索到数据 会过滤掉该词”。

# 数据源的配置

## 基于Mysql数据库的索引数据源配置。

### 5.1.1 全量索引

1. 下载Mysql数据库驱动包，复制到在tomcat安装目录webapps\solr\lib下。
2. 复制 examples/ apache-solr-3.3.0\contrib\dataimporthandler\lib下的jar包到

webapps\solr\lib 下：



1. 编辑 solrconfig.xml 注册一个请求 uri 为 "/dataimport" 的请求处理器

<requestHandler name="/dataimport"

class="org.apache.solr.handler.dataimport.DataImportHandler">

<lst name="defaults">

<str name="config">data-config.xml</str>

</lst>

</requestHandler>

1. 指定数据源：

<dataSource type="JdbcDataSource" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost/dbname" user="db\_username" password="db\_password"/>

1. data-config.xml 配置样例：

<dataConfig>

<dataSource type="JdbcDataSource"

driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://192.168.3.13:3306/solr-db"

user="root"

password="1234"/>

<document>

<entity name="item" query="select \* from item" >

<field column="id" name="id" />

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

<field column="title" name="title" />

</entity>

</document>

</dataConfig>

1. <http://localhost:8080/solr/dataimport?command=full-import> 这 url 告诉 solr 做全量索引，做索引中会删除所有数据。当然也可以用 clean=false 参数来告诉它不删除，但也会删除相同id的（在 scheam.xml 的uniqueKey 声明的）。<http://localhost:8080/solr/dataimport?command=full-import&clean=false>

说明：如果是多表：下面是多表之间的设置样例。

<dataConfig>

<dataSource driver="org.hsqldb.jdbcDriver" url="jdbc:hsqldb:/temp/example/ex" user="sa" />

<document name="products">

<entity name="item" query="select \* from item">

<field column="ID" name="id" />

<field column="NAME" name="name" />

<field column="MANU" name="manu" />

<field column="WEIGHT" name="weight" />

<field column="PRICE" name="price" />

<field column="POPULARITY" name="popularity" />

<field column="INSTOCK" name="inStock" />

<field column="INCLUDES" name="includes" />

<entity name="feature" query="select description from feature where item\_id='${item.ID}'">

<field name="features" column="description" />

</entity>

<entity name="item\_category" query="select CATEGORY\_ID from item\_category where item\_id='${item.ID}'">

<entity name="category" query="select description from category where id = '${item\_category.CATEGORY\_ID}'">

<field column="description" name="cat" />

</entity>

</entity>

</entity>

</document>

</dataConfig>

### 5.1.2 增量索引

在配置好全量索引的基础上作修改即可。

**步骤：**

1. 数据库表中建一个字段. last\_modified,每次添加或者更新时都要更新该字段。
2. 建立增量索引时要根据该字段和dataimport文件判断最后更新时间，进行比较。
3. 增加查询条件：

<entity name="item" query="select \* from item" pk="id" deltaImportQuery="select \* from item where ID='${dataimporter.delta.id}'"

deltaQuery="select id from item where last\_modified = '${dataimporter.last\_index\_time}'">

<field column="id" name="id" />

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

<field column="title" name="title" />

使用访问地址：执行增量索引。http://127.0.0.1:8090/solr/dataimport?command=delta-import

### 5.1.3 多数据源

1. 如果有多个数据源：具体配置如下：

<dataSource type="JdbcDataSource" name="ds-1" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://db1-host/dbname" user="db\_username" password="db\_password"/>

<dataSource type="JdbcDataSource" name="ds-2" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://db2-host/dbname" user="db\_username" password="db\_password"/>

1. 多个数据源引用时：

..

<entity name="one" dataSource="ds-1" ...>

..

</entity>

<entity name="two" dataSource="ds-2" ...>

..

</entity>

..

## 基于程序动态添加索引的配置。

1. 基本添加：

SolrInputDocument doc1=new SolrInputDocument();

doc1.addField( "id", "1", 1.0f );

doc1.addField( "name", "玛瑙宝石真漂亮", 1.0f );

doc1.addField( "price","10", 10 );

SolrInputDocument doc2 = new SolrInputDocument();

doc2.addField( "id", "2", 1.0f );

doc2.addField( "name", "MM也很漂亮，天地真爱可见", 1.0f );

doc2.addField( "price", "20",20 );

SolrInputDocument doc3 = new SolrInputDocument();

doc3.addField( "id", "3", 1.0f );

doc3.addField( "name", "大家都好漂亮", 1.0f );

doc3.addField( "price", "1000",1000 );

Collection<SolrInputDocument> docs = new ArrayList<SolrInputDocument>();

docs.add( doc1 );

docs.add( doc2 );

docs.add( doc3 );

server.add(docs);

server.commit();

1. 基于对象：

Item item=new Item();

item.id = "one";

item.name = "aaa";

Item itema=new Item();

itema.id = "two";

itema.name = "bbb";

Item itemb=new Item();

itemb.id = "three";

itemb.name = "ccc";

List<Item> items=new ArrayList<Item>();

items.add(item);

items.add(itema);

items.add(itemb);

server.addBeans(items);

server.commit();

# Solr的应用分析说明。

## 图 3. 系统架构6.1 搜索引擎的规划设计

### 6.1.1 定义业务模型

定义搜索的业务模型即对搜索的需求进行分析，确定搜索的业务对象和结构：

1）确定要搜索的业务对象：

分析确定有哪些内容需要进行搜索，这些内容的来源，更新的频次等信息，按如下结构进行定义：

**示例：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编码** | **业务对象** | **优先级** | **内容来源** | **更新频次** | **说明** |
| T001 | 天气信息 | 1 | 168平台 | 1小时 | 输入城市名则结果为该城市的天气，没有输入地名则结果为您所属地市的天气信息 |
| T002 | 餐饮酒店 | 1 | 互联网 | 每日 | 搜索词可以是地名、特色菜、餐馆名称，搜索结果是与您的搜索词相关的餐馆信息 |
| T003 | 航班信息 | 1 | 互联网 | 每日 | 搜索词可以是起始和到港的城市名，搜索结果是与您的搜索词相关的航班信息 |
| T004 | 火车时刻 | 1 | 互联网 | 每日 | 搜索词可以是列车车次、起始站点、途经站点，搜索结果是与您的搜索词相关的火车车次详细信息 |
| T005 | 电影影讯 | 1 | 12580 | 每日 | 搜索词可以是最新放映的影片名或影院名称，搜索结果是与您的搜索词相关的各影院近期放映的电影影讯。 |
| T006 | 股票行情 | 1 | 168平台 | 1分钟 | 搜索词可以是股票代码、股票名称，搜索结果是与您的搜索词相关的即时的股票行情信息。 |
| T007 | 移动业务 | 1 | 手工导入 | 每月 | 搜索词可以移动是业务名称，搜索结果是与您的搜索词相关的业务简介和开通方法信息。 |
| T008 | 通讯录 | 1 | 手工导入 | 每月 |  |
| T009 | 短信 | 1 | 手工导入 | 每日 |  |
| T010 | 彩信 | 1 | 手工导入 | 每日 |  |
| T011 | WAP网址 | 1 | 手工导入 | 每日 |  |

2）梳理业务对象的结构

针对以上业务对象进行分析梳理出业务对象的结构。以下是通讯录和wap网址的结构定义

a)通讯录

|  |  |
| --- | --- |
| **字段** | **说明** |
| ID | 内容标识 |
| userName | 姓名 |
| userMail | 用户邮箱 |
| mobile | 用户手机号 |
| department | 部门 |
| sex | 性别 |
| birthday | 生日 |
| msn | MSN |
| qq | QQ号 |

b)wap网站

|  |  |
| --- | --- |
| 字段 | 说明 |
| ID | 内容标识 |
| 标题 | 内容标题 |
| URL地址 | Wap网站的URL地址 |
| 标签 | 搜索的依据，多个关键字以"，"分隔。 |
| 摘要 | Wap网站的简要说明 |
| 网站分类 | 搜索、娱乐、SNS、新闻 |

### 6.1.2 定制索引服务

定制索引服务是对索引源进行结构化的索引，换句话说，索引后的结果是结构化的、有意义的信息。定制索引主要涉及如下几个方面：

1. 分类：多核解决方案
2. 需要检索的字段
3. 需要存储的字段
4. 过滤条件
5. 排序
6. 索引的更新频次

### 6.1.3 定制搜索服务

定制搜索是指确定搜索支持的规则

1. 过滤条件
2. 排序规则

## 6.2 搜索引擎配置

### 6.2.1 Solr Schema 设计(如何定制索引的结构?)

索引的结构配置主要是对schema.xml中的Fieldtype、Fields、copyField、dynamicField的配置,schema.xml的配置文件说明请参考“[2.4 配置文件说明](file:///E:\全文检索\solr\开源企业搜索引擎SOLR的应用教程.doc#_配置文件说明)”的“1 schema.xml”，下面我们以搜索通讯录为例进行说明。

|  |
| --- |
| **public String userName = null;// 姓名**  **public String userMail = null;// 用户邮箱**  **public String mobile = null;// 用户手机号**  **public String department = null;//**  **public String sex = null;// 性别**  **public String birthday = null;// 生日**  **public String msn = null;**  **public String qq = null;** |

#### 定义好需要的类型（Fieldtype）

定义好需要的类型（Fieldtype），同时为类型（Fieldtype）配置合适的分词器，下面以mmsg4j中文分词为示例,：

|  |
| --- |
| <types>  ……  <!--mmseg4j field types-->  <fieldType name="textComplex" class="solr.TextField" positionIncrementGap="100" >  <analyzer>  <tokenizer class="com.chenlb.mmseg4j.solr.MMSegTokenizerFactory" mode="complex" dicPath="**/opt/solr-tomcat/solr/dic**"/>  <filter class="solr.LowerCaseFilterFactory"/>  </analyzer>  </fieldType>  <fieldType name="textMaxWord" class="solr.TextField" positionIncrementGap="100" >  <analyzer>  <tokenizer class="com.chenlb.mmseg4j.solr.MMSegTokenizerFactory" mode="max-word" dicPath="/opt/solr-tomcat/solr/dic"/>  <filter class="solr.LowerCaseFilterFactory"/>  </analyzer>  </fieldType>    <fieldType name="textSimple" class="solr.TextField" positionIncrementGap="100" >  <analyzer>  <tokenizer class="com.chenlb.mmseg4j.solr.MMSegTokenizerFactory" mode="simple" dicPath="/opt/solr-tomcat/solr/dic"/>  <filter class="solr.LowerCaseFilterFactory"/>  </analyzer>  </fieldType>  ……  </types> |

注：dicPath="/opt/solr-tomcat/solr/dic "是你的词库路径。

#### 定义好需要的字段（Fielde）

|  |
| --- |
| <fields>  ……  <field name=" **username** " type="string" indexed="true" stored="true"/>  <field name=" **usermail** " type="text" indexed="true" stored="true"/>  <field name=" **mobile** " type="text" indexed="true" stored="true"/>  <field name=" **department**" type="text" indexed="true" stored="true"/>  <field name=" **sex**" type="text" indexed="true" stored="true"/>  <field name=" **birthday**" type="text" indexed="true" stored="true"/>  <field name=" **msn**" type="text" indexed="true" stored="true"/>  <field name=" **qq**" type="text" indexed="true" stored="true"/>  ……  </fields> |

|  |
| --- |
| <copyField source="simple" dest="text"/>  <copyField source="complex" dest="text"/> |

**示例： 博客应用程序的声明字段**

|  |
| --- |
| <field name="keywords" type="text\_ws" indexed="true" stored="true"  multiValued="true" omitNorms="true"/>  <field name="creationDate" type="date" indexed="true" stored="true"/>  <field name="rating" type="sint" indexed="true" stored="true"/>  <field name="published" type="boolean" indexed="true" stored="true"/>  <field name="content" type="text" indexed="true" stored="true" />  <!-- catchall field,  containing many of the other searchable text fields  (implemented via copyField further on in this schema) -->  <field name="all" type="text" indexed="true" stored="true" multiValued="true"/> |

# Solrj的基本使用。

## 获得CommonsHttpSolrServer。

CommonsHttpSolrServer chslorServer=new CommonsHttpSolrServer(url);

chslorServer.setSoTimeout(1000); // socket read timeout

chslorServer.setConnectionTimeout(100);

chslorServer.setDefaultMaxConnectionsPerHost(100);

chslorServer.setMaxTotalConnections(100);

chslorServer.setFollowRedirects(false);

chslorServer.setAllowCompression(true);

chslorServer.setMaxRetries(1);

chslorServer.setParser(new XMLResponseParser());

## 建立索引。

两种方式同 5.2

## 执行查询。

SolrQuery query = **new** SolrQuery();

query.setQuery( "\*:\*" );

query.addFilterQuery("id:[1 TO 5]");//范围查询

## 删除索引。

server.deleteByQuery("\*:\*");//根据条件删除索引

## 设置权重

query.setQuery("name:(王小五 )^4 title:(王小五)");

权重越高的排在查询结果越靠前。

## 设置查询到的条数（分页时用）

query.setStart(0);//设置开始行

query.setRows(200);//设置最大行数

## 设置排序字段

query.addSortField( "id", SolrQuery.ORDER.desc );

## 设置高亮

query.setHighlight(**true**); // 开启高亮组件

query.addHighlightField("name");// 高亮字段

query.addHighlightField("title");// 高亮字段

query.setHighlightSimplePre("<font color='red'>");// 标记

query.setHighlightSimplePost("</font>");

query.setHighlightSnippets(1);//结果分片数，默认为1

query.setHighlightFragsize(1000);//每个分片的最大长度，默认为100

## 读取结果

### 6.9.1 读取结果—基本

QueryResponse rsp = server.query( query );

SolrDocumentList results= rsp.getResults();

### 6.9.2 读取结果—对象

List<Item> tmpLists = rsp.getBeans(Item.class);

### 6.9.3 读取高亮结果—对象

Map<String,Map<String,List<String>>>hightlight=rsp.getHighlighting();

# 附录---Solr的实际应用测试报告

## 8.1 线下压力测试报告

1）测试环境：

Tomcat启动内存:1024M 支持并发线程数：512

服务器配置：cpu：P4 3.0 双核，内存：1G DDR333

2）测试结果：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 查询字段数 | 存储数据量 | 日查询次数 | 平均每秒查询次数 | 查询效率 | 全量更新索引数据 |
| 73个 | 3W | 2－2.5W次 | 3 | 3条/秒 | 19秒 |

## 8.2 线上环境运行报告

1）测试环境：

Tomcat启动内存：512M 支持并发线程数：512

服务器配置：cpu 双Intel XEON 3.4GB 内存2G

2）测试结果：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 查询字段数 | 存储数据量 | 并发数 | 一次查询效率 | 多条件一次查询效率 | 多条件二次查询效率 | 全量更新数据 |
| 73个 | 4W | 50 | 620条/秒 | 400条/秒 | 300条/秒 | 40秒 |
| 73个 | 2W | 50 | 810条/秒 | 450条/秒 | 315条/秒 | 20秒 |
| 73个 | 1W | 50 | 940条/秒 | 460条/秒 | 340条/秒 | 13秒 |

**注：**1）一次查询：只带一个条件的查询

2）多条件一次查询：多个条件的查询

3）多条件二次查询效率：多个条件查询（并加了fq - （filter query）过虑查询）

# 9. 附录---[solr性能调优](http://xiaoyezi1128.javaeye.com/blog/560685)

## 9.1 Schema Design Considerations

### indexed fields

    indexed fields 的数量将会影响以下的一些性能：

* 索引时的时候的内存使用量
* 索引段的合并时间
* 优化时间
* 索引的大小

     我们可以通过 将 omitNorms=“true” 来减少indexed fields数量增加所带来的影响。

### stored fields

      Retrieving the stored fields  确实是一种开销。这个开销，受每个文档所存储的字节影响很大。每个文档的所占用的空间越大，文档就显的更稀疏，这样从硬盘中读取数据，就需要更多的i/o操作（通常，我们在存储比较大的域的时候，就会考虑这样的事情，比如存储一篇文章的文档。）

       可以考虑将比较大的域放到solr外面来存储。如果你觉得这样做会有些别扭的话，可以考虑使用压缩的域，但是这样会加重cpu在存储和读取域的时候的负担。不过这样却是可以较少i/0的负担。

       如果，你并不是总是使用 stored fields 的话，可以使用stored field的延迟加载，这样可以节省很多的性能，尤其是使用compressed field 的时候。

## 9.2 Configuration Considerations

### mergeFactor

         这个是合并因子，这个参数大概决定了segment(索引段)的数量。

         合并因子这个值告诉lucene，在什么时候，要将几个segment合并成为一个segment, 合并因子就像是一个数字系统的基数一样。

         比如说，如果你将合并因子设成10，那么每往索引中添加1000个文档的时候，就会创建一个新的索引段。当第10个大小为1000的索引段添加进来的时候，这十个索引段就会被合并成一个大小为10，000的索引段。当十个大小为10，000的索引段生成的时候，它们就会被合并成一个大小为100，000的索引段。如此类推下去。

         这个值可以在 solrconfig.xml 中的 \*mainIndex\*中设置。（不用管indexDefaults中设置）

### mergeFactor Tradeoffs

         较高的合并因子

* 会提高索引速度
* 较低频率的合并，会导致 更多的索引文件，这会降低索引的搜索效率

          较低的合并因子

* 较少数量的索引文件，能加快索引的搜索速度。
* 较高频率的合并，会降低索引的速度。

## 9.3 Cache autoWarm Count Considerations

      当一个新的 searcher 打开的时候，它缓存可以被预热，或者说使用从旧的searcher的缓存的数据来“自动加热”。autowarmCount是这样的一个参数，它表示从旧缓存中拷贝到新缓存中的对象数量。autowarmCount这个参数将会影响“自动预热”的时间。有些时候，我们需要一些折中的考虑，seacher启动的时间和缓存加热的程度。当然啦，缓存加热的程度越好，使用的时间就会越长，但往往，我们并不希望过长的seacher启动时间。这个autowarm 参数可以在solrconfig.xml文件中被设置。

       详细的配置可以参考solr的wiki。

## 9.4 Cache hit rate（缓存命中率）

       我们可以通过solr的admin界面来查看缓存的状态信息。提高solr缓存的大小往往是提高性能的捷径。当你使用面搜索的时候，你或许可以注意一下filterCache,这个是由solr实现的缓存。

       详细的内容可以参考 solrCaching这篇wiki。

## 9.5 Explicit Warming of Sort Fields

       如果你有许多域是基于排序的，那么你可以在"newSearcher"和"firstSearcher"event listeners中添加一些明显需要预热的查询，这样FieldCache 就会缓存这部分内容。

## 9.6  Optimization Considerations

        优化索引，是我们经常会做的事情，比如，当我们建立好索引，然后这个索引不会再变更的情况，我们就会做一次优化了。

        但，如果你的索引经常会改变，那么你就需要好好的考虑下面的因素的。

* 当越来越多的索引段被加进索引，查询的性能就会降低， lucene对索引段的数量有一个上限的限制，当超过这个限制的时候，索引段可以自动合并成为一个。
* 在同样没有缓存的情况下，一个没有经过优化的索引的性能会比经过优化的索引的性能少10%……
* 自动加热的时间将会变长，因为它依赖于搜索。
* 优化将会对索引的分发产生影响。
* 在优化期间，文件的大小将会是索引的两倍，不过最终将会回到它原来的大小，或者会更小一点。

      优化，会将所有的索引段合并成为一个索引段，所以，优化这个操作其实可以帮助避免“too many files”这个问题，这个错误是由文件系统抛出的。

## 9.7 Updates and Commit Frequency Tradeoffs

         如果从机太经常从主机更新的话，从机的性能是会受到影响的。为了避免，由于这个问题而引起的性能下降，我们还必须了解从机是怎样执行更新的，这样我们才能更准确去调节一些相关的参数（commit的频率，spappullers,autowarming/autocount）,这样，从机的更新才不会太频繁。

1. 执行commit操作会让solr新生成一个snapshot。如果将postCommit参数设成true的话，optimization也会执行snapShot.
2. slave上的Snappuller程序一般是在crontab上面执行的，它会去master询问，有没有新版的snapshot。一旦发现新的版本，slave就会把它下载下来，然后snapinstall.
3. 每次当一个新的searcher被open的时候，会有一个缓存预热的过程，预热之后，新的索引才会交付使用。

      这里讨论三个有关的参数：

* **number/frequency of snapshots**  ----snapshot的频率。
* **snappullers 是**  在crontab中的，它当然可以每秒一次、每天一次、或者其他的时间间隔一次运行。它运行的时候，只会下载slave上没有的，并且最新的版本。
* **Cache autowarming** 可以在solrconfig.xml文件中配置。

           如果，你想要的效果是频繁的更新slave上的索引，以便这样看起来比较像“实时索引”。那么，你就需要让snapshot尽可能频繁的运行，然后也让snappuller频繁的运行。这样，我们或许可以每5分钟更新一次，并且还能取得不错的性能，当然啦，cach的命中率是很重要的，恩，缓存的加热时间也将会影响到更新的频繁度。

       cache对性能是很重要的。一方面，新的缓存必须拥有足够的缓存量，这样接下来的的查询才能够从缓存中受益。另一方面，缓存的预热将可能占用很长一段时间，尤其是，它其实是只使用一个线程，和一个cpu在工作。snapinstaller太频繁的话，solr slave将会处于一个不太理想的状态，可能它还在预热一个新的缓存，然而一个更新的searcher被opern了。

         怎么解决这样的一个问题呢，我们可能会取消第一个seacher，然后去处理一个更新seacher，也即是第二个。然而有可能第二个seacher 还没有被使用上的时候，第三个又过来了。看吧，一个恶性的循环，不是。当然也有可能，我们刚刚预热好的时候就开始新一轮的缓存预热，其实，这样缓存的作用压根就没有能体现出来。出现这种情况的时候，降低snapshot的频率才是硬道理。

## 9.8 Query Response Compression

        在有些情况下，我们可以考虑将solr xml response 压缩后才输出。如果response非常大，就会触及NIc i/o限制。

        当然压缩这个操作将会增加cpu的负担，其实，solr一个典型的依赖于cpu处理速度的服务，增加这个压缩的操作，将无疑会降低查询性能。但是，压缩后的数据将会是压缩前的数据的6分之一 的大小。然而solr的查询性能也会有15%左右的消耗。

       至于怎样配置这个功能，要看你使用的什么服务器而定，可以查阅相关的文档。

## 9.9 Embedded vs HTTP Post

         使用embeded 来建立索引，将会比使用xml格式来建立索引快50%。

## 9.10  RAM Usage Considerations（内存方面的考虑）

### OutOfMemoryErrors

           如果你的solr实例没有被指定足够多的内存的话，java virtual machine也许会抛outof memoryError，这个并不对索引数据产生影响。但是这个时候，任何的 adds/deletes/commits操作都是不能够成功的。

### Memory allocated to the Java VM

            最简单的解决这个方法就是，当然前提是java virtual machine 还没有使用掉你全部的内存，增加运行solr的java虚拟机的内存。

#### Factors affecting memory usage（影响内存使用量的因素）

             我想，你或许也会考虑怎样去减少solr的内存使用量。

              其中的一个因素就是input document的大小。

              当我们使用xml执行add操作的时候，就会有两个限制。

* document中的field都是会被存进内存的，field有个属性叫maxFieldLength，它或许能帮上忙。
* 每增加一个域，也是会增加内存的使用的。

引用自：<http://mxsfengg.javaeye.com/blog/355007>