目录

1 概述

1.1 企业搜索引擎方案选型

由于搜索引擎功能在门户社区中对提高用户体验有着重在门户社区中涉及大量需要搜索引擎的功能需求，目前在实现搜索引擎的方案上有集中方案可供选择：

1) 基于 **Lucene** 自己进行封装实现站内搜索。工作量及扩展性都较大，不采用。

2) 调用 **Google、Baidu 的 API** 实现站内搜索。同第三方搜索引擎绑定太死，无法满足后期业务扩展需要，暂时不采用。

3) 基于 **Compass+Lucene** 实现站内搜索。适合于对数据库驱动的应用数据进行索引，尤其是替代传统的 like‘%expression%’来实现对 varchar 或 clob 等字段的索引，对于实现站内搜索是一种值得采纳的方案。但在分布式处理、接口封装上尚需要自己进行一定程度的封装。

4) 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。

基于以述的几种方案的综合分析，对于我们公司的搜索引擎方案，采用 solr 来实现比较合适。

1.2 Solr 的特性

ApacheSolr 是一个开源的搜索服务器，Solr 使用 Java 语言开发，主要基于 HTTP 和 Apache Lucene 实现。定制 Solr 索引的实现方法很简单，用 POST 方法向 Solr 服务器发送一个描述所有 Field 及其内容的 XML 文档就可以了。定制搜索的时候只需要发送 HTTPGET 请求即可，然后对 Solr 返回的信息进行重新布局，以产生利于用户理解的页面内容布局。Solr1.3 版本开始支持从数据库（通过 JDBC）、RSS 提要、Web 页面和文件中导入数据，但是不直接支持从二进制文件格式中提取内容，比如 MSOffice、AdobePDF 或其他专有格式。

更重要的是，Solr 创建的索引与 Lucene 搜索引擎库完全兼容。通过对 Solr 进行适当的配置，某些情况下可能需要进行编码，Solr 可以阅读和使用构建到其他 Lucene 应用程序中的索引。此外，很多 Lucene 工具（如 Nutch、Luke）也可以使用 Solr 创建的索引

Solr 的特性包括：

ü 高级的全文搜索功能

ü 专为高通量的网络流量进行的优化

ü 基于开放接口（XML 和 HTTP）的标准

ü 综合的 HTML 管理界面

ü 可伸缩性－能够有效地复制到另外一个 Solr 搜索服务器

ü 使用 XML 配置达到灵活性和适配性

ü 可扩展的插件体系

1.2.1 Solr 使用 Lucene 并且进行了扩展

ü 一个真正的拥有动态域(DynamicField)和唯一键(UniqueKey)的数据模式(DataSchema)

ü 对 Lucene 查询语言的强大扩展！

ü 支持对结果进行动态的分组和过滤

ü 高级的，可配置的文本分析

ü 高度可配置和可扩展的缓存机制

ü 性能优化

ü 支持通过 XML 进行外部配置

ü 拥有一个管理界面

ü 可监控的日志

ü 支持高速增量式更新(Fastincremental Updates)和快照发布(SnapshotDistribution)

1.2.2 Schema（模式）

ü 定义域类型和文档的域

ü 能够驱动智能处理

ü 声明式的 Lucene 分析器规范

ü 动态域能够随时增加域

ü 拷贝域功能允许对一个域进行多种方式的索引，或者将多个域联合成一个可搜索的域

ü 显式类型能够减少对域类型的猜测

ü 能够使用外部的基于文件的终止词列表，同义词列表和保护词列表的配置

1.2.3 查询

ü 拥有可配置响应格式（XML/XSLT,JSON,Python,Ruby）的 HTTP 接口

ü 高亮的上下文搜索结果

ü 基于域值和显式查询的片段式搜索（FacetedSearch）

ü 对查询语言增加了排序规范

ü 常量的打分范围(Constantscoring range)和前缀式查询－没有 idf,coord,或者 lengthNorm 因子，对查询匹配的词没有数量限制

ü 函数查询(FunctionQuery)-通过关于一个域的数值或顺序的函数对打分进行影响

ü 性能优化

1.2.4 核心

ü 可插拔的查询句柄（QueryHandler）和可扩展的 XML 数据格式

ü 使用唯一键的域能够增强文档唯一性

ü 能够高效地进行批量更新和删除

ü 用户可配置的文档索引变化触发器（命令）

ü 并发控制的搜索器

ü 能够正确处理数字类型，从而能够进行排序和范围搜索

ü 能够控制缺失排序域的文档

ü 支持搜索结果的动态分组

1.2.5 缓存

ü 可配置的查询结果，过滤器，和文档缓存实例

ü 可插拔的缓存实现

ü 后台缓存热启：当一个新的搜索器被打开时，可配置的搜索将它热启，避免第一个结果慢下来，当热启时，当前搜索器处理目前的请求（？？？）。

ü 后台自动热启：当前搜索器缓存中最常访问的项目在新的搜索器中再次生成，能够在索引器和搜索器变化的时候高速缓存常查询的结果

ü 快速和小的过滤器实现

ü 支持自动热启的用户级别的缓存

1.2.6 复制

ü 能够将使用 rsync 传输时改变的索引部分有效的发布

ü 使用拉策略(PullStrategy)来简化增加搜索器

ü 可配置的发布间隔能够允许对时间线和缓存使用进行权衡选择

1.2.7 管理接口

ü 能够对缓存使用，更新和查询进行综合统计

ü 文本分析调试器，能够显示每个分析器每个阶段的结果

ü 基于 WEB 的查询和调试输出：解析查询输出，Lucene 的 explain 方法细节，能够解释为何某个文档打分低，被排除在结果中等等

1.3 Solr 服务原理

Solr 对外提供标准的 http 接口来实现对数据的索引的增加、删除、修改、查询。在 Solr 中，用户通过向部署在 servlet 容器中的 Solr Web 应用程序发送 HTTP 请求来启动索引和搜索。Solr 接受请求，确定要使用的适当 SolrRequestHandler，然后处理请求。通过 HTTP 以同样的方式返回响应。默认配置返回 Solr 的标准 XML 响应，也可以配置 Solr 的备用响应格式。

1.3.1 索引

可以向 Solr 索引 servlet 传递四个不同的索引请求：

1) add/update 允许向 Solr 添加文档或更新文档。直到提交后才能搜索到这些添加和更新。

2) commit 告诉 Solr，应该使上次提交以来所做的所有更改都可以搜索到。

3) optimize 重构 Lucene 的文件以改进搜索性能。索引完成后执行一下优化通常比较好。如果更新比较频繁，则应该在使用率较低的时候安排优化。一个索引无需优化也可以正常地运行。优化是一个耗时较多的过程。

4) delete 可以通过 id 或查询来指定。按 id 删除将删除具有指定 id 的文档；按查询删除将删除查询返回的所有文档。

要实现添加文档索引则只需要调用搜索接口以 HTTP POST 的方式提交 XML 报文。(索引接口：http://192.168.10.85:18080/solr/update)，下面是添加文档的示例报文：

注：多核心时为这个地址 http://192.168.10.85:18080/solr/core0/update/

```
<add>
 <doc>
  <field name="id">TWINX2048-3200PRO</field>
  <field name="name">CORSAIR  XMS 2GB (2 x 1GB) 184-Pin DDR SDRAM Unbuffered DDR 400 (PC 3200) Dual Channel Kit System Memory - Retail</field>
  <field name="manu">Corsair Microsystems Inc.</field>
  <field name="cat">electronics</field>
  <field name="cat">memory</field>
  <field name="features">CAS latency 2, 2-3-3-6 timing, 2.75v, unbuffered, heat-spreader</field>
  <field name="price">185</field>
  <field name="popularity">5</field>
  <field name="inStock">true</field>
 </doc>
 <doc>
  <field name="id">VS1GB400C3</field>
  <field name="name">CORSAIR ValueSelect 1GB 184-Pin DDR SDRAM Unbuffered DDR 400 (PC 3200) System Memory - Retail</field>
  <field name="manu">Corsair Microsystems Inc.</field>
  <field name="cat">electronics</field>
  <field name="cat">memory</field>
  <field name="price">74.99</field>
```

```
  <field name="popularity">7</field>
  <field name="inStock">true</field>
 </doc>
</add>
```

1.3.2 搜索

要实现搜索则只需要调用搜索接口发送 HTTP GET，示例：

```
http://192.168.10.85:18080/solr/select?indent=on&version=2.2&q=
solr&start=0&rows=10&fl=*%2Cscore&qt=standard&wt=standard
```

注：多核心时为这个地址 http://192.168.10.85:18080/solr/core0/select/

示例中，查询 词“ipad”的请求被提交，要求返回 10 个结果。想知道更多有关各种可选查询选项的信息，请参看下文的“搜索语法”部分。

返回的搜索结果报文：

```
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">6</int>
    <lst name="params">
      <str name="rows">10</str>
      <str name="start">0</str>
```

```
        <str name="fl">*,score</str>
        <str name="hl">true</str>
        <str name="q">content:"faceted browsing"</str>
      </lst>
    </lst>

    <result name="response" numFound="1" start="0" maxScore="1.058217">
      <doc>
        <float name="score">1.058217</float>
        <arr name="all">
          <str>http://localhost/myBlog/solr-rocks-again.html</str>
          <str>Solr is Great</str>
          <str>solr,lucene,enterprise,search,greatness</str>
          <str>Solr has some really great features, like faceted browsing and replication
</str>
        </arr>
        <arr name="content">
          <str>Solr has some really great features, like faceted browsing and replication
</str>
        </arr>
        <date name="creationDate">2007-01-07T05:04:00.000Z</date>
        <arr name="keywords">
```

```
            <str>solr,lucene,enterprise,search,greatness</str>
        </arr>
        <int name="rating">8</int>
        <str name="title">Solr is Great</str>
        <str name="url">http://localhost/myBlog/solr-rocks-again.html</str>
    </doc>
</result>

<lst name="highlighting">
    <lst name="http://localhost/myBlog/solr-rocks-again.html">
        <arr name="content">
            <str>Solr has some really great features, like <em>faceted</em>
            <em>browsing</em> and replication</str>
        </arr>
    </lst>
</lst>
</response>
```

1.4 源码结构

1.4.1 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”

- **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。

5) 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。

6) 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。

7) 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。

l **src/java**：该目录存放的是 Solr 使用 Java 编写的源代码。

l **src/scripts**：该目录存放的是配置 Solr 服务器的 Unix BashShell 脚本，在后面介绍多服务器配置中将会有重要的作用。

l **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。

l **src/test**：该目录存放的是测试程序的源代码和测试文件。

l **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

1.4.2 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。

l **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。

l **conf/solrconfig.xml**：该文件是 Solr 的主配置文件。

l **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。

3) data：包含了 Lucene 的二进制索引文件。

4) 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。

Solr 是如何找到运行所需要的 home 目录的呢？

Solr 首先检查名为 solr.solr.home 的 Java 系统属性，有几种不同的方式来设置该 Java 系统属性。一种不管你使用什么样的 Java 应用服务器或 Servlet 引擎都通用的方法是在调用 Java 的命令行中进行设置。所以，你可以在启动 Jetty 的时候显式地指定 Solr 的 home 目录 java -Dsolr.solr.home=solr/ -jar start.jar。另一种通用的方法是使用 JNDI，将 home 目录绑定到 java:comp/env/solr/home。并向 src/webapp/web/WEB-INF/web.xml 添加以下一段代码：

```
1    <env-entry>
2     <env-entry-name>solr/home</env-entry-name>
3     <env-entry-value>solr/</env-entry-value>
4     <env-entry-type>java.lang.String</env-entry-type>
5    </env-entry>
```

实际上这段 XML 在 web.xml 文件中已经存在，你只需要把原来注释掉的 xml 取消注释，添加你所要指向的 home 目录即可。因为修改了 web.xml 文件，所以你需要运行 antdist-war 来重新打包之后再部署 WAR 文件。

最后，如果 Solr 的 home 目录既没有通过 Java 系统属性指定也没有通过 JNDI 指定，那么他将默认指向 solr/。

在产品环境中，我们必须设置 Solr 的 home 目录而不是让其默认指向 solr/。而且应该使用绝对路径，而不是相对路径，因为你有可能从不同的目录下面启动应用服务器。

注：Jetty 是一个开源的 servlet 容器，它为基于 Java 的 web 内容，例如 JSP 和 servlet 提供运行环境。Jetty 是使用 Java 语言编写的，它的 API 以一组 JAR 包的形式发布。开发人员可以将 Jetty 容器实例化成一个对象，可以迅速为一些独立运行（stand-alone）的 Java 应用提供网络和 web 连接。

1.4.3 solr 的各包的说明

分析 Apache Solr 的各个包，力图详细地分析 Solr 的设计和架构。Apache Solr 由 12 个包组成，如下：

1. org.apache.solr.analysis
2. org.apache.solr.core
3. org.apache.solr.request
4. org.apache.solr.schema
5. org.apache.solr.search
6. org.apache.solr.search.function
7. org.apache.solr.servlet
8. org.apache.solr.tst
9. org.apache.solr.update
10. org.apache.solr.util
11. org.apache.solr.util.test
12. org.apache.solr.util.xlst

我们先从使用者的角度出发，最先看到的当然是 servlet,因为 Solr 本身是个独立的网络应用程序，需要在 Servlet 容器中运行来提供服务，所以 servlet 是用户接触的最外层。我们看看 org.apache.solr.servlet 包。这个包很简单，只有两个类：SolrServlet 和 SolrUpdateServlet. 我们很容易从类名中猜出这两个类的用途。

SolrServlet 类继承 HttpServlet 类，只有四个方法：

· init()

· destroy()

· doGet()

· doPost()

SolrServlet 类中除了普通的 Java 类对象（包括 Servlet 相关的）外，有四个 Solr 本身的类，还有一个 Solr 本身的异常。其中两个类和一个异常属于 org.apache.solr.core 包，两个类属于 org.apache.solr.request 包。属于 core 包的有：

· Config:

· SolrCore:

属于 request 包的有：

· SolrQueryResponse:

· QueryResponseWriter:

分析一下这个 SolrServlet 类。首先 servlet 会调用 init()方法进行初始化：通过 Context 查找 java:comp/env/solr/home 来确定 Solr 的主目录（home），接着调用 Config.setInstanceDir(home)方法设置这个实例的目录。然后通过 SolrCore.getSolrCore()来获得一个 SolrCore 实例。destroy()方法将会在 Servlet 对象销毁时调用，仅仅调用 core.close()关闭 SolrCore 实例。

当用户请求进来时 doPost()简单地将任务交给 doGet()完成，主要的任务由 doGet()完成。分析一下 doGet()方法：

1) 使用 SolrCore 和 doGet()参数 request 生成一个 SolrServletRequest 对象(注意：这个 SolrServletRequest 类不是公开类，位于 org.apache.solr.servlet 包中，继承了 SolrQueryRequestBase 类，仅仅接受 SolrCore 和 HttpServletRequest 对象作为参数)

2) 然后 SolrCore 执行 execute()方法（参数为 SolrServletRequest 和 SolrQueryResponse）

由此可见，真正的处理核心是 SolrCore 的 execute 方法

1.5 版本说明

1.5.1 1.3 版本

Solr 的 1.3 版本以 Apache Lucene 2.3 版本的巨大性能提升为基础，并增加了一个新的、向后兼容的、即插即用组件架构。该架构使开发人员踊跃创建可以进一步增强 Solr 的组件。例如，1.3 版本就包含能够实现以下功能的组件：

- “您是不是要找……”拼写检查
- 查找“类似的”Document
- 根据编辑输入（又称付费排序）覆盖搜索结果

另外，查询解析、搜索、分类以及调试这样的现有功能也被组件化了。现在，您可以通过组合这些组件来自定义创建 SolrRequestHandler。最后，Solr 还增加了直接为数据库内容创建索引的功能，并且通过分布式搜索来支持庞大的系统，这一点对很多企业都很重要。

Solr 1.3 发生了很大的变化。它有很多新的特性，比如拼写检查、数据导入、编辑排序和分布式搜索。此外，还学习了 Solr 的增强功能，包括一个更新、更快的 Lucene 版本。Solr 有许多地方改变了，也有许多地方没有改变。Solr 仍然是一个可靠的、可行的、支持良好的搜索服务器，并且已经可以部署到企业中。现在，Solr 开发人员开始研究添加文档聚合、更多的分析选项、Windows 友好的复制以及复制文档检测等特性。

1.5.2 1.4 版本

Solr 1.4.1 发布了！此版本是一个 bug 修复版，同时也将 Lucene 升级到了 Lucene 2.9.3。作为 Java 开源世界最为著名的全文检索工具来说，Lucene 名气之大是在是可想而知了。而 Solr 是基于 Lucene 的一个企业级全文检索工具实现。

Solr 底层基于 Lucene，而操作完全基于 web 方式 Solr 同时提供检索高亮标记，动态集群，数据库整合，多种文档支持的特性，因此是用来制作企业级全文检索的一个良好的选择。

Solr 完全采用 Java 编写，因此具有跨平台的能力，可以运行在 Linux，Unix 等多种支持 Java 的平台上。而 Solr 既可以作为独立应用部署在应用程序服务器上（例如 Tomcat），也可以通过的 REST 和 JSONapi 来与现有的应用进行整合，因此使用起来十分灵活方便。

1.6 分布式和复制 Solr 架构

图 1

注意，图 1 中输入的请求可以进入任何一个复制的片中，因为它们是功能齐全的 Solr 实例。然后，检索节点会将请求发送到其他片。这些请求仅仅是普通的 Solr 请求。要将请求提交到 Solr 服务器并分发请求，需要将 shards 参数添加到请求，比如：

```
http://localhost:8983/solr/select?shards=localhost:8983/solr,localhost:7574/solr&q=ipod+solr
```

在这个例子中，我假定在本地主机上运行了两个 Solr 服务器（它不是真正的分布式的；它适合于这里的论述，但不能用于您的设置），主服务器在端口 8983 上，从服务器在端口 7574 上。输入的请求进入端口 8983 上的实例，然后它将请求发送到片式服务器上。应用程序很可能将 shards 参数值设置成 solrconfig.xml 文件中的 SolrRequestHandler 的默认配置的一部分，这样就不需要在每次查询时都传入所有片式服务器的名称了。

2 Solr 的安装与配置

2.1 在 Tomcat 下 Solr 安装

下面以 Linux 下安装配置 Solr 进行说明，windows 与此类似。

2.1.1 安装准备

1) 下载 tomcat6.0.20：

http://tomcat.apache.org/

2) tomcat 调优：

调优 tomcat 加大内存和连接数

Ø MaxThread 500

Ø MinSpareThread 25

Ø MaxSpareThread75

Ø Xmx 1024M

3) 操作系统网络参数优化

用做测试的各台服务器，均在/etc/sysctl.conf 配置文件中增加如下内核参数：

net.ipv4.tcp_syncookies = 1

net.ipv4.tcp_tw_reuse = 1

net.ipv4.tcp_tw_recycle = 1

net.ipv4.tcp_fin_timeout = 5

4) 下载 solr

下载地址：http://apache.etoak.com/lucene/solr/1.4.1/

详情请见：http://wiki.apache.org/solr/Solr1.4

5) 下载分词器：

下载地址：http://code.google.com/p/mmseg4j/

6) 下载词库：

http://code.google.com/p/mmseg4j/downloads/detail?name=data.zip&can=2&q=

2.1.2 安装过程

1) 安装 tomcat6

安装完 tomcat 后修改./conf/server.xml

```
<Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443"   URIEncoding="UTF-8"/>
```

注：如果没有设置 URIEncoding="UTF-8"，在提交查询的 select 的 url 会出现乱码，当然也就查不到了。

2) 将下载的 solr 包下面的 dist 文件夹中的 **apache-solr-1.4.1.war** 拷贝到 **tomcat 的 webapps** 并且改名为 solr.war 一般情况下会自动生成相应的文件夹。

3) 新建/opt/solr-tomcat/solr 文件夹，把下载的 solr 包中的 **example/solr** 文件夹下面的所有文件放入到/opt/solr-tomcat/solr 里面。

4) 最后一步配置添加 solr.home 环境变量，可以有二种方式（两种取其一即可）：

a）基于环境变量

linux 在当前用户的环境变量中(.bash_profile)或在./bin/catalina.sh 中添加如下环境变量：

```
export JAVA_OPTS="$JAVA_OPTS -Dsolr.solr.home=/opt/solr-tomcat/solr"
```

b）基于 JNDI

在 tomcat 的 conf 文件夹建立 Catalina 文件夹，然后在 Catalina 文件夹中在建立 localhost 文件夹，在该文件夹下面建立 solr.xml,Xml 代码:

```
<Context docBase="/usr/local/tomcat6/webapps/solr.war" debug="0" crossContext="true
" >

 <Environment name="solr/home" type="java.lang.String" value="/opt/solr-tomcat/solr"
override="true" />

</Context>
```

注：如果没有设定 solr.solr.home 环境变量或 JNDI 的情况下，Solr 查找./solr，因此在启动时候需要切换到/opt/solr-tomcat

2.1.3 验证安装

访问 solr 管理界面 http://ip:port/solr

1) 打开管理后台

打开浏览器，输入：http://192.168.10.85:18080/solr/admin/（注：多核心时为这个地址 http://192.168.10.85:18080/solr/，首页会列出多核的链接，点击进入），就可以访问 solr 服务了

2) 如果出现如下图示，表示配置成功。

注：观察这个指定的 solr 主位置，里面存在两个文件夹：conf 和 data。其中 conf 里存放了对 solr 而言最重要的两个配置文件 schema.xml 和 solrconfig.xml。data 则用于存放索引文件，schema.xml 主要包括 types、fields 和其他的一些缺省设置。solrconfig.xml 用来配置 Solr 的一些系统属性，例如与索引和查询处理有关的一些常见的配置选项，以及缓存、扩展等等。

2.2 中文分词配置

2.2.1 mmseg4j

下面主要说的是怎么在 solr 中加入中文分词(引入实现 mmseg 算法的第三方开源项目)。

1) 下载分词器：http://code.google.com/p/mmseg4j/

2) 下载词库：http://code.google.com/p/mmseg4j/downloads/detail?name=data.zip&can=2&q=

3) 将解压后的 mmseg4j-1.8.2 目录下的 mmseg4j-all-1.8.2.jar 拷贝到 Tomcat_HOME\webapps\solr\WEB-INF\lib 目录下。

4) 添加词库：在/opt/solr-tomcat/solr 目录下新建 dic 文件夹，将解压后的 sogou-dic\data 目录下的 words.dic 拷贝到/opt/solr-tomcat/solr/dic 目录下。

5) 更改 schema.xml(/opt/solr-tomcat/solr/conf/)文件，使分词器起到作用。

在 schema.xml 的<types>、<fields>和部分新增如下配置：

```
<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。

```
<fields>
        ……
<field name="simple" type="textSimple" indexed="true" stored="true"
multiValued="true"/>
<field name="complex" type="textComplex" indexed="true" stored="true"
multiValued="true"/>
<field name="text" type="textMaxWord" indexed="true" stored="true"
multiValued="true"/>
        ……
</fields>
```

```
<copyField source="simple" dest="text"/>
```

```
<copyField source="complex" dest="text"/>
```

重启你的 tomcat。

访问：http://192.168.10.85:18080/solr/admin/analysis.jsp 可以看 mmseg4j 的分词效果。在 Field 的下拉菜单选择 name，然后在应用输入 complex。分词的结果，如下图：

好了，可以运行起来了，那就添加个文档试下，在 解压后的 apache-solr-1.4.0\example\exampledocs 目录下创建 mmseg4j-solr-demo-doc.xml 文档，内容如下：

```
<add>
  <doc>
    <field name="id">1</field>
    <field name="text">昨日,记者从解放军总参谋部相关部门获悉,截至 3 月 28 日,解放军和武警部队累计出动 7.2 万人次官兵支援地方抗旱救灾。组织民兵预备役人员 20.2 万人次支援地方抗旱救灾。</field>
  </doc>
  <doc>
    <field name="id">2</field>
    <field name="text">下半年房价调整就是挤水分 房价回不到去年水平。</field>
  </doc>
  <doc>
    <field name="id">3</field>
    <field name="text">solr 是基于 Lucene Java 搜索库的企业级全文搜索引擎，目前是 apache 的一个项目。</field>
```

```
    </doc>
    <doc>
        <field name="id">4</field>
        <field name="text">中国人民银行是中华人民共和国的中央银行。</field>
    </doc>
</add>
```

然后在 cmd 下运行 post.jar，如下：

```
l    F:\lucene\solr\apache-solr-1.4.0\example\exampledocs>java
-Durl=http://localhost:8089/solr/update -Dcommit=yes -jar post.jar
mmseg4j-solr-demo-doc.xml
```

注意：mmseg4j-solr-demo-doc.xml 要是 UTF-8 格式，不然提交后会乱码。

还有在查询中文时需要把 tomcat 设置成 URIEncoding="UTF-8";

查看是否有数据，访问：http://localhost:8089/solr/admin/在 QueryString: 中输入“中国”，显示如下图所示：

到这里，分词成功。至于 schema.xml 中的配置属性会在下一章中进行详细的介绍。

[Solr 分词顺序]

Solr 建立索引和对关键词进行查询都得对字串进行分词，在向索引库中添加全文检索类型的索引的时候，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。分词的顺序如下：

索引

1：空格 whitespaceTokenize

2：过滤词(停用词，如：on、of、a、an 等) StopFilter

3：拆字 WordDelimiterFilter

4：小写过滤 LowerCaseFilter

5：英文相近词 EnglishPorterFilter

6：去除重复词 RemoveDuplicatesTokenFilter

查询

1：查询相近词

2：过滤词

3：拆字

4：小写过滤

5：英文相近词

6：去除重复词

以上是针对英文，中文的除了空格，其他都类似。

注：

- mmseg4j ： complex 1200kb/s 左右, simple 1900kb/s 左右
- mmseg4j ：自带 sogou 词库，支持名为 wordsxxx.dic（如：data/words-my.dic），utf8 文本格式的用户自定义词库，一行一词。不支持自动检测。 -Dmmseg.dic.path

MMseg4jHandler(1.8 以后支持):添加 MMseg4jHandler 类，可以在 solr 中用 url 的方式来控制加载检测词库。参数：

- dicPath 是指定词库的目录，特性与 MMSegTokenizerFactory 中的 dicPath 一样（相对目录是，是相对 solr.home）。
- check 是指是否检测词库，其值是 true 或 on。

- reload 是否尝试加载词库，其值是 true 或 on。此值为 true，会忽视 check 参数。

solrconfig.xml：

<requesthandlername="/mmseg4j"class="com.chenlb.mmseg4j.solr.MMseg4jHandler">

<lstname="defaults">

<strname="dicPath">

dic

</str>

</lst>

</requesthandler>

此功能可以让外置程序做相关的控制，如：尝试加载词库，然后外置程序决定是否重做索引。

2.2.2 paoding

对全文检索而言，中文分词非常的重要，这里采用了 qieqie 庖丁分词。集成非常的容易，我下载的是 2.0.4-alpha2 版本，其中它支持最多切分和按最大切分。创建自己的一个中文 TokenizerFactory 继承自 solr 的 BaseTokenizerFactory。

```
/**

 * Created by IntelliJ IDEA.

 * User: ronghao

 * Date: 2007-11-3
```

```
 * Time: 14:40:59

 * 中文切词 对庖丁切词的封装

 */

public class ChineseTokenizerFactory extends BaseTokenizerFactory {

   /**

    * 最多切分  默认模式

    */

   public static final String MOST_WORDS_MODE = "most-words";

   /**

    * 按最大切分

    */

   public static final String MAX_WORD_LENGTH_MODE = "max-word-length";

   private String mode = null;

   public void setMode(String mode) {

         if (mode==null||MOST_WORDS_MODE.equalsIgnoreCase(mode)

               || "default".equalsIgnoreCase(mode)) {

            this.mode=MOST_WORDS_MODE;

         } else if (MAX_WORD_LENGTH_MODE.equalsIgnoreCase(mode)) {
```

```
            this.mode=MAX_WORD_LENGTH_MODE;
        }
        else {
            throw new IllegalArgumentException("不合法的分析器 Mode 参数设置:" + mode);
        }
    }
  @Override
  public void init(Map<String, String> args) {
    super.init(args);
    setMode(args.get("mode"));
  }
  public TokenStream create(Reader input) {
    return new PaodingTokenizer(input, PaodingMaker.make(),
            createTokenCollector());
  }
  private TokenCollector createTokenCollector() {
    if( MOST_WORDS_MODE.equals(mode))
```

```
            return new MostWordsTokenCollector();
        if( MAX_WORD_LENGTH_MODE.equals(mode))
            return new MaxWordLengthTokenCollector();
        throw new Error("never happened");
    }
}
```

在 schema.xml 的字段 text 配置里加入该分词器。

```
<fieldtype name="text" class="solr.TextField" positionIncrementGap="100">
        <analyzer type="index">
            <tokenizer class="com.ronghao.fulltextsearch.analyzer.ChineseTokenizerFactory" mode="most-words"/>

            <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
            <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0"/>
            <filter class="solr.LowerCaseFilterFactory"/>

            <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
        </analyzer>
```

```
        <analyzer type="query">
            <tokenizer class="com.ronghao.fulltextsearch.analyzer.ChineseTokenizerFactory" mode="most-words"/>
            <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
            <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
            <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0"/>
            <filter class="solr.LowerCaseFilterFactory"/>
            <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
        </analyzer>
    </fieldtype>
</types>
```

完成后重启 tomcat，即可在 http://localhost:8080/solr/admin/analysis.jsp

体验到庖丁的中文分词。注意要将 paoding-analysis.jar 复制到 solr 的 lib 下，注意修改 jar 包里字典的 home。

2.3 多核（MultiCore）配置

Solr Multicore 是 solr1.3 的新特性。其目的一个 solr 实例，可以有多个搜索应用。

我们知道你既可以把不同类型的数据放到同一 index 中，也可以使用分开的多 indexes。基于这一点，你只需知道如何使用多 indexes（实际上就是运行 Solr 的多实例）。尽管如此，为每一个类型添加一个完整的 Solr 实例会显得太臃肿庞大。Solr1.3 引入了 Solrcore 的概念，该方案使用一个 Solr 实例管理多个 indexes，这样就有热点 core（hotcore）的重读（reloading）与交换（swap，通常是读 index 与写 index 交换），那么管理一个 core 或 index 也容易些。每个 Solrcore 由它自己的配置文件和索引数据组成。在多 core 执行搜索和索引几乎和没有使用 core 一样。你只是添加 core 的名字为各自不同的 URL。单 core 情况下的如下搜索：

http://localhost:8983/solr/select?q=dave%20matthews

在多 core 环境下，你可以通过如下方式访问一个名为 mbartists 的 core：

http://localhost:8983/solr/core0/select?q=dave%20matthews

并非在 URL 中引入 corename 的参数名值对，而是用不同的 context。这样就可以像在单 core 中执行你的管理任务，搜索，更新操作。

2.3.1 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" ?>
    <solrpersistent="false">
      <coresadminPath="/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

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。

```
<corename="core0" instanceDir="core0">
    <property name="dataDir"value="/opt/solr-tomcat/solr/data/core0" />
 </core>
<corename="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/.

2.3.2 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。

2.4 配置文件说明

运行 solr 是个很简单的事，如何让 solr 高效运行你的项目，这个就不容易了。要考虑的因素太多。这里很重要一个就是对 solr 的配置要了解。懂得配置文件每个配置项的含义，这样操作起来就会如鱼得水！

在 solr 里面主要的就是 solr 的主目录下面的 **schema.xml**,**solrConfig.xml**。

2.4.1 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>
```

1. 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

2. 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>
    <fieldname="id" type="integer" indexed="true"stored="true" required="true" />
    <fieldname="name" type="text" indexed="true"stored="true" />
    <fieldname="summary" type="text" indexed="true"stored="true" />
    <fieldname="author" type="string" indexed="true"stored="true" />
    <fieldname="date" type="date" indexed="false"stored="true" />
    <fieldname="content" type="text" indexed="true"stored="false" />
    <fieldname="keywords" type="keyword_text"indexed="true" stored="false" multiValued="true"/>
    <fieldname="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。

3. 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"/>

注：“拷贝字段”就是查询的时候不用再输入：userName:张三 anduserProfile: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。

4. 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 solrwiki：
http://wiki.apache.org/solr/SchemaXml

2.4.2 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。

l **dataDirparameter**：<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。

l **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。

l **maxMergeDocs： 控制可由 SoIr ,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。**

l **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。**

l **unlockOnStartup： unlockOnStartup 告知 SoIr 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>
```

l **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）。

<updateHandlerclass="solr.DirectUpdateHandler2">

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

Settingthis 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 docsif certain criteria are met. Future versions may expand the available

criteria -->

<autoCommit>

<maxDocs>10000</maxDocs> <!-- maximum uncommited docs beforeautocommit 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 addinga 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">

<strname="exe">snapshooter</str>

<strname="dir">solr/bin</str>

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

<!--

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

<arrname="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 ofclauses 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 forfilters (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 cachesresults 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 LuceneDocument 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 Luceneinternal 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 genericcache.
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。
<!--
<cachename="myUserCache"
class="solr.LRUCache"
size="4096"
initialSize="1024"
autowarmCount="1024"
regenerator="org.mycompany.mypackage.MyRegenerator"
/>
-->

<!-- An optimization thatattempts 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 optimizationfor 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 hashrepresentation 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.
     -->
    <HashDocSetmaxSize="3000" loadFactor="0.75"/>

    <!-- boolToFilterOptimizerconverts boolean clauses with zero boost
      cached filters ifthe number of docs selected by the clause exceeds the
        threshold (represented as a fraction of the total index)
    -->
```

<boolTofilterOptimizerenabled="true" cacheSize="32"threshold=".05"/>

<!-- Lazy field loading willattempt to read only parts of documents on disk that are requested. Enabling should be faster if you aren't retrieving all storedfields. -->

<enableLazyFieldLoading>false</enableLazyFieldLoading>

3 Solr 的应用

关于 solr 的详细使用说明，请参考:http://wiki.apache.org/solr/FrontPage

3.1 SOLR 应用概述

3.1.1 Solr 的应用模式

在门户社区中需要使用 solr，可采用如下模式：

1) **对于原有系统已有的数据或需要索引的数据量较大的情况**

直接采用通过 http 方式调用 solr 的接口方式，效率较差，采用 solr 本身对 csv 的支持（http://wiki.apache.org/solr/UpdateCSV），将数据导出为 csv 格式，然后调用 solr 的 csv 接口 http://localhost:8080/solr/update/csv

2) 对于系统新增的数据

先将需要索引查询的数据组装成 xml 格式，然后使用 httpclient 将数据提交到 solr 的 http 接口，例如：http://localhost:8080/solr/update

也可以参考 post.jar 中的 SimplePostTool 的实现。
http://svn.apache.org/viewvc/lucene/solr/trunk/src/java/org/apache/solr/util/SimplePostTool.java?view=co

3) 中文分词

采用 mmseg4j 作为 solr（Lucene）缺省的中文分词方案

项目库：http://code.google.com/p/mmseg4j/

也可以采用庖丁解牛作为 solr（Lucene）缺省的中文分词方案

项目库：http://code.google.com/p/paoding/

4) 与 nutch 的集成使用

http://blog.foofactory.fi/2007/02/online-indexing-integrating-nutch-with.html

5) 嵌入式 Solr

http://wiki.apache.org/solr/Solrj#EmbeddedSolrServer

6) 分布式索引

http://wiki.apache.org/solr/CollectionDistribution

3.1.2 SOLR 的使用过程说明

1) 第一步：搜索引擎规划设计

ü 定制好业务模型

ü 定制好索引结构

ü 定制好搜索策略

ü 配置 solr 的 schemal 配置文件

2) 第二步：搜索引擎配置

根据搜索引擎的规划，配置 solr 的 schema.xml 等配置文件。

3) 第三步：构建索引并定时更新索引

通过调用索引接口进行索引的构建与更新。

4) 第三步：搜索

通过调用搜索接口进行搜索。

3.2 一个简单的例子

3.2.1 Solr Schema 设计

见“2.2.1 mmseg4j”中的第(5)点。

3.2.2 构建索引

找到下载的软件包，在\apache-solr-1.4.1\example\exampledocs 目录下创建 mmseg4j-solr-demo-doc.xml 文档，内容如下：

```
<add>
  <doc>
    <field name="id">1</field>
    <field name="text">昨日,记者从解放军总参谋部相关部门获悉,截至 3 月 28 日,解放军和武警部队累计出动 7.2 万人次官兵支援地方抗旱救灾。组织民兵预备役人员 20.2 万人次支援地方抗旱救灾。</field>
  </doc>
  <doc>
    <field name="id">2</field>
    <field name="text">下半年房价调整就是挤水分 房价回不到去年水平。</field>
  </doc>
  <doc>
    <field name="id">3</field>
    <field name="text">solr 是基于 Lucene Java 搜索库的企业级全文搜索引擎，目前是
```

```
apache 的一个项目。</field>
    </doc>
    <doc>
        <field name="id">4</field>
        <field name="text">中国人民银行是中华人民共和国的中央银行。</field>
    </doc>
</add>
```

然后在 cmd 切换到“apache-solr-1.4.0\example\exampledocs>”,运行 post.jar，命令如下：

```
java -Durl=http://192.168.10.85:18080/solr/update -Dcommit=yes -jar post.jar
mmseg4j-solr-demo-doc.xml
```

注：多核心时为这个地址 http://192.168.10.85:18080/solr/core0/update/

3.2.3 搜索测试

查看是否有数据

访问：http://192.168.10.85:18080/solr/admin/

在 Query String：中输入“中国”，显示如下图所示：

http://192.168.10.85:18080/solr/core0/select/?q=中国&version=2.2&start=0&rows=10&indent=on

3.3 搜索引擎的规划设计

搜索应用架构

3.3.1 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

3.3.2 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

3.3.3 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

3.4 搜索引擎配置

3.4.1 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”的“1schema.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;
```

1. 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。

2. 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"/>
```

3.5 如何进行索引操作？

3.5.1 基本索引操作

在 Solr 中，通过向部署在 servlet 容器中的 Solr Web 应用程序发送 HTTP 请求来启动索引。您可以向 Solr 索引 servlet 传递四个不同的索引请求：

1) **add/update** 允许您向 Solr 添加文档或更新文档。直到提交后才能搜索到这些添加和更新。

2) **commit** 告诉 Solr，应该使上次提交以来所做的所有更改都可以搜索到。

3) **optimize** 重构 Lucene 的文件以改进搜索性能。索引完成后执行一下优化通常比较好。如果更新比较频繁，则应该在使用率较低的时候安排优化。一个索引无需优化也可以正常地运行。优化是一个耗时较多的过程。

4) **delete** 可以通过 id 或查询来指定。按 id 删除将删除具有指定 id 的文档；按查询删除将删除查询返回的所有文档。

1. 新增、更新索引

```
<add>
<doc>
<fieldname="url">http://localhost/myBlog/solr-rocks.html</field>
<field name="title">Solr Search is Simply Great</field>
<field name="keywords">solr,lucene,enterprise,search</field>
```

```
<fieldname="creationDate">2007-01-06T05:04:00.000Z</field>
<field name="rating">10</field>
<field name="content">Solr is a really great open source searchserver. It scales,
it's easy to configure and the Solr community is reallysupportive.</field>
<field name="published">on</field>
</doc>
</add>
```

<doc>中的每个 field 条目告诉 Solr 应该将哪些 Field 添加到所创建文档的 Lucene 索引中。可以向 add 命令添加多个<doc>。通过 HTTP POST 将命令发往：http://localhost:8080/solr/update。如果一切进展顺利，则会随<result status="0"/>返回一个 XML 文档，添加文档成功，如果是相同的 URL 自动更新文档（示例应用程序中的 URL 是 Solr 识别文档以前被添加过所使用的惟一 id）。

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

```
<uniqueKey>url</uniqueKey>
```

2. 删除索引

1） 删除制定 ID 的索引

```
<delete><id>05138022</id></delete>
```

2） 删除查询到的索引数据

```
<delete><query>id:IW-02</query></delete>
```

3） 删除所有索引数据

```
<delete><query>*:*</query></delete>
```

通过 HTTP POST 将命令发往：http://localhost:8080/solr/update

注：以上 xml 文件通过 HTTP POST 将命令发往在 http://192.168.10.85:18080/solr/update/，多核心时为这个地址 http://192.168.10.85:18080/solr/core0/update/

3.5.2 批量索引操作

对原有系统已有的数据或需要索引的数据量较大的情况，需要进行批量的索引操作

1. 通过 CVS 文件的方式提交

直接采用通过 http 方式调用 solr 的接口方式，效率较差，采用 solr 本身对 csv 的支持（http://wiki.apache.org/solr/UpdateCSV），将数据导出为 csv 格式，然后调用 solr 的 csv 接口 http://localhost:8080/solr/update/csv

具体操作步聚：

1.修改 conf/solrconfig.xml

1）新增 csv 处理配置项

```
   <!-- CSV update handler, loaded on demand -->
   <requestHandler name="/update/csv" class="solr.CSVRequestHandler"
startup="lazy">
   </requestHandler>
```

2）修改 enableRemoteStreaming=true

```
<requestParsers enableRemoteStreaming="true"
multipartUploadLimitInKB="2048" />
```

2.下面的命令能直接读取输入文件提交到 Solr

```
curl
http://localhost:8983/solr/update/csv?stream.file=exampledocs/books.c
sv&stream.contentType=text/plain;charset=utf-8
#NOTE: The full path, or a path relative to the CWD of the running solr
server must be used.
```

2. 数据库数据导入生成索引（DataImportHandler DIH）

写程序可以将数据读出 100 条，如果你的内存够大，可以是 1000 条甚至更多，然后放入 Collection 中，批量提交至 solr。或者读取数据写入 xml 文件中，再将该文件提交到 solr 等等。但是，我们还可以通过配置文件直接读取数据库建立索引。

1）　全量更新索引

一、提供对应数据库的 jdbc 驱动。

将 jdbc 驱动放在 TOMCAT_HOME\webapps\solr\WEB-INF\lib 目录下。

二、在 C:\solr-tomcat\solr\conf 目录下新建 db 文件夹，在 db 文件夹中新建 db-data-config.xml 内容如下：

```
<dataConfig>

        <dataSourcetype="JdbcDataSource"driver="oracle.jdbc.driver.OracleDriver"url="jdbc:oracle:thin:@192.168.1.1:1521:数据库名" user="用户名" password="密码"/>

              <document name="messages">

        <entity name="message"transformer="ClobTransformer" query="select * fromtb_message">

            <fieldcolumn="ID" name="id" />

                        <fieldcolumn="TITLE" name="title"/>

            <fieldcolumn="CONTENT" clob="true" name="content" />

            <fieldcolumn="SENDTIME" name="sendtime" />

       </entity>

     </document>

 </dataConfig>
```

三、修改 C:\solr-tomcat\solr\conf 目录下的 solrconfig.xml 文件。在相应的位置添加如下代码：

```
<!-- DataImportHandler -->

 <requestHandler name="/dataimport"
class="org.apache.solr.handler.dataimport.DataImportHandler">

    <lst name="defaults">

      <str name="config">C:\solr-tomcat\solr\conf\db\db-data-config.xml</str>

    </lst>

  </requestHandler>
```

注：C:\solr-tomcat\solr\conf\db\db-data-config.xml 是 db-data-config.xml 的存放路径，你要根据实际情况而定。

document：一个文档也就是 lucene 的 document 这个没什么解释的。

entity：主要针对的是一个数据库表。

filed：属性 column 是数据库的字段，name 是 filed 的名字，即 schema 中的 field name。

更多请参考官方 wiki：http://wiki.apache.org/solr/DataImportHandler

四、启动 TOMCAT，输入地址进行导入，导入分为多种模式：我用的是完全导入模式。

http://localhost:8080/solr/dataimport?command=full-import

结果：

00C:\solr-tomcat\solr\conf\db\db-data-config.xmlfull-importidle1202009-09-0521:28:08Indexing completed. Added/Updated: 2 documents. Deleted 0documents.2009-09-05 21:28:092009-09-05 21:28:090:0:0.579This response formatis experimental. It is likely to change in the future.

五、再去查询你刚才提交的数据。

上面的例子只不过是很简单的一个部分。针对 solr 的 MultiCore，通过配置 db-data-config.xml 也可以实现，还有多表，或者多表关联等等操作只要在 db-data-config.xml 配置清楚都可以进行数据的导入。

在 solr1.4 中还有更多的扩展功能，这些功能为重建索引提供能很方便的操作。而且 datasource 不单单指的是 database，可以是 xml 文件，还可以是来自网络上的等等。

2） 增量更新索引

1、首先要确认表中有 last_modified 字段。

2、修改 C:\solr-tomcat\solr\conf\db\db-data-config.xml 文件，内容如下：

```
<dataConfig>

    <dataSource type="JdbcDataSource" driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@192.168.1.1:1521:数据库名" user="用户名" password="密码"/>

     <document name="messages">

    <entity name="message" pk="ID"

                        transformer="ClobTransformer"

                        query="select * from tb_message"

                        deltaQuery="select id from tb_message where
to_char(last_modified,'YYYY-MM-DD HH24:MI:SS') > '${dataimporter.last_index_time}'">

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

```
        <field column="TITLE" name="title" />
        <field column="CONTENT" clob="true" name="content" />
        <field column="SENDTIME" name="sendtime" />
      </entity>
    </document>
</dataConfig>
```

3、重启 tomcat。添加一条记录。

访问：http://localhost:8089/solr/dataimport?command=delta-import

再查询一下，是不是可以查询到刚才添加的记录了。

3.6 如何进行搜索

3.6.1 搜索语法

1. solr 查询参数说明

1） 常用

1) q - 查询字符串，必须的。

2) fl - 指定返回那些字段内容，用逗号或空格分隔多个。

3) start - 返回第一条记录在完整找到结果中的偏移位置，0 开始，一般分页用。

4) rows - 指定返回结果最多有多少条记录，配合 start 来实现分页。

5) sort - 排序，格式：sort=<field name>+<desc|asc>[,<fieldname>+<desc|asc>]…。示例：（inStock desc, price asc）表示先 “inStock”降序, 再 “price”升序，默认是相关性降序。

注：排序字段只能针对数值型如：int,dobuble 等…

6) wt - (writer type)指定输出格式，可以有 xml, json, php, phps,后面 solr 1.3 增加的，要用通知我们，因为默认没有打开。

7) fq -（filter query）过虑查询，作用：在 q 查询符合结果中同时是 fq 查询符合的，例如：q=mm&fq=date_time:[20081001 TO 20091031]，找关键字 mm，并且 date_time 是 20081001 到 20091031 之间的。官方文档：http://wiki.apache.org/solr/CommonQueryParameters#head-6522ef80f22d0e50d2f12ec487758577506d6002

2） 不常用

1) q.op - 覆盖 schema.xml 的 defaultOperator（有空格时用"AND"还是用"OR"操作逻辑），一般默认指定

2) df - 默认的查询字段，一般默认指定

3) qt -（query type）指定那个类型来处理查询请求，一般不用指定，默认是 standard。

3） 其它

1) indent - 返回的结果是否缩进，默认关闭，用 indent=true|on 开启，一般调试 json,php,phps,ruby 输出才有必要用这个参数。

2) version - 查询语法的版本，建议不使用它，由服务器指定默认值。

2. Solr 的检索运算符

1. “:” 指定字段查指定值，如返回所有值*:*

2. “?” 表示单个任意字符的通配

3. “*” 表示多个任意字符的通配（不能在检索的项开始使用*或者?符号）

4. “~” 表示模糊检索，如检索拼写类似于”roam”的项这样写：roam~将找到形如 foam 和 roams 的单词；roam~0.8，检索返回相似度在 0.8 以上的记录。

5. 邻近检索，如检索相隔 10 个单词的”apache”和”jakarta”，”jakarta apache”~10

6. “^” 控制相关度检索，如检索 jakarta apache，同时希望去让”jakarta”的相关度更加好，那么在其后加上”^”符号和增量值，即 jakarta^4 apache

7. 布尔操作符 AND、||

8. 布尔操作符 OR、&&

9. 布尔操作符 NOT、!、- （排除操作符不能单独与项使用构成查询）

10.“+” 存在操作符，要求符号”+”后的项必须在文档相应的域中存在

11. ( ) 用于构成子查询

12. [] 包含范围检索，如检索某时间段记录，包含头尾，date:[200707 TO 200710]

13. {} 不包含范围检索，如检索某时间段记录，不包含头尾

date:{200707 TO 200710}

注：范围检索字段只适用于:String,int,dobule,date 不能用于 long 型的字段

14. \ 转义操作符，特殊字符包括+ - && || ! ( ) { } [ ] ^ ” ~ * ? :\

3. solr 查询的一些常用语法

1、首先假设我的数据里 fields 有:name, tel, address 预设的搜寻是 name 这个字段, 如果要搜寻的数据刚好就是 name 这个字段,就不需要指定搜寻字段名称。

2、查询规则:

如欲查询特定字段(非预设字段),请在查询词前加上该字段名称加“:” (不包含”号) 符号,

例如: address:北京市海淀区上地软件园 tel:88xxxxx1

1>. q 代表 query input

2>. version 代表 solr 版本(建议不要变动此变量)

3>. start 代表显示结果从哪一笔结果资料开始,预设为 0 代表第一笔, rows 是说要显示几笔数据,预设为 10 笔

(因为有时查询结果可能有几百笔,但不需要显示所有结果,所以预设是从第一笔开始到第十笔)

所以若要显示第 10 到 30 笔就改为:

http://localhost:8080/solr/select/?indent=on&version=2.2&q=address:北京市海淀区上地软件园+tel:88xxxxx1&version=2.2&start=10&rows=20&indent=on

(indent 代表输出的 xml 要不要缩行.预设为开启 on)

3、另外,要限定输出结果的内容可用 “fl=” 加上你要的字段名称,如以下这个范例:

http://localhost:8080/solr/select/?indent=on&version=2.2&q=text:北京+ OR+text:亿度&start=0&rows=10&fl=name,address,tel

在 fl=之后加上了 name,adress,tel

所以结果会如下:

<result name=”response” numFound=”1340″ start=”0″>

<doc>

<str name=”name”>北京亿度</str>

<str name=”address”>北京市海淀区上地软件园</str>

<strname=”tel”>88xxxxxx1</str>

</doc>

<doc>

<str name=”name”>北京亿度</str>

<str name=”address”/>

<strname=”tel”>88xxxxxx1</str>

</doc>

</result>

4、查询 name 或 address:直接输入查询词, 如: 亿度

送出的内容即为:

name:亿度 AND address:海淀

5、若要搜寻联集结果,请在词与词间空格或加上大写 “OR”(不包含”号).

例如: text:海淀 OR text:亿度

text:海淀 OR 亿度

或

海淀 亿度

或

name:亿度 OR tel:88xxxxxx1

或

name:亿度 tel:88xxxxxx1

6、若要搜寻交集结果,请在词与词间加上大写 “AND” 或 “+” (不包含”号).

例如: text:海淀 AND 亿度

或

+text:海淀 +text:亿度

或

name:亿度 AND tel:88xxxxxx1

或

name: (+亿度 +海淀)

7、排除查询

在要排除的词前加上 “-” (不包含”号) 号

例如: 海淀 -亿度

搜寻结果不会有包含亿度的词的结果在内

8、Group 搜寻

使用 “()” 来包含一个 group

如希望搜寻在店名字段内同时有 “台北”(不包含”号) 及 “火车站”(不包含”号)

9、增加权重: 如要搜寻 “北京 加油站”(不包含”号) 但因为回传太多笔资料内有 “中华”(不包含”号) 或 “加油站”(不包含”号) 的结果,

所以想要把有包含 “加油站”(不包含”号)的数据往前排,可使用 “^”(不包含”号)符号在后面加上愈增加的权重数,

像是 “2″,则可以这样做:

北京 加油站^2

会同时搜寻含有北京或加油站的结果,并把加油站这个词加权所以搜寻时会先判断加油站这一个词在搜寻结果中的比重,甚至假设一笔数据内加油站出现过两次以上的就更加会有优先权。

查询时在查询词后加上 “^” (不包含”号) 再加上权重分数

例如: 亿度 AND “北京”^2

或

亿度^2 OR 北京

10、Wildcard 搜寻使用 “*” 符号;如果输入 “中国*银” (不包含”号), 结果会有中国信托商业银行, 中国输出入银行图书阅览室, 中国商银证券

中国及银之间可夹任何长短字词.

3.6.2 排序

3.6.3 字段增加权重

在很多时候，我们可能会需要增加某一个字段的权重，以合理的显示搜索结果。

例如：有一个 schma，有三个字段：chapterId,title, content.

我们希望某一个关键字如果在 title 中匹配了，就要优先显示，而在 content 中匹配了，就放在搜索结果的后面。当然，如果两者同时匹配当然没什么好说的了。看看 solr 中如何做到吧。

title:(test1 test2)^4 content:(test1 test2)

给 title 字段增加权重，优先匹配

关于^后面的数字 4，经过我测试，最佳值应该是有 n 个字段就写成 n+1，当然希望大家能更好的去测试！

3.6.4 Solr 分词器、过滤器、分析器

关于 lucene 的分析器，分词器，过滤器，请看：http://lianj-lee.javaeye.com/blog/501247

对一个 document 进行索引时，其中的每个 field 中的数据都会经历分析（根据上面的一个博客可以知道，分析就是组合分词和过滤），最终将一句话分成单个的单词，去掉句子当中的空白符号，大写转换小写，复数转单数，去掉多余的词，进行同义词代换等等。

如：This is a blog!this, is, a 会被去除，最后最剩下 blog。当然!这个符号也会被去除的。

这个过程是在索引和查询过程中都会进行的，而且通常两者进行的处理的都是一样的，这样做是为了保证建立的索引和查询的正确匹配。

分析器（Analyzer）

分析器是包括两个部分：分词器和过滤器。分词器功能将句子分成单个的词元 token，过滤器就是对词元进行过滤。

solr 自带了一些分词器，如果你需要使用自定义的分词器，那么就需要修改 schema.xml 文件。

schema.xml 文件允许两种方式修改文本被分析的方式，通常只有 field 类型为 solr.TextField 的 field 的内容允许定制分析器。

方法一：使用任何 org.apache.lucene.analysis.Analyzer 的子类进行设定。

```
<fieldType name="text"class="solr.TextField">
    <analyzerclass="org.wltea.analyzer.lucene.IKAnalyzer"/>
</fieldType>
```

方法二：指定一个 TokenizerFactory，后面跟一系列的 TokenFilterFactories（它们将按照所列的顺序发生作用），Factories 被用来创建分词器和分词过滤器，它们用于对分词器和分词过滤器的准备配置，这样做的目的是为了避免 the overhead of creation via reflection。

```
<analyzertype="index">
    <tokenizerclass="org.wltea.analyzer.solr.IKTokenizerFactory"isMaxWordLength="false"/>
……
  </analyzer>
  <analyzer type="query">
    <tokenizerclass="org.wltea.analyzer.solr.IKTokenizerFactory"isMaxWordLength="true"/>
  ……
  </analyzer>
```

需要说明的一点是，Any Analyzer,TokenizerFactory, or TokenFilterFactory 应该用带包名的全类名进行指定，请确保它们位于 Solr 的 classpath 路径下。对于 org.apache.solr.analysis.* 包下的类，仅仅通过 solr.*就可以进行指定。

如果你需要使用自己的分词器和过滤器，你就需要自己写一个 factory ，它必须是 BaseTokenizerFactory(分词器) 或 BaseTokenFilterFactory（过滤器）的子类。就像下面一样。

```
public class MyFilterFactory extendsBaseTokenFilterFactory {
 public TokenStream create(TokenStream input) {
  return new MyFilter(input);
```

}

}

对于 IK3.1.5 版本已经完全支持了 solr 的分词，这样就不用自己来编写了， 而对于中文的切词的话，ik 对 solr 的支持已经很完美了。

Solr 提供了哪些 TokenizerFactories？

1. solr.LetterTokenizerFactory

创建 org.apache.lucene.analysis.LetterTokenizer.

分词举例：

"I can't" ==> "I", "can", "t"，字母切词。

2. solr.WhitespaceTokenizerFactory

创建 org.apache.lucene.analysis.WhitespaceTokenizer，主要是切除所有空白字符。

3. solr.LowerCaseTokenizerFactory

创建 org.apache.lucene.analysis.LowerCaseTokenizer

分词举例：

"I can't" ==> "i", "can", "t"，主要是大写转小写。

4. solr.StandardTokenizerFactory

创建 org.apache.lucene.analysis.standard.StandardTokenizer

分词举例： "I.B.M. cat's can't" ==>

ACRONYM: "I.B.M.",APOSTROPHE:"cat's", APOSTROPHE:"can't"

说明：该分词器，会自动地给每个分词添加 type，以便接下来的对 type 敏感的过滤器进行处理，目前仅仅只有 StandardFilter 对 Token 的类型是敏感的。

5. solr.HTMLStripWhitespaceTokenizerFactory

从结果中除去 HTML 标签，将结果交给 WhitespaceTokenizer 处理。

例子：

my <ahref="www.foo.bar">link

my link

<?xml?>
hello<!--comment-->

hello

hello<script><--f('<--internal--></script>'); --></script>

hello

if a<b then print a;

if a<b then print a;

hello <td height=22 nowrapalign="left">

hello

a<b &#65 Alpha&OmegaΩ

a<b A Alpha&Omega Ω

6. solr.HTMLStripStandardTokenizerFactory

从结果中出去 HTML 标签，将结果交给 StandardTokenizer 处理。

7. solr.PatternTokenizerFactory

说明：按照规则表达式样式对分本进行分词。

例子：处理对象为，mice; kittens; dogs，他们由分号加上一个或多个的空格分隔。

```
<fieldTypename="semicolonDelimited" class="solr.TextField">
 <analyzer>
    <tokenizer class="solr.PatternTokenizerFactory"pattern="; *" />
 </analyzer>
</fieldType>
```

Solr 有哪些 TokenFilterFactories？

1. solr.StandardFilterFactory

创建：org.apache.lucene.analysis.standard.StandardFilter.

移除首字母简写中的点和 Token 后面的’s。仅仅作用于有类的 Token，他们是由 StandardTokenizer 产生的。

例：StandardTokenizer+ StandardFilter

"I.B.M. cat's can't" ==>"IBM", "cat", "can't"

2. solr.LowerCaseFilterFactory

创建：org.apache.lucene.analysis.LowerCaseFilter.

3. solr.TrimFilterFactory【solr1.2】

创建：org.apache.solr.analysis.TrimFilter

去掉 Token 两端的空白符

例：

" Kittens! ", "Duck" ==>"Kittens!", "Duck".

4. solr.StopFilterFactory

创建：org.apache.lucene.analysis.StopFilter

去掉如下的通用词，多为虚词。

"a", "an", "and", "are","as", "at", "be", "but","by",

"for", "if", "in", "into","is", "it",

"no", "not", "of", "on","or", "s", "such",

"t", "that", "the", "their","then", "there", "these",

"they", "this", "to", "was","will", "with"

自定义的通用词表的使用可以通过 schema.xml 文件中的"words"属性来指定，如下。

<fieldtype name="teststop"class="solr.TextField">

<analyzer>

<tokenizer class="solr.LowerCaseTokenizerFactory"/>

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

5. solr.KeepWordFilterFactory【solr1.3】

创建：org.apache.solr.analysis.KeepWordFilter

作用与 solr.StopFilterFactory 相反，保留词的列表也可以通过”word”属性进行指定。

```
<fieldtype name="testkeep"class="solr.TextField">
  <analyzer>
    <filter class="solr.KeepWordFilterFactory"words="keepwords.txt" ignoreCase="true"/>
  </analyzer>
</fieldtype>
```

6. solr.LengthFilterFactory

创建：solr.LengthFilter

过滤掉长度在某个范围之外的词。范围设定方式见下面。

```
<fieldtype name="lengthfilt"class="solr.TextField">
 <analyzer>
   <tokenizer class="solr.WhitespaceTokenizerFactory"/>
   <filter class="solr.LengthFilterFactory" min="2"max="5" />
 </analyzer>
```

</fieldtype>

7. solr.PorterStemFilterFactory

创建：org.apache.lucene.analysis.PorterStemFilter

采用 Porter Stemming Algorithm 算法去掉单词的后缀，例如将复数形式变成单数形式，第三人称动词变成第一人称，现在分词变成一般现在时的动词。

8. solr.EnglishPorterFilterFactory

创建：solr.EnglishPorterFilter

关于句子主干的处理，其中的"protected"指定不允许修改的词的文件。

9. solr.SnowballPorterFilterFactory

关于不同语言的词干处理

10.solr.WordDelimiterFilterFactory

关于分隔符的处理。

11.solr.SynonymFilterFactory

关于同义词的处理。

12.solr.RemoveDuplicatesTokenFilterFactory

避免重复处理。

3.6.5 Solr 高亮使用

1、SolrQuery 类，此类有方法 setHighlight(true)，当设置为 true 时，表示开启了高亮。

2、SolrQuery 类，有方法：

```
// 以下给两个字段开启了高亮，分别是 name，description，

query.addHighlightField("name");

query.addHighlightField("description");

// 以下两个方法主要是在高亮的关键字前后加上 html 代码

query.setHighlightSimplePre("<fontcolor=\"red\">");

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

3、下面是获取高亮的内容：

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

Map 的 Key 为 document 的 Id，即你在 schema.xml 中设置的 Id，Value 为该 Id 对应的 document 的值，Value 也为一个 Map，该 Map 的 Key 为 fieldName，Value 为 List<String>，这个 List 里面的内容就是该文档的高亮字段。

所以当做逻辑处理的时候，只要按照这个层次，依次把东西给取出来即可，如果取出来的东西为空，则用 QueryResponse 中的 SolrDocument 的 getFieldValue(filedName)的值。

对了，请注意在 solrConfig.xml 中开启高亮组件，这个可以看看官方 wiki 或者看 solrconfig.xml 中注释！

4 SolrJ 的用法

SolrJ 是**访问 Solr 的 Java 客户端框架**，它提供添加索引、更新索引及搜索查询 Solr 的接口。以 JAR 包的方式提供（JavaDoc 文档：http://lucene.apache.org/solr/api/solrj/index.html）

4.1 搜索接口的调用实例

```
public class SolrJSearch {


  private static final String SOLR_URL =
```

```
    //"http://10.0.8.10:8081/solr/spacearticle/";
    "http://localhost:8080/solr/";
  private CommonsHttpSolrServer solrServer = null;

  public SolrJSearch(){
    try {
      solrServer = new CommonsHttpSolrServer(SOLR_URL);
solrServer.setMaxTotalConnections(100);
      solrServer.setSoTimeout(10000);// socket read timeout
      solrServer.setConnectionTimeout(5000);

    } catch (MalformedURLException e) {
      e.printStackTrace();
    }
  }

  public void search(){
    SolrQuery query = new SolrQuery();
    query.setQuery("A_Title:新浪科技");

  //新加条件（and 的作用对 TERM_VAL 的 6 到 8 折的限制）
```

```
query.addFilterQuery("TERM_VAL:{6TO 8 折}");
//排序用的
//query.addSortField( "price", SolrQuery.ORDER.asc );

    try {
        QueryResponse rsp = solrServer.query( query );

        SolrDocumentList docs = rsp.getResults();
        System.out.println("文档个数：" + docs.getNumFound());
        System.out.println("查询时间：" + rsp.getQTime());
        for (SolrDocument doc : docs) {
            String title = (String) doc.getFieldValue("A_Title");
            Integer id = (Integer) doc.getFieldValue("A_ID");
            System.out.println(id);
            System.out.println(title);
        }
    } catch (SolrServerException e) {
        e.printStackTrace();
    }
}
```

```
  public static void main(String[] args) {
      SolrJSearch sj = new SolrJSearch();
      sj.search();
  }
}
```

4.2 Solrj 的使用说明

solrj 被设计成一个可扩展的框架，用以向 solr 服务器提交请求，并接收回应。我们已经将最通用的一些命令封装在了 solrServer 类中了。

4.2.1 Adding Data to Solr

首先需要获得一个 server 的实例，

Java 代码

```
1.  SolrServer server = getSolrServer();
```

如果，你使用的是一个远程的 solrServer 的话呢，你或许会这样来实现 getSolrServer()这个方法：

Java 代码

```
1.  public SolrServer getSolrServer(){
2.      //the instance can be reused
3.      return new CommonsHttpSolrServer();
4.  }
```

· 如果，你使用的是一个本地的 solrServer 的话，你或许会这样来实现 getSolrServer()方法：

Java 代码

```
1.  public SolrServer getSolrServer(){
2.      //the instance can be reused
3.      return new EmbeddedSolrServer();
4.  }
```

· 如果，你在添加数据之前，想清空现有的索引，那么你可以这么做：

Java 代码

```
1.  server.deleteByQuery( "*:*" );// delete everything!
```

· 构造一个 document

Java 代码

```
1.  SolrInputDocument doc1 = new SolrInputDocument();
2.   doc1.addField( "id", "id1", 1.0f );
3.   doc1.addField( "name", "doc1", 1.0f );
4.   doc1.addField( "price", 10 );
```

· 构造另外一个文档，每个文档都能够被独自地提交给 solr，但是，批量提交是更高效的。每一个对 SolrServer 的请求都是 http 请求，当然对于 EmbeddedSolrServer 来说，是不一样的。

Java 代码

```
1.  SolrInputDocument doc2 = new SolrInputDocument();
2.  doc2.addField( "id", "id2", 1.0f );
3.  doc2.addField( "name", "doc2", 1.0f );
```

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

· 构造一个文档的集合

Java 代码

```
1.  Collection docs = new  ArrayList();

2.   docs.add( doc1 );

3.   docs.add( doc2 );
```

· 将 documents 提交给 solr

Java 代码

```
1.  server.add( docs );
```

· 提交一个 commit

Java 代码

```
1.  server.commit();
```

· 在添加完 documents 后，立即做一个 commit，你可以这样来写你的程序：

Java 代码

```
1.  UpdateRequest req = new UpdateRequest();

2.   req.setAction( UpdateRequest.ACTION.COMMIT, false, false );

3.   req.add( docs );

4.   UpdateResponse rsp = req.process( server );
```

4.2.2 Directly adding POJOs to Solr

· 使用 java 注释创建 java bean。@Field ，可以被用在域上，或者是 setter 方法上。如果一个域的名称跟 bean 的名称是不一样的，那么在 java 注释中填写别名，具体的，可以参照下面的域 categories

Java 代码

```
1.  import org.apache.solr.client.solrj.beans.Field;
2.    public class Item {
3.     @Field
4.     String id;
5.
6.     @Field("cat")
7.     String[] categories;
8.
9.     @Field
10.    List features;
11.
12.  }
```

· java 注释也可以使用在 setter 方法上，如下面的例子：

Java 代码

```
1.  @Field("cat")
```

```
2.    public void setCategory(String[] c){
3.        this.categories = c;
4.    }
```

这里应该要有一个相对的，get 方法（没有加 java 注释的）来读取属性

· Get an instance of server

```
Java 代码
1.    SolrServer server = getSolrServer();
```

· 创建 bean 实例

```
Java 代码
1.    Item item = new Item();
2.     item.id = "one";
 item.categories =  new String[] { "aaa", "bbb", "ccc" };
```

· 添加给 solr

```
Java 代码
1.    server.addBean(item);
```

· 将多个 bean 提交给 solr

```
Java 代码
1.    List beans ;
```

```
2.    //add Item objects to the list

3.    server.addBeans(beans);
```

注意：你可以重复使用 SolrServer，这样可以提高性能。

4.2.3 Reading Data from Solr

· 获取 solrserver 的实例

```
Java 代码

1.  SolrServer server = getSolrServer();
```

· 构造 SolrQuery

```
Java 代码

1.  SolrQuery query = new SolrQuery();

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

query.addSortField( "price", SolrQuery.ORDER.asc );
```

· 向服务器发出查询请求

```
Java 代码

QueryResponse rsp = server.query( query );
```

· 获取结果。

```
Java 代码

1.  SolrDocumentList docs = rsp.getResults();
```

· 想要以javabean的方式获取结果，那么这个javabean必须像之前的例子一样有java注释。

```
Java 代码

List beans = rsp.getBeans(Item.class);
```

4.3 创建查询

solrJ 提供了一组 API，来帮助我们创建查询，下面是一个 faceted query 的例子。

```
Java 代码

SolrServer server = getSolrServer();

 SolrQuery solrQuery = new  SolrQuery().

            setQuery("ipod").

            setFacet(true).

            setFacetMinCount(1).

            setFacetLimit(8).

            addFacetField("category").

            addFacetField("inStock");

QueryResponse rsp = server.query(solrQuery);
```

所有的 setter/add 方法都是返回它自己本身的实例，所以就像你所看到的一样，上面的用法是链式的。

4.4 使用 SolrJ 创建索引

```
Collection<SolrInputDocument> docs = new HashSet<SolrInputDocument>();

for (int i = 0; i < 10; i++) {
```

```
  SolrInputDocument doc = new SolrInputDocument();

  doc.addField("link", "http://non-existent-url.foo/" + i + ".html");

  doc.addField("source", "Blog #" + i);

  doc.addField("source-link", "http://non-existent-url.foo/index.html");

  doc.addField("subject", "Subject: " + i);

  doc.addField("title", "Title: " + i);

  doc.addField("content", "This is the " + i + "(th|nd|rd) piece of content.");

  doc.addField("category", CATEGORIES[rand.nextInt(CATEGORIES.length)]);

  doc.addField("rating", i);

  //System.out.println("Doc[" + i + "] is " + doc);

  docs.add(doc);

}
```

以上代码中循环只是创建了 SolrInputDocument（实际是一个夸张的 Map），然后给它添加 Field。我将它添加到了一个集合中，这样一次就能将所有的文档发送到 Solr。借助这个功能可以极大地加索引的创??，并减少通过 HTTP 发送请求导致的开销。然后我调用了 UpdateResponseresponse = server.add(docs);，它负责序列化文档并将其提交到 Solr。UpdateResponse 返回的值包含处理文档所用的时间的信息。为了让这些文档能够被搜索到，我又发出一个提交命令：server.commit();。

4.5 Solrj 包的结构说明

4.5.1 CommonsHttpSolrServer

CommonsHttpSolrServer 使用 HTTPClient 和 solr 服务器进行通信。

Java 代码

```
1.  String url = "http://localhost:8983/solr";

2.  SolrServer server = new CommonsHttpSolrServer( url );
```

4.5.2 Setting XMLResponseParser

sorlr J 目前使用二进制的格式作为默认的格式。对于 solr1.2 的用户通过显示的设置才能使用 XML 格式。

Java 代码

```
1.  server.setParser(new XMLResponseParser());
```

4.5.3 Changing other Connection Settings

CommonsHttpSorlrServer 允许设置链接属性。

```
1.  String url = "http://localhost:8983/solr"

2.   CommonsHttpSolrServer server = new CommonsHttpSolrServer( url );

3.   server.setSoTimeout(1000);  // socket read timeout

4.   server.setConnectionTimeout(100);

5.   server.setDefaultMaxConnectionsPerHost(100);

6.   server.setMaxTotalConnections(100);

7.   server.setFollowRedirects(false);  // defaults to false

8.   // allowCompression defaults to false.

9.   // Server side must support gzip or deflate for this to have any effect.

10.  server.setAllowCompression(true);

11.  server.setMaxRetries(1); // defaults to 0.  > 1 not recommended.
```

4.5.4 EmbeddedSolrServer

EmbeddedSorrServer 提供和 CommonsHttpSorlrServer 相同的接口，它不需要 http 连接。

Java 代码

```
1.  SolrCore core = SolrCore.getSolrCore();
2.  SolrServer server = new EmbeddedSolrServer( core );
3.  ...
```

如果你想要使用 Multicore 特性，那么你可以这样使用：

Java 代码

```
1.  File home = new File( getSolrHome() );
2.  File f = new File( home, "solr.xml" );
3.  multicore.load( getSolrHome(), f );
4.
EmbeddedSolrServer server = new EmbeddedSolrServer( multicore, "core name as defined in solr.xml" );
```

注：如果你在你的项目中内嵌 solr 服务，这将是一个不错的选择。无论你能否使用 http，它都提供相同的接口。

5 Solr 的实际应用测试报告

5.1 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

5.2 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）

6 solr 性能调优

6.1 Schema Design Considerations

6.1.1 indexedfields

indexed fields 的数量将会影响以下的一些性能：

l 索引时的时候的内存使用量

l 索引段的合并时间

l 优化时间

l 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。

6.1.2 storedfields

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。

6.2 ConfigurationConsiderations

6.2.1 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）

6.2.2 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。

6.3 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。

6.4 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。

6.5 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"eventlisteners 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。

6.6 OptimizationConsiderations

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。

6.7 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。

6.8 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。

6.9 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%。

6.10 RAM UsageConsiderations（内存方面的考虑）

6.10.1 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。

6.10.2 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。

1. 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。

l 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。

l 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

7 FAQ

7.1 出现乱码或者查不到结果的排查方法：

1) Tomcat 的 server.xml 需要保证：

如果没有设置 URIEncoding="UTF-8"，在提交查询的 select 的 url 会出现乱码，当然也就查不到了。

2) 添加索引的时候，xml 数据文件需要包含 utf-8 声明，也就是：

然后使用 Solr 自带的 post.jar 或者 curl 来进行 post 索引：

java: java -Durl=http://localhost:8080/solr/update -Dcommit=yes-jar post.jar data.xml

该步骤如果没有设置好，出错的表现是，查询的结果是乱码(可以设置查询关键词为 id:[*TO *]确保显示出所有的结果).

3) 如果确保了如上的两个步骤都设置正确，但是使用 http://localhost:8080/solr/admin/的查询表单(该表单支持 utf-8 没有问题)仍旧查不到结果，则问题可能出在分词器，可以尝试不同的分词器。