Solr

全文检索服务

**讲师：传智.入云龙**

企业站内搜索技术选型

在一些大型门户网站、电子商务网站等都需要站内搜索功能，使用传统的数据库查询方式实现搜索无法满足一些高级的搜索需求，比如：搜索速度要快、搜索结果按相关度排序、搜索内容格式不固定等，这里就需要使用全文检索技术实现搜索功能。

* 1. 单独使用Lucene实现

单独使用Lucene实现站内搜索需要开发的工作量较大，主要表现在：索引维护、索引性能优化、搜索性能优化等，因此不建议采用。

* 1. 使用Google或Baidu接口

通过第三方搜索引擎提供的接口实现站内搜索，这样和第三方引擎系统依赖紧密，不方便扩展，不建议采用。

* 1. 使用Solr实现

基于Solr实现站内搜索扩展性较好并且可以减少程序员的工作量，因为Solr提供了较为完备的搜索引擎解决方案，因此在门户、论坛等系统中常用此方案。

什么是Solr

什么是Solr

Solr 是Apache下的一个顶级开源项目，采用Java开发，它是基于Lucene的全文搜索服务器。Solr提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展，并对索引、搜索性能进行了优化。

Solr可以独立运行，运行在Jetty、Tomcat等这些Servlet容器中，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 Field 及其内容的 XML 文档，Solr根据xml文档添加、删除、更新索引 。Solr 搜索只需要发送 HTTP GET 请求，然后对 Solr 返回Xml、json等格式的查询结果进行解析，组织页面布局。Solr不提供构建UI的功能，Solr提供了一个管理界面，通过管理界面可以查询Solr的配置和运行情况。



Solr与Lucene的区别

Lucene是一个开放源代码的全文检索引擎工具包，它不是一个完整的全文检索引擎，Lucene提供了完整的查询引擎和索引引擎，目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者以Lucene为基础构建全文检索引擎。

Solr的目标是打造一款企业级的搜索引擎系统，它是一个搜索引擎服务，可以独立运行，通过Solr可以非常快速的构建企业的搜索引擎，通过Solr也可以高效的完成站内搜索功能。

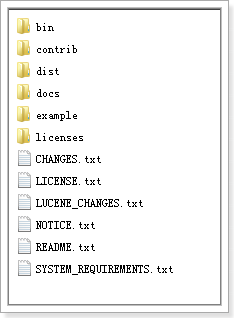
# Solr安装配置

## 下载

从Solr官方网站（http://lucene.apache.org/solr/ ）下载Solr4.10.3，根据Solr的运行环境，Linux下需要下载lucene-4.10.3.tgz，windows下需要下载lucene-4.10.3.zip。

Solr使用指南可参考：https://wiki.apache.org/solr/FrontPage。

下载lucene-4.10.3.zip并解压：



bin：solr的运行脚本

contrib：solr的一些贡献软件/插件，用于增强solr的功能。

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。

docs：solr的API文档

example：solr工程的例子目录：

* example/solr：

该目录是一个包含了默认配置信息的Solr的Core目录。

* example/multicore：

该目录包含了在Solr的multicore中设置的多个Core目录。

* example/webapps：

该目录中包括一个solr.war，该war可作为solr的运行实例工程。

licenses：solr相关的一些许可信息

## 运行环境

solr 需要运行在一个Servlet容器中，Solr4.10.3要求jdk使用1.7以上，Solr默认提供Jetty（java写的Servlet容器），本教程使用Tocmat作为Servlet容器，环境如下：

Solr：Solr4.10.3

Jdk：jdk1.7.0\_72

Tomcat：apache-tomcat-7.0.53

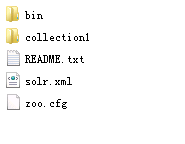
## Solr与Tomcat整合配置

### Solr Home与SolrCore

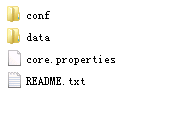
创建一个Solr home目录，SolrHome是Solr运行的主目录，目录中包括了运行Solr实例所有的配置文件和数据文件，Solr实例就是SolrCore，一个SolrHome可以包括多个SolrCore（Solr实例），每个SolrCore提供单独的搜索和索引服务。

目录结构

example\solr是一个solr home目录结构，如下：



上图中“collection1”是一个SolrCore（Solr实例）目录 ，目录内容如下所示：



说明：

collection1：叫做一个Solr运行实例SolrCore，SolrCore名称不固定，一个solr运行实例对外单独提供索引和搜索接口。

solrHome中可以创建多个solr运行实例SolrCore。

一个solr的运行实例对应一个索引目录。

conf是SolrCore的配置文件目录 。

配置

创建目录 F:\develop\solr

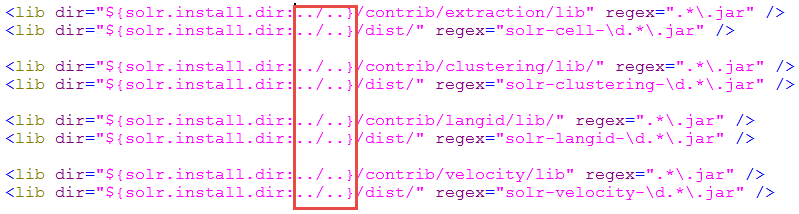
1. 将example\solr目录 拷贝至 F:\develop\solr目录下并改名为solrhome

#### solrconfig.xml

solrconfig.xml，在SolrCore的conf目录下，它是SolrCore运行的配置文件。

##### 加载jar包

将contrib和dist两个目录拷贝到F:\develop\solr下，修改solrconfig.xml文件：



##### dataDir

配置SolrCore的数据目录，数据目录下包括了index索引目录 和tlog日志文件目录，数据目录默认在solrCore下的data目录 ，也可以更改目录地址 ，如下：



##### requestHandler

requestHandler请求处理器，定义了索引和搜索的访问方式。

通过/update维护索引，可以完成索引的添加、修改、删除操作。



提交xml、json数据完成索引维护，索引维护小节详细介绍。

通过/select搜索索引。



设置搜索参数完成搜索，搜索参数也可以设置一些默认值，如下：

<requestHandler name="/select" class="solr.SearchHandler">

<!-- 设置默认的参数值，可以在请求地址中修改这些参数-->

<lst name="defaults">

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

<int name="rows">10</int><!--显示数量-->

<str name="wt">json</str><!--显示格式-->

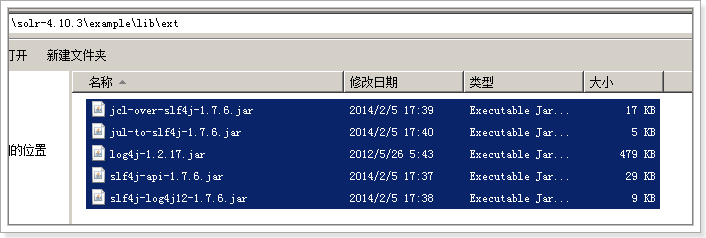
<str name="df">text</str><!--默认搜索字段-->

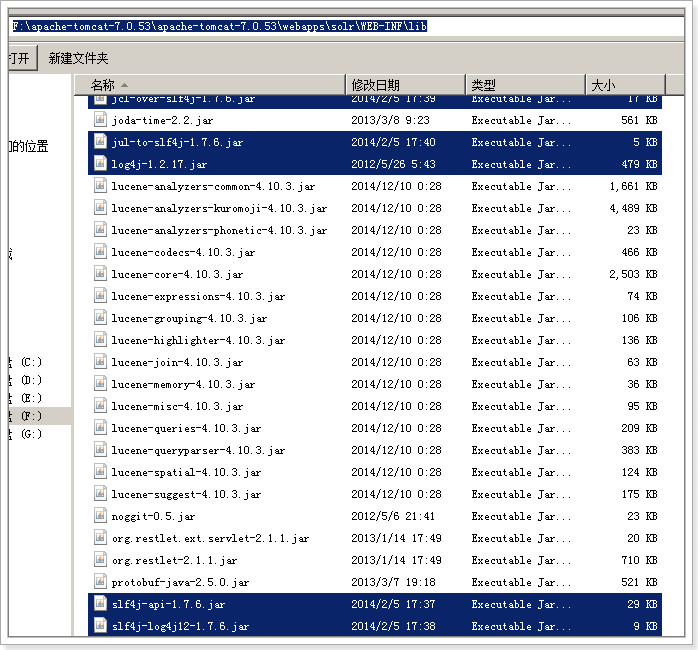
</lst>

</requestHandler>

Solr工程部署

1. 将dist\solr-4.10.3.war拷贝到Tomcat的webapp目录下改名为solr.war
2. 启动tomcat后，solr.war自动解压，将原来的solr.war删除。
3. 拷贝example\lib\ext 目录下所有jar包到Tomcat的webapp\solr\WEB-INF\lib目录下





1. 修改Tomcat目录 下webapp\solr\WEB-INF\web.xml文件，如下所示：

设置Solr home

<env-entry>

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

<env-entry-value>f:/develop/solr/solrhome</env-entry-value>

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

</env-entry>

1. 拷贝log4j.properties文件

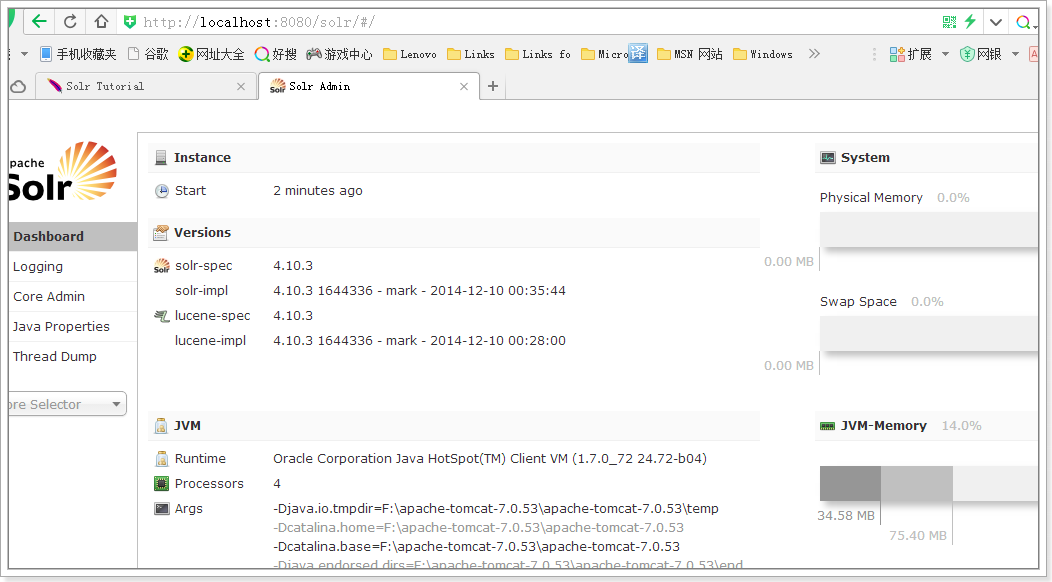
在 Tomcat下webapps\solr\WEB-INF目录中创建文件 classes文件夹，

复制Solr目录下example\resources\log4j.properties至Tomcat下webapps\solr\WEB-INF\classes目录

### 启动Tomcat

访问http://localhost:8080/solr

## 管理界面



### Dashboard：

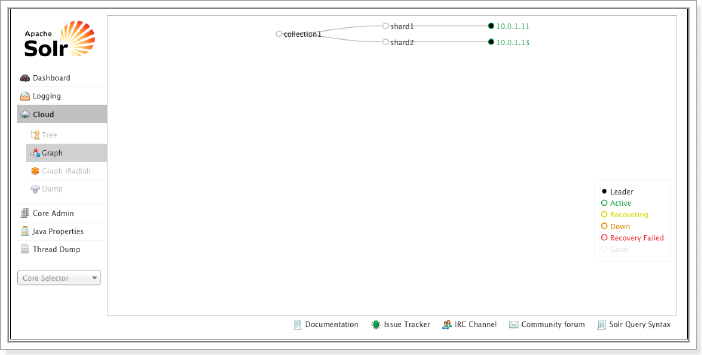
仪表盘，显示了该Solr实例开始启动运行的时间、版本、系统资源、jvm等信息。

### Logging：

Solr运行日志信息

### Cloud：

Cloud即SolrCloud，即Solr云（集群），当使用Solr Cloud模式运行时会显示此菜单，如下图是Solr Cloud的管理界面：



### Core Admin：

Solr Core的管理界面。Solr Core 是Solr的一个独立运行实例单位，它可以对外提供索引和搜索服务，一个Solr工程可以运行多个SolrCore（Solr实例），一个Core对应一个索引目录。

### java properties

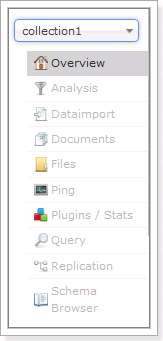
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

### Tread Dump

显示Solr Server中当前活跃线程信息，同时也可以跟踪线程运行栈信息。

### Core selector

选择一个SolrCore进行详细操作，如下：



### Analysis（重点）



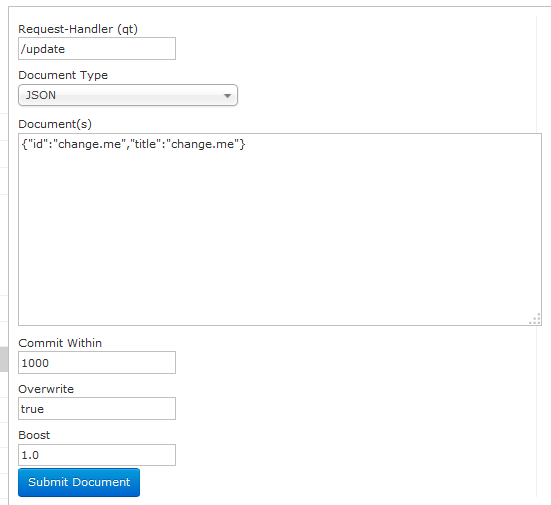
通过此界面可以测试索引分析器和搜索分析器的执行情况。

### dataimport

可以定义数据导入处理器，从关系数据库将数据导入 到Solr索引库中。

Document（重点）

通过此菜单可以创建索引、更新索引、删除索引等操作，界面如下：



/update表示更新索引，solr默认根据id（唯一约束）域来更新Document的内容，如果根据id值搜索不到id域则会执行添加操作，如果找到则更新。

### query（重点）

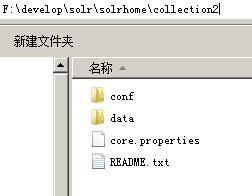


通过/select执行搜索索引，必须指定“q”查询条件方可搜索。

## 多core配置

复制原来的core目录为collection2，目录结构如下：





修改collection2下的core.properties，如下：

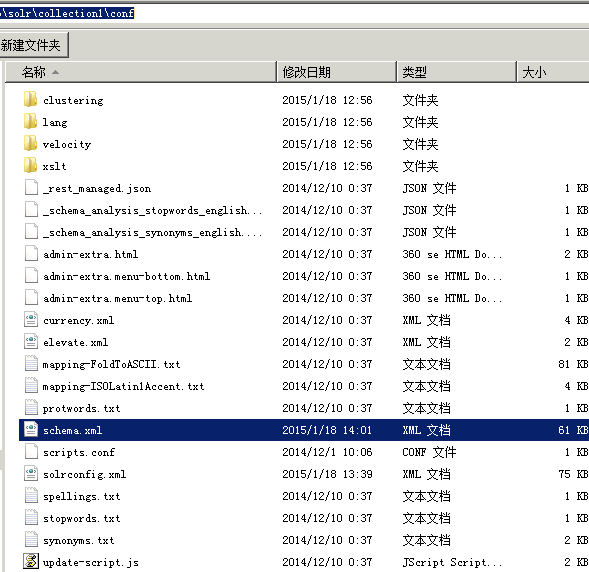


演示多core的使用，强调通常情况在sor集群下使用多core。

# Solr索引

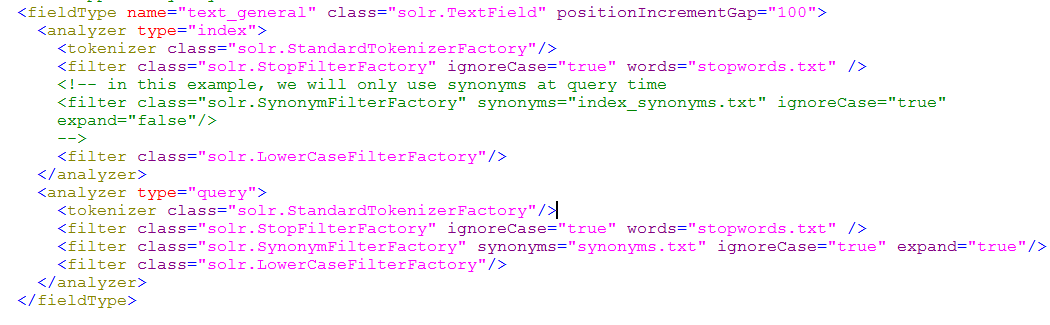
## scheam.xml

schema.xml，在SolrCore的conf目录下，它是Solr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FieldTypes、Fields和其他的一些缺省设置。



FieldType域类型定义

下边“text\_general”是Solr默认提供的FieldType，通过它说明FieldType定义的内容：



FieldType子结点包括：name,class,positionIncrementGap等一些参数：

name：是这个FieldType的名称

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

positionIncrementGap：可选属性，定义在同一个文档中此类型数据的空白间隔，避免短语匹配错误，此值相当于Lucene的短语查询设置slop值，根据经验设置为100。

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

索引分析器中：使用solr.StandardTokenizerFactory标准分词器，solr.StopFilterFactory停用词过滤器，solr.LowerCaseFilterFactory小写过滤器。

搜索分析器中：使用solr.StandardTokenizerFactory标准分词器，solr.StopFilterFactory停用词过滤器，这里还用到了solr.SynonymFilterFactory同义词过滤器。

Field定义

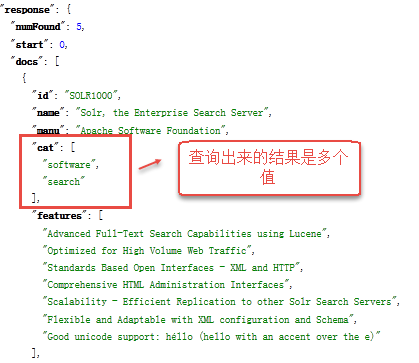
在fields结点内定义具体的Field，filed定义包括name,type（为之前定义过的各种FieldType）,indexed（是否被索引）,stored（是否被储存），multiValued（是否存储多个值）等属性。

如下：

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

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

multiValued：该Field如果要存储多个值时设置为true，solr允许一个Field存储多个值，比如存储一个用户的好友id（多个），商品的图片（多个，大图和小图），通过使用solr查询要看出返回给客户端是数组：



uniqueKey

Solr中默认定义唯一主键key为id域，如下：



Solr在删除、更新索引时使用id域进行判断，也可以自定义唯一主键。

copyField复制域

copyField复制域，可以将多个Field复制到一个Field中，以便进行统一的检索：

比如，输入关键字搜索title标题内容content，

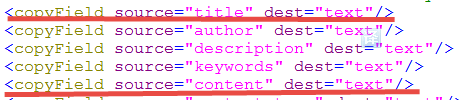
定义title、content、text的域：







根据关键字只搜索text域的内容就相当于搜索title和content，将title和content复制到text中，如下：



dynamicField（动态字段）

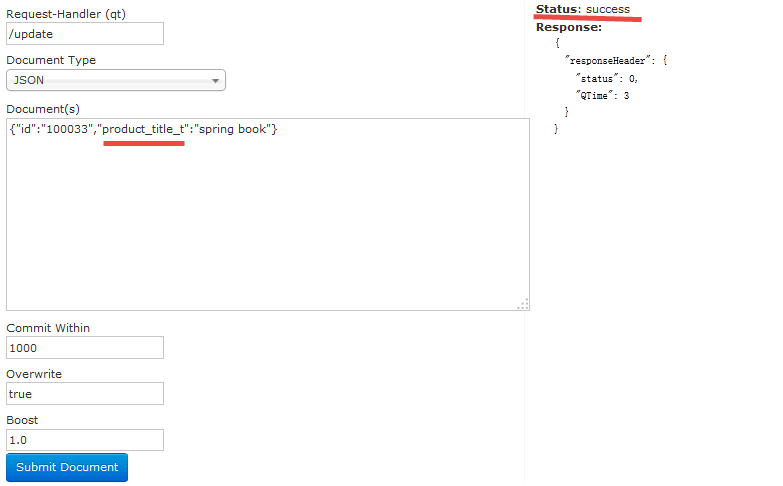
动态字段就是不用指定具体的名称，只要定义字段名称的规则，例如定义一个 dynamicField，name 为\*\_i，定义它的type为text，那么在使用这个字段的时候，任何以\_i结尾的字段都被认为是符合这个定义的，例如：name\_i，gender\_i，school\_i等。

自定义Field名为：product\_title\_t，“product\_title\_t”和scheam.xml中的dynamicField规则匹配成功，如下：

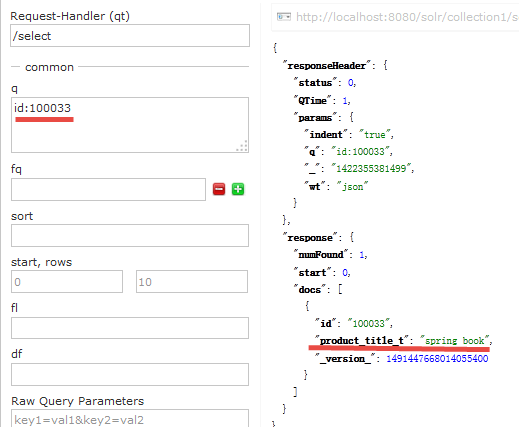


“product\_title\_t”是以“\_t”结尾。

创建索引：



搜索索引：



## Analyzer

安装中文分词器

#### IKAnalyzer部署

拷贝IKAnalyzer的文件到Tomcat下Solr目录 中

将IKAnalyzer2012FF\_u1.jar拷贝到 Tomcat的webapps/solr/WEB-INF/lib 下。

在Tomcat的webapps/solr/WEB-INF/下创建classes目录

将IKAnalyzer.cfg.xml、ext\_stopword.dic mydict.dic copy到 Tomcat的

webapps/solr/WEB-INF/classes

注意：ext\_stopword.dic 和mydict.dic必须保存成无BOM的utf-8类型。

#### 修改schema.xml文件

1. FieldType

首先需要在types结点内定义一个FieldType子结点，包括name,class,等参数，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

修改Solr的schema.xml文件，添加FieldType：

<!-- IKAnalyzer-->

<fieldType name="text\_ik" class="solr.TextField">

<analyzer type="index" isMaxWordLength="false" class="org.wltea.analyzer.lucene.IKAnalyzer"/>

<analyzer type="query" isMaxWordLength="true" class="org.wltea.analyzer.lucene.IKAnalyzer"/>

</fieldType>

**其中查询采用IK自己的最大分词法,索引则采用它的细粒度分词法.所以各自配置了isMaxWordLength属性.**

1. Field：

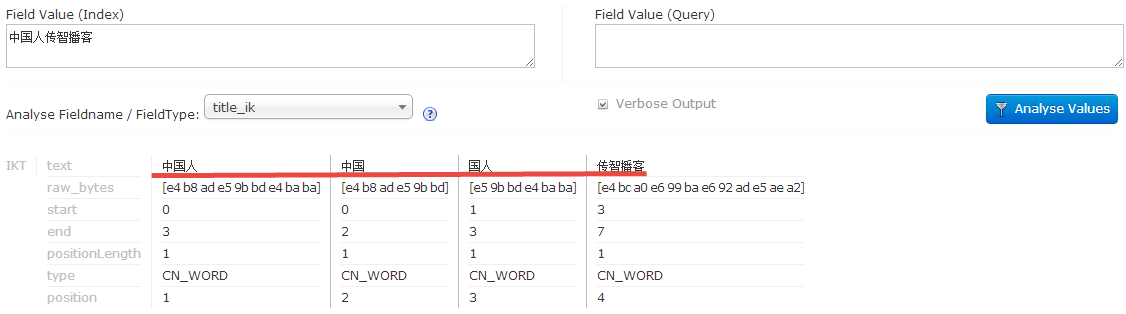
FieldType定义好后就可以在fields结点内定义具体的field，filed定义包括name,type（即FieldType）,indexed（是否被索引）,stored（是否被储存），multiValued（是否有多个值）等

<!--IKAnalyzer Field-->

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

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

1. 测试



## 设置业务系统Field

如果不使用Solr提供的Field可以针对具体的业务需要自定义一套Field，如下是商品信息Field：

<!--product-->

<field name="product\_name" type="text\_ik" indexed="true" stored="true"/>

<field name="product\_price" type="float" indexed="true" stored="true"/>

<field name="product\_description" type="text\_ik" indexed="true" stored="false" />

<field name="product\_picture" type="string" indexed="false" stored="true" />

<field name="product\_catalog\_name" type="string" indexed="true" stored="true" />

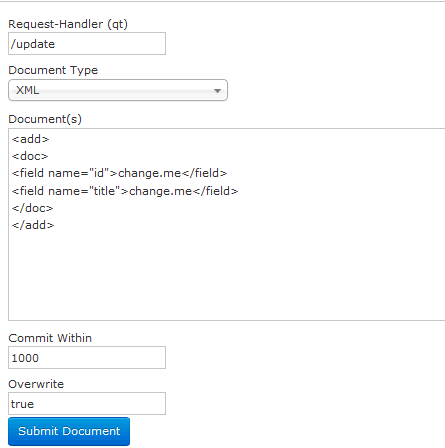
<field name="product\_keywords" type="text\_ik" indexed="true" stored="false" multiValued="true"/>

<copyField source="product\_name" dest="product\_keywords"/>

<copyField source="product\_description" dest="product\_keywords"/>

## 索引维护

使用/update进行索引维护，进入Solr管理界面SolrCore下的Document下：



overwrite="true" ： solr在做索引的时候，如果文档已经存在，就用xml中的文档进行替换

commitWithin="10000" ： solr 在做索引的时候，每个10000（10秒）毫秒，做一次文档提交。

为了方便测试也可以在Document中立即提交，在</add>后边添加“<commit/>”，如下：

<add>

<doc>

<field name="id">change.me</field>

<field name="title">change.me</field>

</doc>

</add>

<commit/>

添加/更新索引

solr默认根据id（唯一约束）域来更新Document的内容，如果根据id值搜索不到id域则会执行添加操作，如果找到则更新。

请求xml格式如下：

<add>

<doc>

<field name="id">change.me</field>

<field name="？？" >？？</field>

。。。

</doc>

</add>

说明：唯一标识 Field必须有，这里使用Solr默认的id。

删除索引

删除索引格式如下：

1） 删除制定ID的索引

<delete>

<id>8</id>

。。。

</delete>

2） 删除查询到的索引数据

<delete>

<query>product\_catalog\_name:幽默杂货</query>

</delete>

3） 删除所有索引数据

<delete>

<query>\*:\*</query>

</delete>

## SolrJ完成索引维护

什么是SolrJ

solrj是访问Solr服务的java客户端，提供索引和搜索的请求方法，SolrJ通常在嵌入在业务系统中，通过SolrJ的API接口操作Solr服务，如下图：

Index索引库

javaEE应用程序

SolrJ程序客户端

Solr服务

Tomcat

通过SolrJ请求Solr服务

最终Solr完在索引和搜索

创建索引

使用SolrJ创建索引，通过调用SolrJ提供的API请求Solr服务，Document通过SolrInputDocument进行构建。

// 创建索引

**public** **void** testCreateIndex() **throws** SolrServerException, IOException {

SolrServer solrServer = **new** HttpSolrServer(*urlString*);

SolrInputDocument document = **new** SolrInputDocument();

document.addField("id", "c0001");

document.addField("product\_name", "传智java教程");//商品名称

document.addField("product\_price", 86.5f);//商品价格

document.addField("product\_picture", "382782828.jpg");//商品图片

document.addField("product\_description", "这是一本深入浅出讲解java技术的书籍！");//商品描述

document.addField("product\_catalog\_name", "javabook");//商品分类

UpdateResponse response = solrServer.add(document);

// 提交

solrServer.commit();

}

说明：根据id（唯一约束）域来更新Document的内容，如果根据id值搜索不到id域则会执行添加操作，如果找到则更新。

删除索引（自学）

上边介绍的删除方法，使用SolrJ也可以完成，代码如下：

// 删除索引

@Test

**public** **void** testDeleteIndex() **throws** SolrServerException, IOException {

SolrServer solrServer = **new** HttpSolrServer(*urlString*);

//根据id删除

UpdateResponse response = solrServer.deleteById("c0001");

//根据多个id删除

// solrServer.deleteById(ids);

//自动查询条件删除

// solrServer.deleteByQuery("product\_keywords:教程");

// 提交

solrServer.commit();

}

说明：deleteById(String id)根据id删除索引，此方法为重载方法，也可以传个多个id批量删除，也可以调用deleteByQuery() 根据查询条件删除

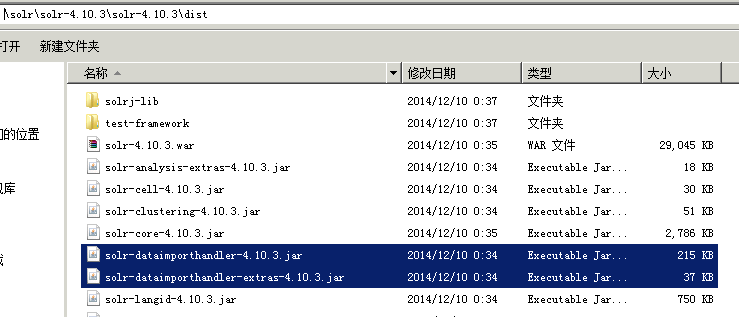
## dataimport-handler

安装dataimport-Handler从关系数据库将数据导入到索引库。

第一步：向SolrCore中加入jar包

在SolrCore目录中创建lib目录，将dataimportHandler和mysql数据库驱动的jar拷贝至lib下：

dataimportHandler在solr安装目录的dist 下：



第二步：修改solrconfig.xml，添加requestHandler：

<requestHandler name="/dataimport"

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

<lst name="defaults">

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

</lst>

</requestHandler>

第三步：编辑data-config.xml文件，存放在SolrCore的conf目录

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

<dataConfig>

<dataSource type="JdbcDataSource"

driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/lucene"

user="root"

password="mysql"/>

<document>

<entity name="product" query="SELECT pid,name,catalog\_name,price,description,picture FROM products ">

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

<field column="name" name="product\_name"/>

<field column="catalog\_name" name="product\_catalog\_name"/>

<field column="price" name="product\_price"/>

<field column="description" name="product\_description"/>

<field column="picture" name="product\_picture"/>

</entity>

</document>

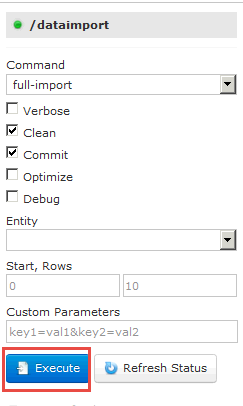
</dataConfig>

说明：

<field column="pid" name="id"/>必须有一个id域，这里使用Solr默认的id域，域值是从关系数据库查询的pid列值。

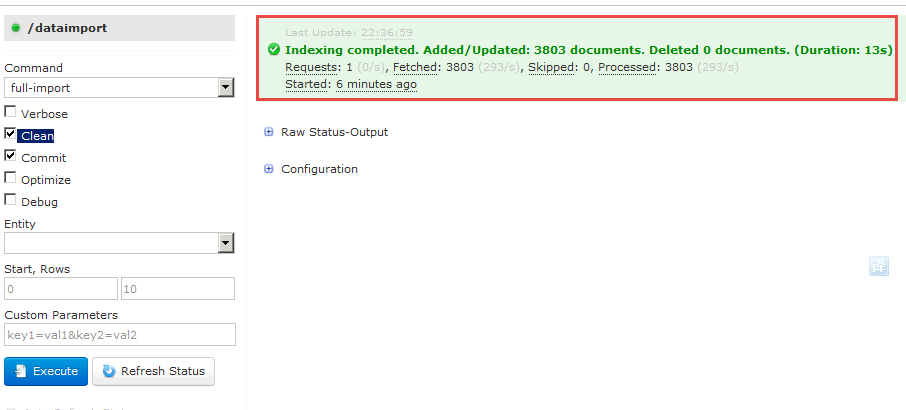
下边以“product\_”开头的Field都是在schema.xml中自定义的商品信息Field。

第四步：重启Tomcat，进入管理界面--》SolrCore--》dataimport下执行导入



第五步：查看导入结果

进入管理界面--》SolrCore--》dataimport下



# Solr搜索

<requestHandler name="/select" class="solr.SearchHandler">

<!-- 设置默认的参数值，可以在请求地址中修改这些参数-->

<lst name="defaults">

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

<int name="rows">10</int><!--显示数量-->

<str name="wt">json</str><!--显示格式-->

<str name="df">text</str><!--默认搜索字段-->

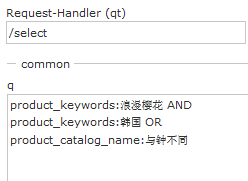
</lst>

</requestHandler>

## 查询语法

通过/select搜索索引，Solr制定一些参数完成不同需求的搜索：

1. q - 查询字符串，必须的，如果查询所有使用\*:\*。

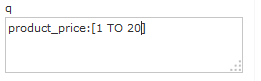


1. fq - （filter query）过虑查询，作用：在q查询符合结果中同时是fq查询符合的，例如：：



过滤查询价格从1到20的记录。

也可以在“q”查询条件中使用product\_price:[1 TO 20]，如下：



也可以使用“\*”表示无限，例如：

20以上：product\_price:[20 TO \*]

20以下：product\_price:[\* TO 20]

1. sort - 排序，格式：sort=<field name>+<desc|asc>[,<field name>+<desc|asc>]… 。示例：

按价格降序

1. start - 分页显示使用，开始记录下标，从0开始
2. rows - 指定返回结果最多有多少条记录，配合start来实现分页。



显示前10条。

1. fl - 指定返回那些字段内容，用逗号或空格分隔多个。

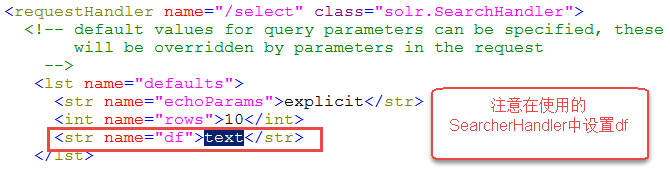


显示商品图片、商品名称、商品价格

1. df-指定一个搜索Field



也可以在SolrCore目录 中conf/solrconfig.xml文件中指定默认搜索Field，指定后就可以直接在“q”查询条件中输入关键字。



1. 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。
2. hl 是否高亮 ,设置高亮Field，设置格式前缀和后缀。



## SolrJ完成搜索

搜索索引

// 搜索

@Test

**public** **void** testSearch() **throws** SolrServerException {

SolrServer solr = **new** HttpSolrServer(*urlString*);

// 查询对象

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

//设置查询条件,名称“q”是固定的且必须 的

//搜索product\_keywords域，product\_keywords是复制域包括product\_name和product\_description

query.set("q", "product\_keywords:java教程");

// 请求查询

QueryResponse response = solr.query(query);

// 查询结果

SolrDocumentList docs = response.getResults();

// 查询文档总数

System.*out*.println("查询文档总数" + docs.getNumFound());

**for** (SolrDocument doc : docs) {

//商品主键

String id = (String) doc.getFieldValue("id");

//商品名称

String product\_name = (String) doc.getFieldValue("product\_name");

//商品价格

Float product\_price = (Float) doc.getFieldValue("product\_price");

//商品图片

String product\_picture = (String) doc.getFieldValue("product\_picture");

//商品分类

String product\_catalog\_name = (String) doc.getFieldValue("product\_catalog\_name");

System.*out*.println("=============================");

System.*out*.println(id);

System.*out*.println(product\_name);

System.*out*.println(product\_price);

System.*out*.println(product\_picture);

System.*out*.println(product\_catalog\_name);

}

}

组合查询

// 根据商品分类、价格范围、关键字查询，查询结果按照价格降序排序

@Test

**public** **void** testSearch2() **throws** SolrServerException {

SolrServer solr = **new** HttpSolrServer(*urlString*);

// 查询对象

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

// 搜索product\_keywords域，product\_keywords是复制域包括product\_name和product\_description

// 设置商品分类、关键字查询

// query.set("q", "product\_keywords:挂钩 AND product\_catalog\_name:幽默杂货");

query.setQuery("product\_keywords:挂钩 AND product\_catalog\_name:幽默杂货");

// 设置价格范围

query.set("fq", "product\_price:[1 TO 20]");

// 查询结果按照价格降序排序

// query.set("sort", "product\_price desc");

query.addSort("product\_price", ORDER.*desc*);

// 请求查询

QueryResponse response = solr.query(query);

// 查询结果

SolrDocumentList docs = response.getResults();

// 查询文档总数

System.*out*.println("查询文档总数" + docs.getNumFound());

**for** (SolrDocument doc : docs) {

// 商品主键

String id = (String) doc.getFieldValue("id");

// 商品名称

String product\_name = (String) doc.getFieldValue("product\_name");

// 商品价格

Float product\_price = (Float) doc.getFieldValue("product\_price");

// 商品图片

String product\_picture = (String) doc

.getFieldValue("product\_picture");

// 商品分类

String product\_catalog\_name = (String) doc

.getFieldValue("product\_catalog\_name");

System.*out*.println("=============================");

System.*out*.println("id=" + id);

System.*out*.println("product\_name=" + product\_name);

System.*out*.println("product\_price=" + product\_price);

System.*out*.println("product\_picture=" + product\_picture);

System.*out*.println("product\_catalog\_name=" + product\_catalog\_name);

}

}

分页、高亮

// 分页和高亮

@Test

**public** **void** testSearch3() **throws** SolrServerException {

SolrServer solr = **new** HttpSolrServer(*urlString*);

// 查询对象

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

// 设置商品分类、关键字查询

query.setQuery("product\_keywords:透明挂钩 ");

// 分页参数

// 每页显示记录数

**int** pageSize = 2;

// 当前页码

**int** curPage = 2;

// 开始记录下标

**int** begin = pageSize \* (curPage - 1);

// 起始下标

query.setStart(begin);

// 结束下标

query.setRows(pageSize);

// 设置高亮参数

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

query.addHighlightField("product\_name");// 高亮字段

query.setHighlightSimplePre("<span color='red'>");// 前缀标记

query.setHighlightSimplePost("</span>");// 后缀标记

// 请求查询

QueryResponse response = solr.query(query);

// 查询结果

SolrDocumentList docs = response.getResults();

// 查询文档总数

System.*out*.println("查询文档总数" + docs.getNumFound());

**for** (SolrDocument doc : docs) {

// 商品主键

String id = (String) doc.getFieldValue("id");

// 商品名称

String product\_name = (String) doc.getFieldValue("product\_name");

// 商品价格

Float product\_price = (Float) doc.getFieldValue("product\_price");

// 商品图片

String product\_picture = (String) doc

.getFieldValue("product\_picture");

// 商品分类

String product\_catalog\_name = (String) doc

.getFieldValue("product\_catalog\_name");

System.*out*.println("=============================");

System.*out*.println("id=" + id);

System.*out*.println("product\_name=" + product\_name);

System.*out*.println("product\_price=" + product\_price);

System.*out*.println("product\_picture=" + product\_picture);

System.*out*.println("product\_catalog\_name=" + product\_catalog\_name);

// 高亮信息

**if** (response.getHighlighting() != **null**) {

**if** (response.getHighlighting().get(id) != **null**) {

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

.get(id);// 取出高亮片段

**if** (map.get("product\_name") != **null**) {

**for** (String s : map.get("product\_name")) {

System.*out*.println(s);

}

}

}

}

}

}

## 相关度排序

索引时设置boost

在创建索引时设置boost值：可以针对Field设置boost，也可以针对Document设置boost值，如下：

<add>

<doc>

<field name="id">a01</field>

<field name="product\_name" boost="20.0">幸福一家人</field>

<field name="product\_description" >我们是幸福一家人</field>

</doc>

<doc boost="30.0">

<field name="id">a02</field>

<field name="product\_name">幸福两家人</field>

<field name="product\_description" >我们是幸福两家人</field>

</doc>

</add>

<commit/>

针对Field设置boost必须搜索时匹配到该Field在计算相关度得分时才有效，针对Document设置Field则在搜索时只要匹配到该Document的任意Field在计算相关度得分都有效。

测试：

如果要搜索product\_description，关键字“幸福”，“a02”的doc会排在前边，因为“a02”是整体设置了boost值。

搜索时设置boost

在创建搜索时设置域的boost值：搜索匹配到的域如果boost值设置的高则计算的相关度得分就可能会高，如下：

创建索引：

<add>

<doc>

<field name="id">a03</field>

<field name="product\_name">lucene实战</field>

<field name="product\_description" >这是一本讲解全文检索技术的书籍</field>

</doc>

<doc>

<field name="id">a04</field>

<field name="product\_name">全文检索</field>

<field name="product\_description" >Lucene是一个全文检索工具包</field>

</doc>

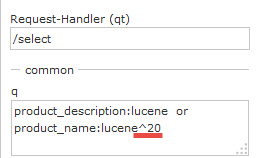
</add>

<commit/>

说明：“a03”文档的product\_name中有lucene，product\_description中没有，

“a04”文档中的product\_name中没有lucene，"product\_description中有lucene。

product\_name和product\_description组合域搜索时设置product\_name的boost值高点，如下：



当product\_name中匹配到重要性要比product\_description高，查询结果是product\_name中匹配到关键字的排在前边，如下：

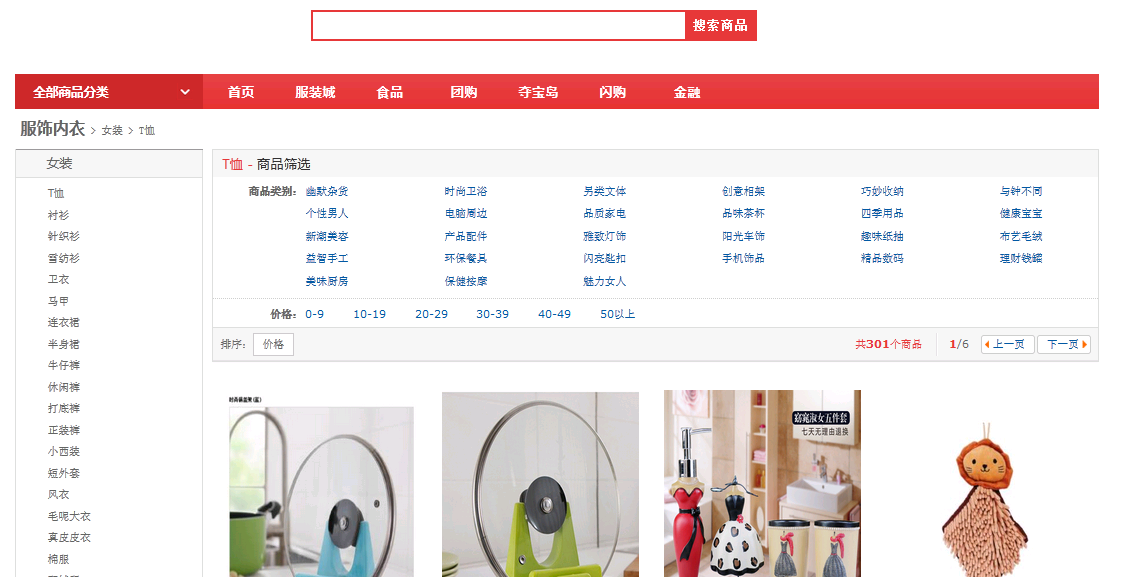


案例

需求

使用Solr实现电商网站中商品信息搜索功能，可以根据关键字、分类、价格搜索商品信息，也可以根据价格进行排序。

界面如下：



系统架构

spring容器

表现层

springmvc

Service层

Dao层

SolrJ

Solr索引库

mysql数据库

商品信息

Tomcat

Solr服务

索引、搜索请求