

Solr 框架技术

课程目标

- 1、 上章回顾
- 2、 Solr 概述
- 3、 Solr 安装及配置
- 4、 Solr 管理索引库
- 5、 Solr 整合 SpringMVC-案例：京东商城

课程内容

1、 上章回顾

- 1、 Lucene 是 Apache 开源的全文检索的工具包 创建索引 查询索引
- 2、 遇到问题？ 文件名 及文件内容 顺序扫描法 全文检索
- 3、 什么是全文检索？ 这种先创建索引 再对索引进行搜索的过程叫全文检索
- 4、 索引是什么？ 非结构数据中提取一个数据、并重新组合的过程叫索引
- 5、 Lucene 实现
- 6、 入门程序

磁盘文件为原始文件

创建索引

第一步：获取文件

第二步：创建文档对象

第三步：创建分析器

第四步：保存索引及文档到索引库

搜索索引

第一步：用户接口（百度）

第二步：创建 Query 查询对象（KV）域名：值

第三步：执行查询

第四步：渲染

- 7、 分析器（中文分析器）

标准

中日韩

SmartChineseAnalyzer

IKAnalyzer (扩展、停止)

8、使用 IK 分析器

第一步：导入 IK.jar

第二步：复制 ik.cfg.xml

第三步：复制 stopwords.dic ext.dic (**必须使用 Editplus 保存为无 BOM 的 UTF-8 格式**)

全放在 classpath 下

9、Lucene 的索引维护

添加

删除

修改

查询

10、高级查询

第一个：全查询 MatchAllDocsQuery

*: *

第二个：TermQuery 精准查询

第三个：区间查询（根据数值）L

第四个：组合查询 BooleadQuery 多个 AND OR NOT Occur.MUST MUST_NOT SHOULD

2、 Solr 概述

2.1) 什么是 Solr:

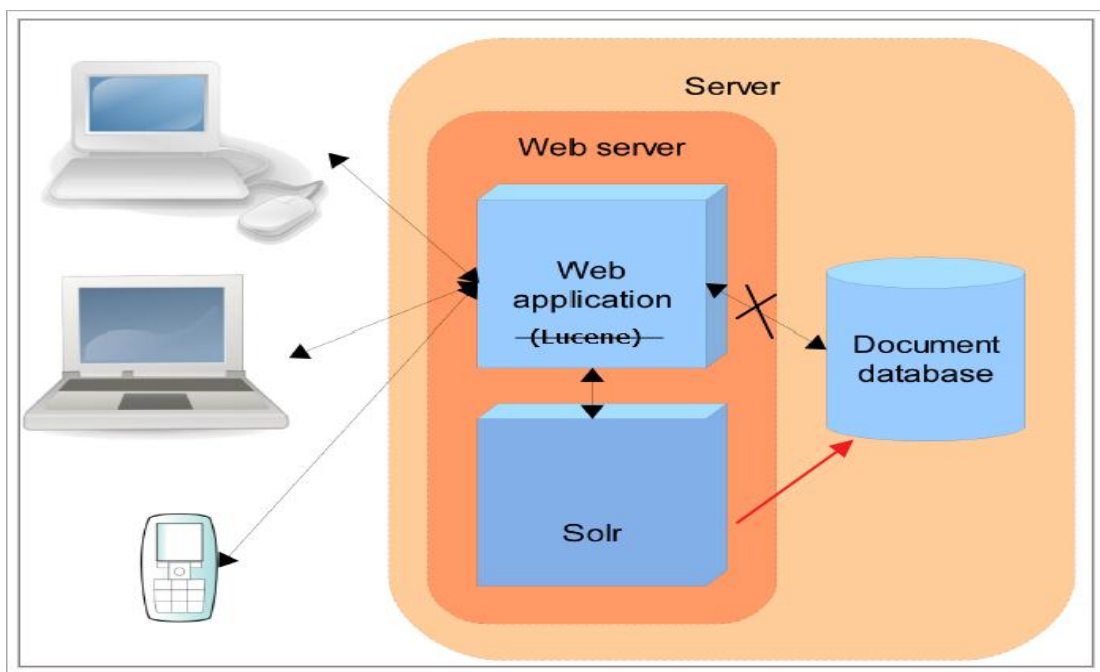
Solr 是 Apache 下的一个顶级开源项目，采用 Java 开发，它是基于 Lucene 的全文搜索服务器。Solr 提供了比 Lucene 更为丰富的查询语言，同时实现了可配置、可扩展，并对索引、搜索性能进行了优化。

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。

2.2) 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。



3、 Solr 安装及配置

3.1) 下载 Solr:

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>。

3.2) solr 的目录结构：



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

3.3) Solr 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 Tomcat 作为 Servlet 容器，环境如下：

Solr: Solr4.10.3

Jdk: jdk1.7.0_72

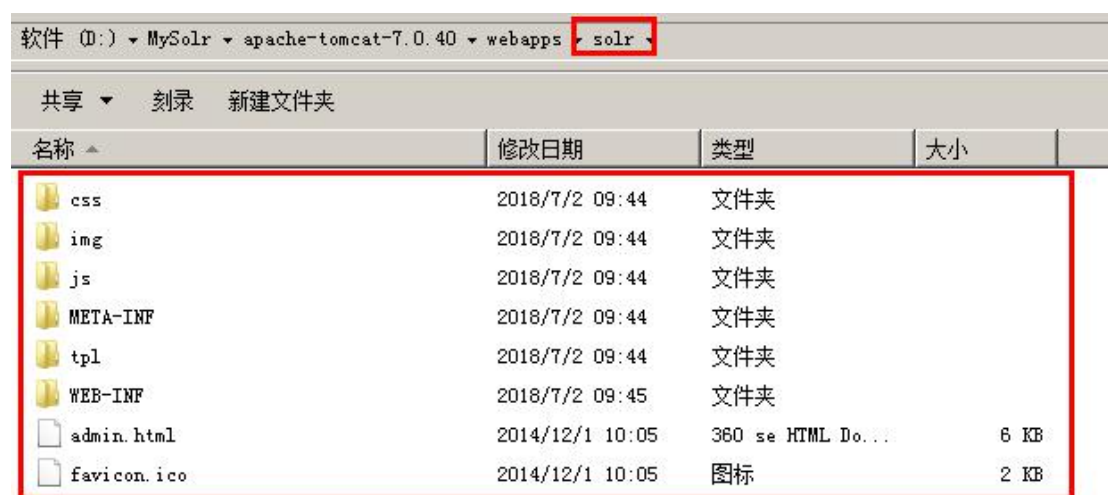
Tomcat: apache-tomcat-7.0.53

3.4) Solr 整合 Tomcat:

第一步：在 d:/mySolr 目录，将 Tomcat7.40 解压到其中。



第二步：从下载的 Solr4.10.3.zip 包中将 solr.war 文件复制到 tomcat/webapps 目录中并解压并删除 solr.war 文件。



第三步：复制 Solr4.10.3.zip 包中 solr-4.10.3\example\lib\ext 目录中的所有的 jar 文件到上面的 solr 工程中的 /WEB-INF/lib 目录中：

软件 (D:) > MySolr > apache-tomcat-7.0.40 > webapps > solr > WEB-INF > lib

名称	修改日期	类型	大小
antlr-runtime-3.5.jar	2013/1/4 22:35	Executable Jar...	164 KB
asm-4.1.jar	2012/10/14 11:37	Executable Jar...	47 KB
asm-commons-4.1.jar	2012/10/14 11:38	Executable Jar...	38 KB
commons-cli-1.2.jar	2009/3/19 16:08	Executable Jar...	41 KB
commons-codec-1.9.jar	2013/12/20 22:57	Executable Jar...	258 KB
commons-configuration-1.6.jar	2009/2/4 14:44	Executable Jar...	292 KB
commons-fileupload-1.2.1.jar	2008/2/11 02:07	Executable Jar...	57 KB
commons-io-2.3.jar	2012/4/10 11:12	Executable Jar...	179 KB
commons-lang-2.6.jar	2011/1/16 17:21	Executable Jar...	278 KB
concurrentlinkedhashmap-lru-1.2.jar	2011/5/20 21:06	Executable Jar...	52 KB
dom4j-1.6.1.jar	2005/8/1 22:27	Executable Jar...	307 KB
guava-14.0.1.jar	2013/3/14 19:57	Executable Jar...	2,138 KB
hadoop-annotations-2.2.0.jar	2013/10/7 02:28	Executable Jar...	17 KB
hadoop-auth-2.2.0.jar	2013/10/7 02:28	Executable Jar...	49 KB
hadoop-common-2.2.0.jar	2013/10/7 02:30	Executable Jar...	2,672 KB
hadoop-hdfs-2.2.0.jar	2013/10/7 02:32	Executable Jar...	5,120 KB
hpc-0.5.2.jar	2013/7/9 03:55	Executable Jar...	1,259 KB
httpclient-4.3.1.jar	2013/10/3 15:42	Executable Jar...	572 KB

将下载的solr4.10.3中的 example/lib/ext目录中的 扩展库包放到这里。

第四步：将 solr-4.10.3 下的 example 下的 solr 目录复制到 d:/mySolr 目录中并改名为：solrHome

cene@Solr资料 > 资料 > 01.参考资料 > solr-4.10.3 > example

名称	修改日期	类型	大小
contexts	2017/1/2 21:17	文件夹	
etc	2017/1/2 21:17	文件夹	
example-DIH	2017/1/2 21:17	文件夹	
exampledocs	2017/1/2 21:17	文件夹	
example-schemaless	2017/1/2 21:17	文件夹	
lib	2017/1/2 21:17	文件夹	
logs	2014/12/1 10:06	文件夹	
multicore	2017/1/2 21:17	文件夹	
resources	2017/1/2 21:17	文件夹	
scripts	2017/1/2 21:17	文件夹	
solr	2017/1/2 21:17	文件夹	
solr-webapp	2014/12/9 22:52	文件夹	
webapps	2017/1/2 21:17	文件夹	
README.txt	2014/12/10 00:37	文本文档	3 KB
start.jar	2013/3/12 11:55	Executable Jar...	46 KB

复制此目录到 d:/mySolr目录下 并改名为solrHome

软件 (D:) \ MySolr			
共享 刻录 新建文件夹			
名称	修改日期	类型	大小
apache-tomcat-7.0.40	2013/5/5 08:55	文件夹	
solrHome	2018/7/2 09:48	文件夹	

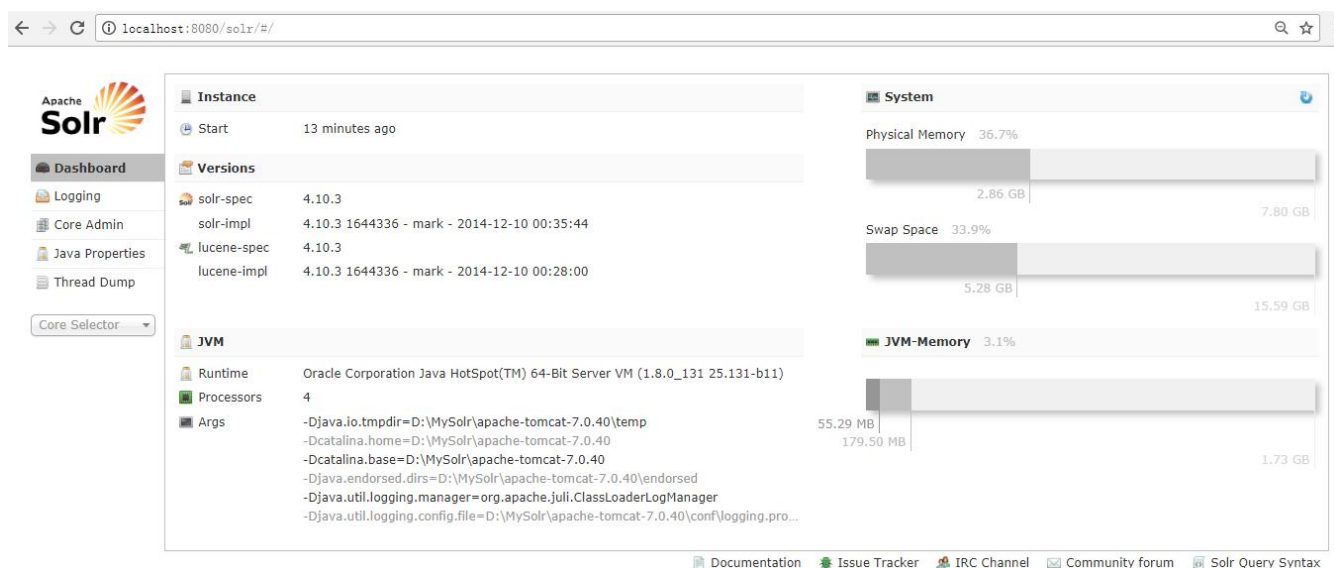
第五步: 在 apache-tomcat-7.0.40 的 webapps/solr/WEB-INF/web.xml 文件中配置上面的 solrHome

<!--这里代表修改solrHome的位置-->

```
<env-entry>
  <env-entry-name>solr/home</env-entry-name>
  <env-entry-value>D:\MySolr\solrHome</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

配置solrHome目录

第六步: 运行 Solr 后台管理程序:



4、 Solr 管理索引库

4.1) 如何创建一个新的实例：

第一步：复制 collection1 为 collection2

bin	2014/12/1 10:06	文件夹
collection1	2018/7/2 09:56	文件夹
collection2	2018/7/2 10:15	文件夹
README.txt	2014/12/10 00:37	文本文档
solr.xml	2014/12/10 00:37	XML 文档
zoo.cfg	2014/12/10 00:37	CFG 文件

第一步：复制此目录并改名为：collection2

第二步：将其中的 core.properties 的内容修改：
name=collection2

conf	2018/7/2 10:15	文件夹
data	2018/7/2 10:15	文件夹
lib	2018/7/2 09:50	文件夹
core.properties	2018/7/2 10:16	PROPERTIES 文件
README.txt	2014/12/10 00:37	文本文档

在刚才新建的collection2中的这个文件中将文件的内容改为：name=collection2

第三步：查看效果：

The screenshot shows the Solr Admin interface. On the left, a sidebar lists 'collection1' and 'collection2', with 'collection2' selected. The main content area displays the configuration for the 'Core' named 'collection2'. The configuration includes:

- Core:**
 - startTime: 5 minutes ago
 - instanceDir: D:\MySolr\solrHome\collection1\
 - dataDir: D:\MySolr\solrHome\collection1\data\
- Index:**
 - lastModified: -
 - version: 1
 - numDocs: 0
 - maxDoc: 0
 - deletedDocs: -
 - optimized: ✓
 - current: ✓
 - directory: org.apache.lucene.store.NRTCachingDirectory:NRTCachingDirectory(MMapDirectory@D:\MySolr\solrHome\collection1\data\index lockFactory=NativeFSLockFactory@D:\MySolr\solrHome\collection1\data\index; maxCacheMB=48.0 maxMergeSizeMB=4.0)

At the bottom of the interface, there are links for Documentation, Issue Tracker, IRC Channel, Community forum, and Solr Query Syntax.

4.2) 后台管理界面：

1.1.1. 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。

1.1.2. Logging

Solr 运行日志信息

1.1.3. Cloud

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



1.1.4. java properties

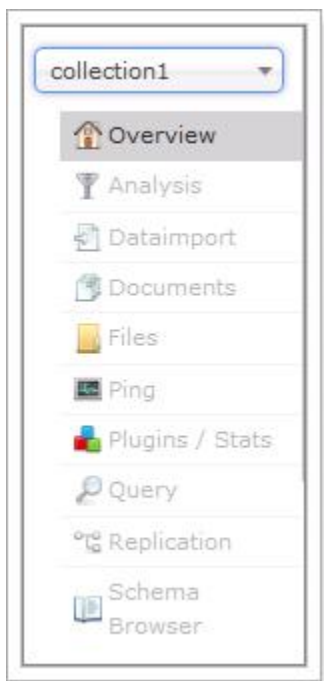
Solr 在 JVM 运行环境中的属性信息，包括类路径、文件编码、jvm 内存设置等信息。

1.1.5. 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。

1.1.6. Core selector

选择一个 SolrCore 进行详细操作，如下：



1.1.7. Analysis



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。

1.1.8. 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。

1.1.9. Document

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：

Request-Handler (qt)

/update

Document Type

JSON

Document(s)

{ "id": "change.me", "title": "change.me" }

Commit Within

1000

Overwrite

true

Boost

1.0

Submit Document

/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。

1.1.10 Query

Request-Handler (qt)

/select

— common —

q
,

fq
title:solr

sort

start, rows
0 10

fl

df

Raw Query Parameters
key1=val1&key2=val2

wt
json

☒ indent

☐ debugQuery

☐ dismax

☐ edismax

— ☒ hl —

hl.fl
title

hl.simple.pre
<em style='color:red'>

hl.simple.post

☐ hl.requireFieldMatch

☐ hl.usePhraseHighlighter

☐ hl.highlightMultiTerm

☐ facet

☐ spatial

☐ spellcheck

Execute Query

http://localhost:8080/solr/collection1/select?q=*&fq=title%3Asolr&s

```

{
  "responseHeader": {
    "status": 0,
    "QTime": 5,
    "params": {
      "q": "*,*",
      "hl": "true",
      "hl.simple.post": "</em>",
      "indent": "true",
      "fq": "title:solr",
      "sort": "id desc",
      "hl.fl": "title",
      "wt": "json",
      "hl.simple.pre": "<em style='color:red'>",
      "_": "1530500798552"
    }
  },
  "response": {
    "numFound": 2,
    "start": 0,
    "docs": [
      {
        "id": "ccc",
        "title": [
          "solr is lucene"
        ],
        "_version_": 1604846150143180800
      },
      {
        "id": "bbb",
        "title": [
          "solr is java"
        ],
        "_version_": 1604846138631913500
      }
    ],
    "highlighting": {
      "ccc": {},
      "bbb": {}
    }
  }
}
                
```

4.3) 配置中文分词器:

4.3.1) Schema.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。

\\solr\collection1\conf			
新建文件夹			
名称 ^	修改日期	类型	大小
clustering	2015/1/18 12:56	文件夹	
lang	2015/1/18 12:56	文件夹	
velocity	2015/1/18 12:56	文件夹	
xslt	2015/1/18 12:56	文件夹	
_rest_managed.json	2014/12/10 0:37	JSON 文件	1 KB
_schema_analysis_stopwords_english...	2014/12/10 0:37	JSON 文件	1 KB
_schema_analysis_synonyms_english...	2014/12/10 0:37	JSON 文件	1 KB
admin-extra.html	2014/12/10 0:37	360 se HTML Do...	2 KB
admin-extra.menu-bottom.html	2014/12/10 0:37	360 se HTML Do...	1 KB
admin-extra.menu-top.html	2014/12/10 0:37	360 se HTML Do...	1 KB
currency.xml	2014/12/10 0:37	XML 文档	4 KB
elevate.xml	2014/12/10 0:37	XML 文档	2 KB
mapping-FoldToASCII.txt	2014/12/10 0:37	文本文档	81 KB
mapping-ISOLatinIAccent.txt	2014/12/10 0:37	文本文档	4 KB
protwords.txt	2014/12/10 0:37	文本文档	1 KB
schema.xml	2015/1/18 14:01	XML 文档	61 KB
scripts.conf	2014/12/1 10:06	CONF 文件	1 KB
solrconfig.xml	2015/1/18 13:39	XML 文档	75 KB
spellings.txt	2014/12/10 0:37	文本文档	1 KB
stopwords.txt	2014/12/10 0:37	文本文档	1 KB
synonyms.txt	2014/12/10 0:37	文本文档	2 KB
update-script.js	2014/12/10 0:37	JScript Script...	2 KB

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 name="text_general" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <!-- in this example, we will only use synonyms at query time
    <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true"
    expand="false"/>
    -->
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>

```

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, field 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:

```

"response": {
  "numFound": 5,
  "start": 0,
  "docs": [
    {
      "id": "SOLR1000",
      "name": "Solr, the Enterprise Search Server",
      "manu": "Apache Software Foundation",
      "cat": [
        "software",
        "search"
      ],
      "features": [
        "Advanced Full-Text Search Capabilities using Lucene",
        "Optimized for High Volume Web Traffic",
        "Standards Based Open Interfaces - XML and HTTP",
        "Comprehensive HTML Administration Interfaces",
        "Scalability - Efficient Replication to other Solr Search Servers",
        "Flexible and Adaptable with XML configuration and Schema",
        "Good unicode support: hello (hello with an accent over the e)"
      ]
    }
  ]
}

```

查询出来的结果是多个值

uniqueKey

Solr 中默认定义唯一主键 key 为 id 域，如下：

```
<uniqueKey>id</uniqueKey>
```

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。
注意在创建索引时必须指定唯一约束。

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：

```

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

```

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：

```

<copyField source="title" dest="text"/>
<copyField source="author" dest="text"/>
<copyField source="description" dest="text"/>
<copyField source="keywords" dest="text"/>
<copyField source="content" dest="text"/>

```

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 scheme.xml 中的 dynamicField 规则匹配成功，如下：



```
<dynamicField name="*_t" type="text_general" indexed="true" stored="true"/>
```

“product_title_t” 是以 “_t” 结尾。

创建索引：

Request-Handler (qt)

Document Type
JSON

Document(s)

```
{ "id": "100033", "product_title_t": "spring book" }
```

Commit Within

Overwrite

Boost

Status: success
Response:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 3
  }
}
```

搜索索引：

Request-Handler (qt)

common
q

fq

sort

start, rows

fl

df

Raw Query Parameters

<http://localhost:8080/solr/collection1/s>

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "q": "id:100033",
      "_": "1422355381499",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 1,
    "start": 0,
    "docs": [
      {
        "id": "100033",
        "product title t": "spring book",
        "_version_": 1491447668014055400
      }
    ]
  }
}
```

4.3.2) IK 分词器的安装步骤：（非常重要）

第一步：

软件 (D:) \ MySolr \ apache-tomcat-7.0.40 \ webapps \ solr \ WEB-INF \ lib

刻录 新建文件夹

第一步：添加中文分词器jar包复制到 solr/WEB-INF/lib 这个目录下

名称	修改日期	类型	大小
hadoop-hdfs-2.2.0.jar	2013/10/7 02:32	Executable Jar...	5,120 KB
hppc-0.5.2.jar	2013/7/9 03:55	Executable Jar...	1,259 KB
httpclient-4.3.1.jar	2013/10/3 15:42	Executable Jar...	572 KB
httpcore-4.3.jar	2013/7/29 11:06	Executable Jar...	276 KB
httpmime-4.3.1.jar	2013/10/3 15:43	Executable Jar...	37 KB
IKAnalyzer2012FF_ul.jar	2012/10/26 20:46	Executable Jar...	1,139 KB
jcl-over-slf4j-1.7.6.jar	2014/2/5 17:39	Executable Jar...	17 KB
joda-time-2.2.jar	2013/3/8 09:23	Executable Jar...	561 KB

第二步：

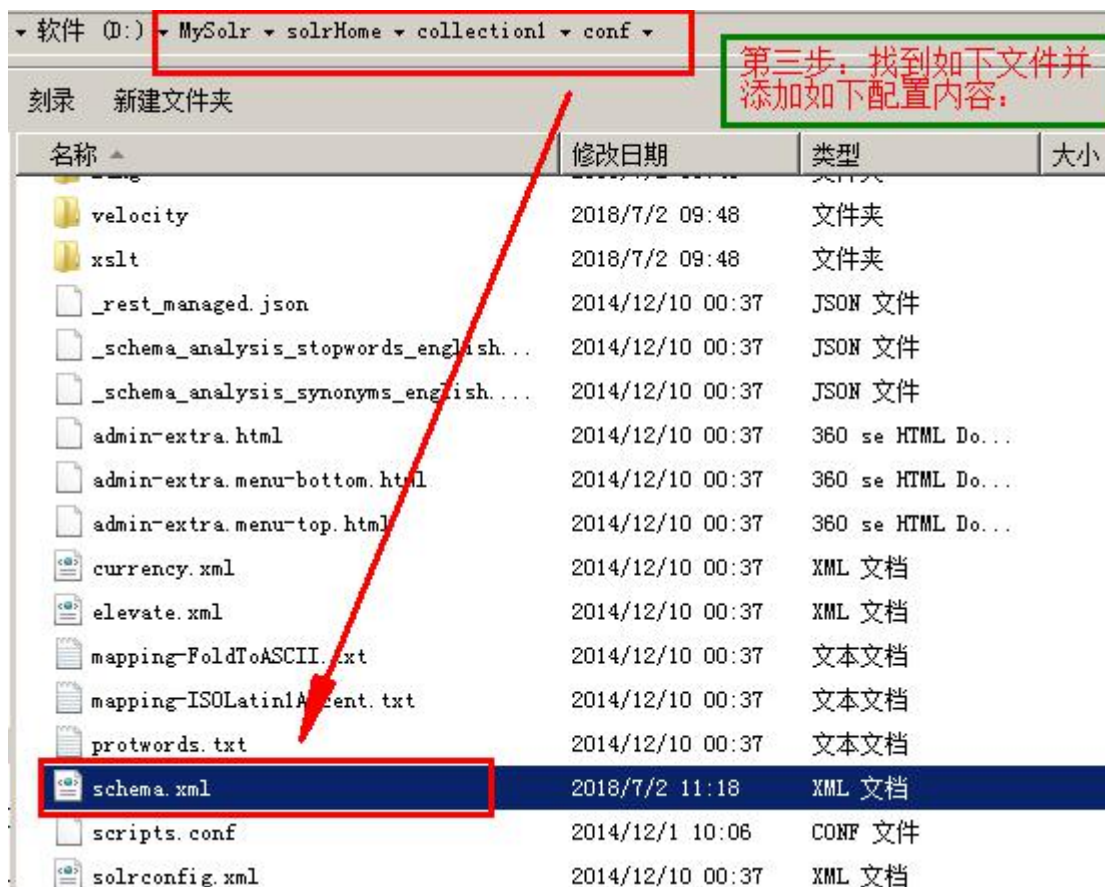
软件 (D:) \ MySolr \ apache-tomcat-7.0.40 \ webapps \ solr \ WEB-INF \ classes

共享 刻录 新建文件夹

在当前solr的/WEB-INF目录下新建classes目录，然后，再将IK分词器需要的配置文件及两个词典文件复制到其中，注意两个词典文件需要保存为无BOM的UTF-8格式（建议使用EditPlus保存）

名称	修改日期	类型
ext.dic	2018/6/29 11:24	DIC 文件
IKAnalyzer.cfg.xml	2018/6/29 11:10	XML 文档
stopword.dic	2018/6/29 11:23	DIC 文件

第三步：



<!-- IKAnalyzer（定义字段类型）-->

<fieldType name="text_ik" class="solr.TextField">

<analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>

</fieldType>

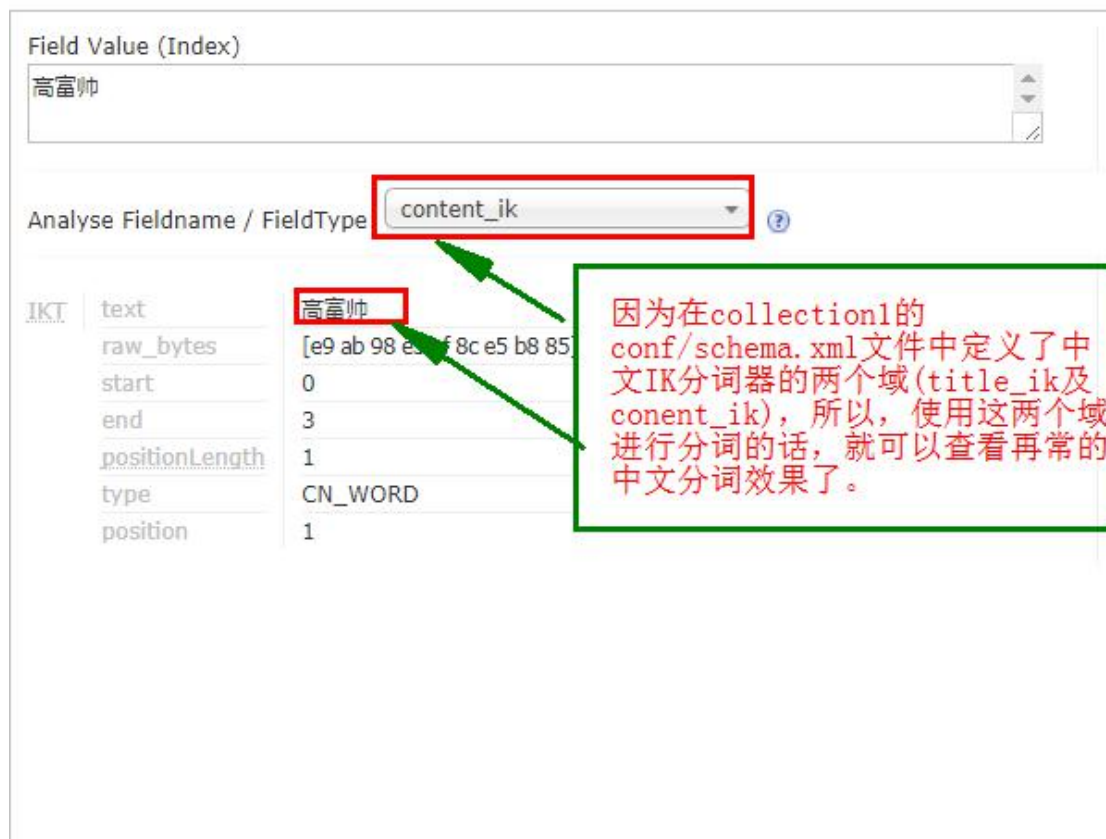
第四步：

<!--使用分词器-->

<field name="title_ik" type="text_ik" indexed="true" stored="true" />

<field name="content_ik" type="text_ik" indexed="true" stored="false" multiValued="true"/>

第五步：测试



Field Value (Index)

高富帅

Analyse Fieldname / FieldType **content_ik**

IKT	text	高富帅
raw_bytes	[e9 ab 98 e5 f8 c5 b8 85]	
start	0	
end	3	
positionLength	1	
type	CN_WORD	
position	1	

因为在collection1的conf/schema.xml文件中定义了中文IK分词器的两个域(title_ik及content_ik)，所以，使用这两个域进行分词的话，就可以查看再常的中文分词效果了。

4.4) 批量导入数据功能配置与使用：

第一步：将如下三个包添加到 **solrHome/collection1/lib** 目录下：

 mysql-connector-java-5.1.7-bin.jar	2018/6/8 11:20	Executable Jar...	694 KB
 solr-dataimporthandler-4.10.3.jar	2014/12/10 00:34	Executable Jar...	215 KB
 solr-dataimporthandler-extras-4.10.3.jar	2014/12/10 00:34	Executable Jar...	37 KB

上面三个包的原始位置，后两个包放在：solr4.10.3/dist/下面：

ene@Solr资料 ▾ 资料 ▾ 01. 参考资料 ▾ solr-4.10.3 ▾ dist ▾

共享 ▾ 刻录 新建文件夹

名称 ▴	修改日期	类型	大小
 solrj-lib	2017/1/2 21:17	文件夹	
 test-framework	2017/1/2 21:17	文件夹	
 solr-4.10.3.war	2014/12/10 00:35	WAR 文件	29,045 KB
 solr-analysis-extras-4.10.3.jar	2014/12/10 00:34	Executable Jar...	18 KB
 solr-cell-4.10.3.jar	2014/12/10 00:34	Executable Jar...	30 KB
 solr-clustering-4.10.3.jar	2014/12/10 00:34	Executable Jar...	51 KB
 solr-core-4.10.3.jar	2014/12/10 00:35	Executable Jar...	2,786 KB
 solr-dataimporthandler-4.10.3.jar	2014/12/10 00:34	Executable Jar...	215 KB
 solr-dataimporthandler-extras-4.10...	2014/12/10 00:34	Executable Jar...	37 KB
 solr-langid-4.10.3.jar	2014/12/10 00:34	Executable Jar...	750 KB
 solr-map-reduce-4.10.3.jar	2014/12/10 00:35	Executable Jar...	127 KB
 solr-morphlines-cell-4.10.3.jar	2014/12/10 00:35	Executable Jar...	25 KB

第二步: 在 **solrHome/collection1/conf/solrConfig.xml** 文件, 在其中搜索 **requestHandler**, 并在第一个 **requestHandler** 前面添加如下代码:

<!--第一步: 配置批量数据导入的处理器配置-->

```
<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: 在上面的 **solrconfig.xml** 文件的同级目录下新建 **data-config.xml** 文件, 并添加如下内容:

```
<?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="123"/>
<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>
```

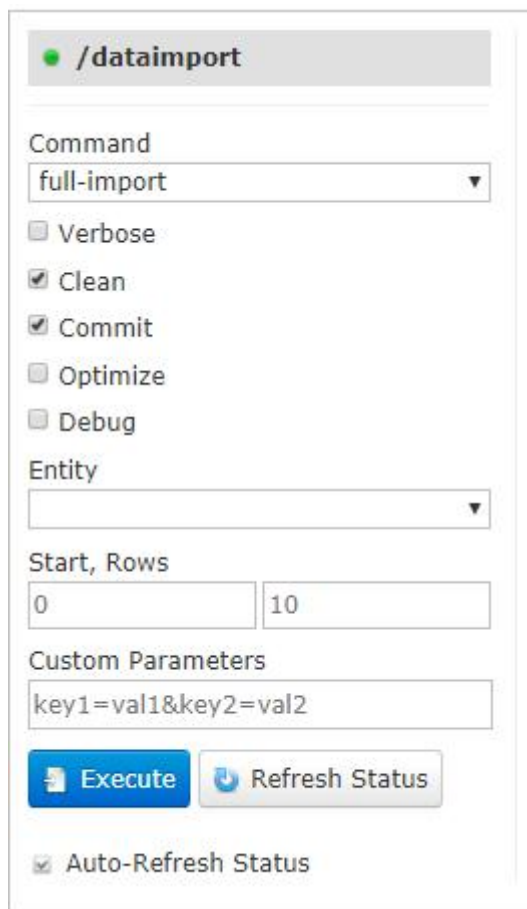
第四步：在 **solrHome/collection1/conf/schema.xml** 文件中添加对数据库相应的字段的配置：

<!--自定义与商品相关的域-->

```
<!--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"/>
<!--将商品描述及商品名称这两个域中的内容拷贝到 product_keywords 这个域中-->
<copyField source="product_name" dest="product_keywords"/>
<copyField source="product_description" dest="product_keywords"/>
```


第五步：重启 tomcat 服务器，看到如下效果：



The screenshot shows a web interface for data import. At the top, there's a header bar with a green dot and the text "/dataimport". Below this, there's a "Command" dropdown menu set to "full-import". There are several checkboxes: "Verbose" (unchecked), "Clean" (checked), "Commit" (checked), "Optimize" (unchecked), and "Debug" (unchecked). Below these is an "Entity" dropdown menu. Further down, there are two input fields for "Start, Rows" with values "0" and "10". A "Custom Parameters" text area contains "key1=val1&key2=val2". At the bottom, there are two buttons: "Execute" (blue) and "Refresh Status" (grey). Below the buttons is a checkbox for "Auto-Refresh Status" which is checked.

第六步：测试

测试一：删除测试：

删除索引格式如下：

1) 删除制定 ID 的索引

```
<delete>
```

```
  <id>8</id>
```

```
</delete>
```

```
<commit/>
```

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>
```

测试二：查询测试

Request-Handler (qt)

/select

— common —

q

,

fq

product_catalog_name:幽默杂货

sort

start, rows

0 10

fl

df

Raw Query Parameters

key1=val1&key2=val2

wt

json

☒ indent

☐ debugQuery

☐ dismax

☐ edismax

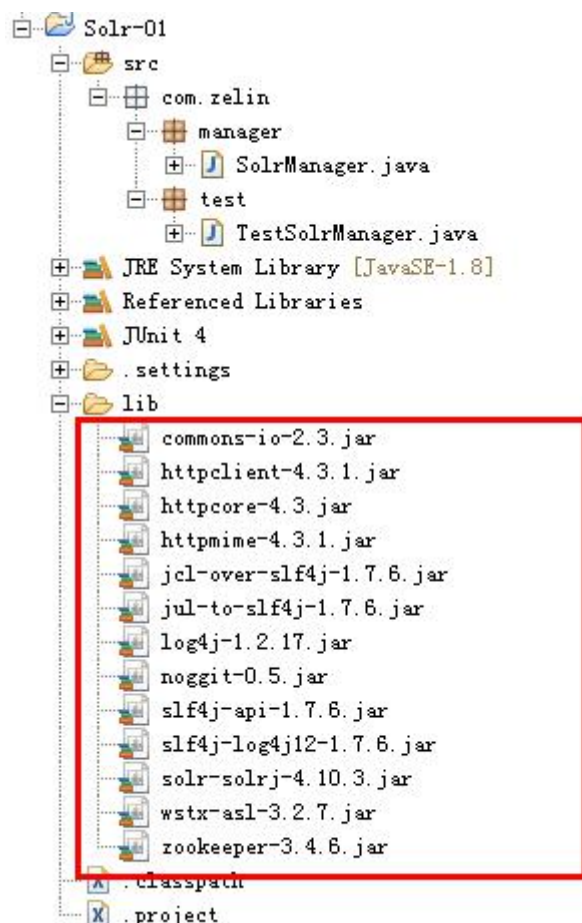
http://localhost:8080/solr/collection1/select?q=**&fq=product_catalog

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 18,
    "params": {
      "q": "**",
      "indent": "true",
      "fq": "product_catalog_name:幽默杂货",
      "wt": "json",
      "_": "1530514923511"
    }
  },
  "response": {
    "numFound": 220,
    "start": 0,
    "docs": [
      {
        "product_catalog_name": "幽默杂货",
        "product_price": 18.9,
        "product_name": "花儿朵朵彩色金属门后挂 8钩免钉门背挂钩2066",
        "id": "1",
        "product_picture": "2014032613103438.png",
        "_version_": 1604860506528546800
      },
      {
        "product_catalog_name": "幽默杂货",
        "product_price": 18.9,
        "product_name": "幸福一家人彩色金属门后挂 8钩免钉门背挂钩2088",
        "id": "2",
        "product_picture": "2014032612461139.png",

```

5、 Sorj 应用：

第一步：新建 Java 工程添加如下 jar 包：



上面的 jar 包的位置：

ene@Solr资料 ▾ 资料 ▾ 01. 参考资料 ▾ solr-4.10.3 ▾ example ▾ lib ▾ ext			
共享 ▾ 刻录 新建文件夹			
名称 ▴	修改日期	类型	大小
jcl-over-slf4j-1.7.6.jar	2014/2/5 17:39	Executable Jar...	17 KB
jul-to-slf4j-1.7.6.jar	2014/2/5 17:40	Executable Jar...	5 KB
log4j-1.2.17.jar	2012/5/26 05:43	Executable Jar...	479 KB
slf4j-api-1.7.6.jar	2014/2/5 17:37	Executable Jar...	29 KB
slf4j-log4j12-1.7.6.jar	2014/2/5 17:38	Executable Jar...	9 KB

Lucene@Solr资料 ▾ 资料 ▾ 01.参考资料 ▾ solr-4.10.3 ▾ dist ▾

共享 ▾ 刻录 新建文件夹

名称 ▴	修改日期
 solrj-lib	2017/1/2 21:17
 test-framework	2017/1/2 21:17
 solr-4.10.3.war	2014/12/10 00:35
 solr-analysis-extras-4.10.3.jar	2014/12/10 00:34
 solr-cell-4.10.3.jar	2014/12/10 00:34
 solr-clustering-4.10.3.jar	2014/12/10 00:34
 solr-core-4.10.3.jar	2014/12/10 00:35
 solr-dataimporthandler-4.10.3.jar	2014/12/10 00:34
 solr-dataimporthandler-extras-4.10.3.jar	2014/12/10 00:34
 solr-langid-4.10.3.jar	2014/12/10 00:34
 solr-map-reduce-4.10.3.jar	2014/12/10 00:35
 solr-morphlines-cell-4.10.3.jar	2014/12/10 00:35
 solr-morphlines-core-4.10.3.jar	2014/12/10 00:35
 solr-solrj-4.10.3.jar	2014/12/10 00:35
 solr-test-framework-4.10.3.jar	2014/12/10 00:34
 solr-uima-4.10.3.jar	2014/12/10 00:35
 solr-velocity-4.10.3.jar	2014/12/10 00:35

总共 13 个 jar 包。

第二步：添加到索引库：

//添加到索引库

```

public void addIndex() throws Exception{
    //1.构建一个 solrServer 对象
    SolrServer solrServer = new
HttpSolrServer("http://localhost:8080/solr");
    //2.构建一个文档对象
    SolrInputDocument doc = new SolrInputDocument();
    doc.setField("id", "10001");
    doc.setField("title", "java 编程实践");
    doc.setField("name", "java is good!");
    //3.添加文档到索引库
    solrServer.add(doc );
    //4.提交到索引库中
    solrServer.commit();
}
    
```

第三步：删除索引库：

```
//删除索引库

public void deleteInIndex() throws Exception{
    //1.构建一个 solrServer 对象
    SolrServer solrServer = new
HttpSolrServer("http://localhost:8080/solr");
    //2.根据 id 删除指定索引
    solrServer.deleteById("10001");
    //3.根据查询删除指定索引
    solrServer.deleteByQuery("product_name:幸福");
    solrServer.commit();
}
```

第四步：对索引库进行简单查询：

```
//进行简单查询
public void findBySimple() throws Exception{
    //1. 构造 solrServer
    SolrServer solrServer = new HttpSolrServer("http://localhost:8080/solr");
    //2. 定义 SolrParams 查询参数
    SolrQuery params = new SolrQuery();
    //3. 向查询参数中添加查询条件
    params.setQuery("*:~"); //查询所有
    params.addFilterQuery("product_name:浪漫"); //添加过滤查询
    params.addSort("product_price", SolrQuery.ORDER.desc); //添加排序查询
    //4. 根据查询参数进行查询
    QueryResponse queryResponse = solrServer.query(params);
    //5. 得到查询的结果集
    SolrDocumentList results = queryResponse.getResults();
    //6. 得到查询的总记录数
    long total = results.getNumFound();
    System.out.println("一共查询出" + total + "条记录!");
    //7. 打印查询内容
    for (SolrDocument doc : results) {
        showInfo(doc);
    }
    //8. 提交查询
    solrServer.commit();
}

//显示文档信息
private void showInfo(SolrDocument doc) {
```



```
String id = (String) doc.get("id");
//id
String product_name = (String) doc.get("product_name");
//product_name
float product_price =
Float.parseFloat(doc.get("product_price").toString()); //product_price
String product_category_name = (String) doc.get("product_catalog_name");
//product_catalog_name
String product_picture = (String) doc.get("product_picture");
//product_picture
System.out.println("id:" + id);
System.out.println("product_name:" + product_name);
System.out.println("product_price:" + product_price);
System.out.println("product_category_name:" + product_category_name);
System.out.println("product_picture:" + product_picture);

System.out.println("-----");
}
}
```

第五步：对索引库进行复杂查询：

```
//进行复杂查询
public void queryByFuza() throws Exception{
    //1. 构造 solrServer
    SolrServer solrServer = new HttpSolrServer("http://localhost:8080/solr");
    //2. 定义 SolrParams 查询参数
    SolrQuery params = new SolrQuery();
    //3. 向查询参数中添加查询条件
    params.set("q", "浪漫"); //添加查询
    //4. 设置默认查询字段
    params.set("df", "product_name", "product_catalog_name");
    //5. 设置查询的字段列表

    params.setFields("id", "product_name", "product_catalog_name", "product_price", "product_picture");

    //6. 添加过滤查询
    params.addFilterQuery("product_catalog_name:幽默杂货");
    //7. 添加排序查询
```



```
params.addSort("product_price", SolrQuery.ORDER.desc);

//8. 设置高亮查询
params.setHighlight(true); //可以进行高亮查询
params.addHighlightField("product_name"); //添加高亮查询的字段
params.setHighlightSimplePre("<span style='color:red'>"); //设置高亮查询的前缀
params.setHighlightSimplePost("</span>"); //设置高亮查询的后缀

//9. 设置开始分页
params.setStart(0); //从第 0 条记录开始查询
params.setRows(10); //每次显示 10 条

//10. 开始查询
QueryResponse queryResponse = solrServer.query(params);
//11. 得到查询结果
SolrDocumentList results = queryResponse.getResults();
//12. 得到高亮查询结果
Map<String, Map<String, List<String>>> highlighting =
queryResponse.getHighlighting();
//13. 遍历查询结果
for (SolrDocument doc : results) {
    //① 打印文档信息
    showInfo(doc);
    //② 根据文档 id 得到某条高亮数据
    Map<String, List<String>> maps = highlighting.get(doc.get("id").toString());
    //③ 根据存放到高亮中的字段取出此字段的内容
    List<String> list = maps.get("product_name");
    //④ 打印取出的数据
    if (list != null && list.size() > 0) {
        System.out.println("商品名称[高亮]: " + list.get(0));
    }
}
//14. 提交查询
solrServer.commit();
}
```

第六步：整体测试查询：

```
public class TestSolrManager {
    private SolrManager solrManager;
    @Before
    public void init(){
```

```

        solrManager = new SolrManager();
    }
    //测试添加到索引库
    @Test
    public void testSimpleSearch() throws Exception{
        solrManager.addIndex();
    }
    //测试删除
    @Test
    public void testDeleteInIndex() throws Exception{
        solrManager.deleteInIndex();
        System.out.println("删除索引成功!");
    }
    //测试简单查询
    @Test
    public void testQueryIndex1() throws Exception{
        solrManager.queryIndex1();
    }
    //测试复杂查询
    @Test
    public void testQueryIndex2() throws Exception{
        solrManager.queryIndex2();
    }
}
    
```

第七步：运行效果如下：

```

id:71
product_name:家天下音乐之城纯净水晶般大钢琴八音盒镀金机芯YKL-09&nbsp;创意浪漫礼品
product_price:60.0
product_category_name:幽默杂货
product_picture:2012091510441826.jpg
    
```

商品名称[高亮]:家天下音乐之城纯净水晶般大钢琴八音盒镀金机芯YKL-09 创意浪漫礼品

```

id:69
product_name:家天下音乐之城纯净水晶般大提琴八音盒镀金机芯YKL-11&nbsp;创意浪漫礼品
product_price:58.0
product_category_name:幽默杂货
product_picture:2012091511161729.jpg
    
```

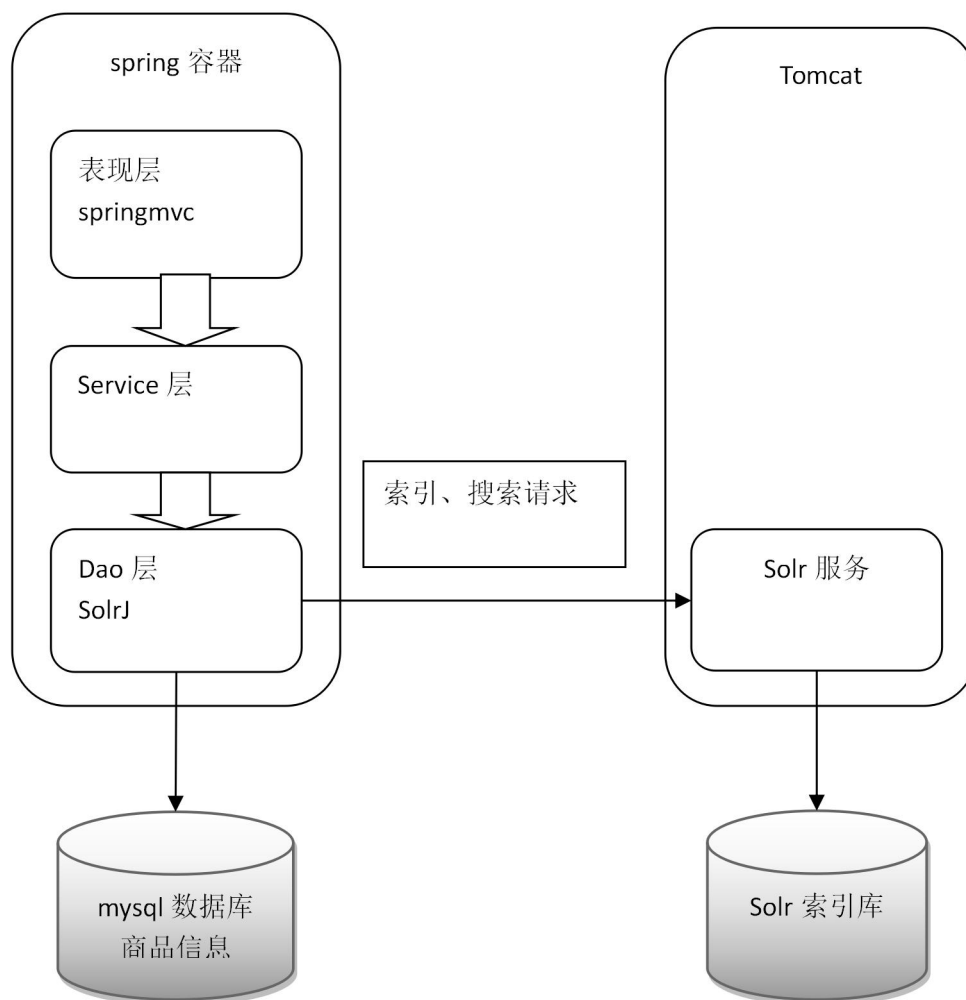
商品名称[高亮]:家天下音乐之城纯净水晶般大提琴八音盒镀金机芯YKL-11 创意浪漫礼品

```

id:67
product_name:音乐之城亚克力水晶浪漫大心形八音盒音乐盒YKL-14&nbsp;创意浪漫礼品
    
```


6、 Solor 整合 SpringMVC-案例:京东商城







整合工作原理图如下：



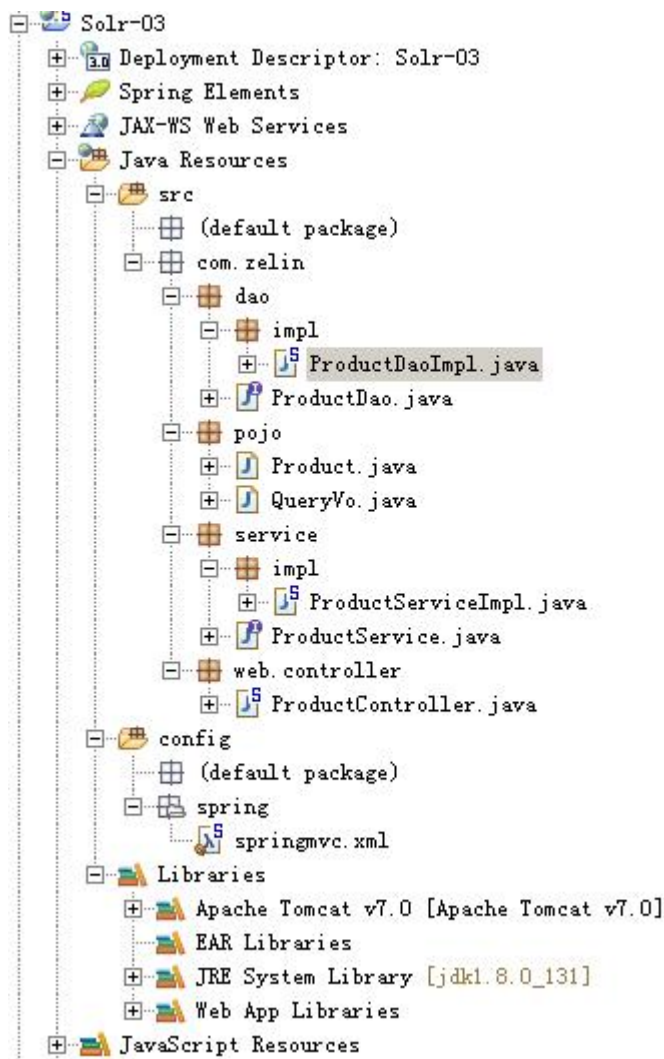
说明：当我们在当前的 web 应用程序(左图)中利用 solrj 请求索引服务时，solrj 会请求在另一台服务器上的 solr 服务，从而访问其索引库。注意，现在的 web 应用程序不再访问数据库，代替访问的是索引库。

第一步：添加上面 Sorj 的 13 个 jar 包，再添加 springmvc 的相关 jar 包：

zenas@Solr资料 ▾ 资料 ▾ 01.参考资料 ▾ Spring4.1.3包含mvc

名称 ▴	修改日期	类型	大小
 aopalliance-1.0.jar	2016/5/24 11:28	Executable Jar...	5 KB
 asm-3.3.1.jar	2016/5/24 11:28	Executable Jar...	43 KB
 aspectjweaver-1.6.11.jar	2016/5/24 11:29	Executable Jar...	1,651 KB
 cglib-2.2.2.jar	2016/5/24 11:28	Executable Jar...	281 KB
 commons-dbcp-1.2.2.jar	2016/5/24 11:28	Executable Jar...	119 KB
 commons-fileupload-1.2.2.jar	2016/5/24 11:28	Executable Jar...	59 KB
 commons-io-2.4.jar	2016/5/24 11:28	Executable Jar...	181 KB
 commons-logging-1.1.1.jar	2016/5/24 11:28	Executable Jar...	60 KB
 commons-pool-1.3.jar	2016/5/24 11:28	Executable Jar...	61 KB
 jackson-annotations-2.4.0.jar	2016/5/24 11:28	Executable Jar...	38 KB
 jackson-core-2.4.2.jar	2016/5/24 11:29	Executable Jar...	221 KB
 jackson-databind-2.4.2.jar	2016/5/24 11:29	Executable Jar...	1,051 KB
 javassist-3.17.1-GA.jar	2016/5/24 11:29	Executable Jar...	696 KB
 jstl-1.2.jar	2016/5/24 11:29	Executable Jar...	405 KB
 junit-4.9.jar	2016/5/24 11:29	Executable Jar...	244 KB
 log4j-1.2.17.jar	2016/5/24 11:29	Executable Jar...	479 KB
 log4j-api-2.0-rc1.jar	2016/5/24 11:29	Executable Jar...	114 KB
 log4j-core-2.0-rc1.jar	2016/5/24 11:29	Executable Jar...	687 KB
 mybatis-3.2.7.jar	2016/5/24 11:30	Executable Jar...	697 KB
 mybatis-spring-1.2.2.jar	2016/5/24 11:29	Executable Jar...	48 KB
 mysql-connector-java-5.1.7-bin.jar	2016/5/24 11:30	Executable Jar...	694 KB
 slf4j-api-1.7.5.jar	2016/5/24 11:29	Executable Jar...	26 KB
 slf4j-log4j12-1.7.5.jar	2016/5/24 11:29	Executable Jar...	9 KB
 spring-aop-4.1.3.RELEASE.jar	2016/5/24 11:29	Executable Jar...	351 KB
 spring-aspects-4.1.3.RELEASE.jar	2016/5/24 11:29	Executable Jar...	56 KB

项目结构如下：



第二步：修改 web.xml，添加 springmvc 的 DispatcherServlet:

```
<!-- 只定义 springmvc 的配置 -->
<display-name>Solr-2</display-name>
<servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springmvc.xml</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
```



```
<url-pattern>*.action</url-pattern>
</servlet-mapping>
<!-- 解决 post 乱码问题 -->
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter
</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

第三步：在类路径 config 文件夹下添加 springmvc.xml 文件：

```
<context:component-scan base-package="com.zelin"/>
<!-- 配置 springmvc 的三大组件 -->
<mvc:annotation-driven/>
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolve
r">
    <property name="prefix" value="/WEB-INF/jsp/">
    <property name="suffix" value=".jsp"/>
</bean>
<!-- 配置 solrServer 对象 -->
<bean id="solrServer"
class="org.apache.solr.client.solrj.impl.HttpSolrServer">
    <constructor-arg value="http://localhost:8080/solr"/>
</bean>
```

第四步：根据 product_list.jsp 页面分析出实体类如下：

```
public class Product {

    private String id;
    private String name;
    private String picture;
    private float price;
```



```
public class QueryVo {  
  
    private String queryString;  
    private String catalog_name;  
    private String price;  
    private String sort;  
}
```

第五步: 定义 ProductDaoImpl.java 类如下:

```
@Repository  
public class ProductDaoImpl implements ProductDao {  
    @Autowired  
    private SolrServer solrServer;  
    @Override  
    public List<Product> getProductsByQueryVo(QueryVo queryVo) throws  
Exception {  
        //1.构造一个查询器  
        SolrQuery query = new SolrQuery();  
        //2.判断查询的文本框中的字符串是否为 null, 如果为 null, 查询所有  
        if(queryVo.getQueryString() == null ||  
        "".equals(queryVo.getQueryString())){  
            query.set("q", "*:*");  
        }else{  
            query.set("q", "product_keywords:" + queryVo.getQueryString());  
        }  
        //3.判断分类 product_catalog_name  
        if(queryVo.getCatalog_name() != null  
        && !"".equals(queryVo.getCatalog_name())){  
            query.set("fq",  
            "product_catalog_name:" + queryVo.getCatalog_name());  
        }  
        //4.判断价格区间  
        if(queryVo.getPrice() != null && !"".equals(queryVo.getPrice())){  
            //拆分价格区间  
            String[] split = queryVo.getPrice().split("-");  
            query.set("fq", "product_price:[" + split[0] + " TO " + split[1] +  
            "]" );  
        }  
        //5.对排序的判断  
        if(null!=queryVo.getSort() && queryVo.getSort().equals("1")){  
            query.addSort("product_price", ORDER.desc);  
        }else{  

```



```

        query.addSort("product_price",ORDER.asc);
    }
    //6.分页处理
    query.setStart(0);
    query.setRows(16);           //每页显示 16 条
    //7.高亮处理
    query.setHighlight(true);           //处理高亮
    query.setHighlightSimplePre("<span style='color:red'>"); //处理高亮
前缀
    query.setHighlightSimplePost("</span>"); //处理高亮后缀
    query.addHighlightField("product_name"); //添加高亮字段

    //8.开始查询
    QueryResponse response = solrServer.query(query);
    SolrDocumentList results = response.getResults();//查询出所有的结果集
    //9.查询出高亮字段的的结果集
    Map<String, Map<String, List<String>>> highlighting =
response.getHighlighting();

    //定义要存放查询结果的列表集合
    List<Product> products = new ArrayList<>();
    //10.开始遍历(所有的结果集)
    for (SolrDocument doc : results) {
        Product product = new Product();
        //10.1) 取得商品的 id
        String id = (String) doc.get("id");
        //10.2) 取得商品的图片
        String picture = (String) doc.get("product_picture");
        //10.3) 取得商品的价格
        Float price = (Float) doc.get("product_price");
        //10.4) 根据是否在高亮中有商品名称取得 product_name 域的值
        //◎ 根据 id 取得高亮集合中某条数据
        Map<String, List<String>> map = highlighting.get(id);
        //◎ 根据 product_name 取得商品名称的值,
        List<String> pnames = map.get("product_name");
        String name = null;
        //◎ 判断这里的 pnames 是否有值, 有值就放到 product 对象中
        if(pnames != null && pnames.size() > 0){
            name = pnames.get(0);
        }else{
            name = (String) doc.get("product_name");
        }
        //10.5)为 product 的每个属性赋值
        product.setId(id);
    }

```

```
        product.setName(name);
        product.setPicture(picture);
        product.setPrice(price);
        //10.6)将 product 添加到集合 products 中
        products.add(product);
    }
    return products;
}
```

第六步：定义控制器 ProductController 层：

```
@Controller
@RequestMapping("product")
public class ProductController {

    @Autowired
    private ProductService productService;
    @RequestMapping("list")
    public String list(Model model, QueryVo queryVo) throws Exception{
        System.out.println("queryVo-----" + queryVo);
        List<Product> products =
productService.getProductsByQueryVo(queryVo);
        model.addAttribute("products",products);
        //回显数据
        model.addAttribute("queryString",queryVo.getQueryString());
        model.addAttribute("catalog_name",queryVo.getCatalog_name());
        model.addAttribute("price",queryVo.getPrice());
        model.addAttribute("sort",queryVo.getSort());
        return "product_list";
    }
}
```

第七步：修改 JSP 页面：

```
<script type="text/javascript">
    function query() {
        //执行关键词查询时清空过滤条件
        document.getElementById("catalog_name").value="";
        document.getElementById("price").value="";
        document.getElementById("page").value="";
        //执行查询
        queryList();
    }
}
```




```
function queryList() {
    //提交表单
    document.getElementById("actionForm").submit();
}
function filter(key, value) {
    document.getElementById(key).value=value;
    queryList();
}
function sort() {
    var s = document.getElementById("sort").value;
    if (s != "1") {
        s = "1";
    } else {
        s = "0";
    }
    document.getElementById("sort").value = s;
    queryList();
}
function changePage(p) {
    var curpage = Number(document.getElementById("page").value);
    curpage = curpage + p;
    document.getElementById("page").value = curpage;
    queryList();
}
</script>
<div class="w" id="header-2013">
    <div id="Logo-2013" class="ld"><a href="http://www.jd.com/"
hidefocus="true"><b></b></a></div>
    <!--logo end-->
    <div id="search-2013">
        <div class="i-search ld">
            <ul id="shelper" class="hide"></ul>
            <form id="actionForm"
action="${pageContext.request.contextPath }/product/List" method="POST">
                <div class="form">
                    <input type="text" class="text" accesskey="s" name="queryString"
id="key" value="${queryString }"
                        autocomplete="off"
onkeydown="javascript:if(event.keyCode==13) {query()}">
                    <input type="button" value="搜索" class="button"
onclick="query()">
                </div>
                <input type="hidden" name="catalog_name" id="catalog_name"

```



```
value="${catalog_name }"/>
    <input type="hidden" name="price" id="price" value="${price }"/>
    <input type="hidden" name="sort" id="sort" value="${sort }"/>
    </form>
</div>
<div id="hotwords"></div>
</div>
<!--search end-->
<!-- 商品列表开始-->
<div id="plist" class="m plist-n7 plist-n8 prebuy">
    <ul class="list-h">
        <c:forEach var="item" items="${products }">
            <li pid="${item.id }">
                <div class="lh-wrap">
                    <div class="p-img">
                        <a target="_blank" href="#">
                            
                        </a>
                    </div>
                    <div class="p-name">
                        <a target="_blank" href="#">${item.name }</a>
                    </div>
                    <div class="p-price">
                        <strong>¥<fmt:formatNumber value="${item.price}"
maxFractionDigits="2"/></strong><span id="p1269191543"></span>
                    </div>
                </div>
            </li>
        </c:forEach>
    </ul>
</div>
<!-- 商品列表结束-->
```

第八步：运行效果如下：

