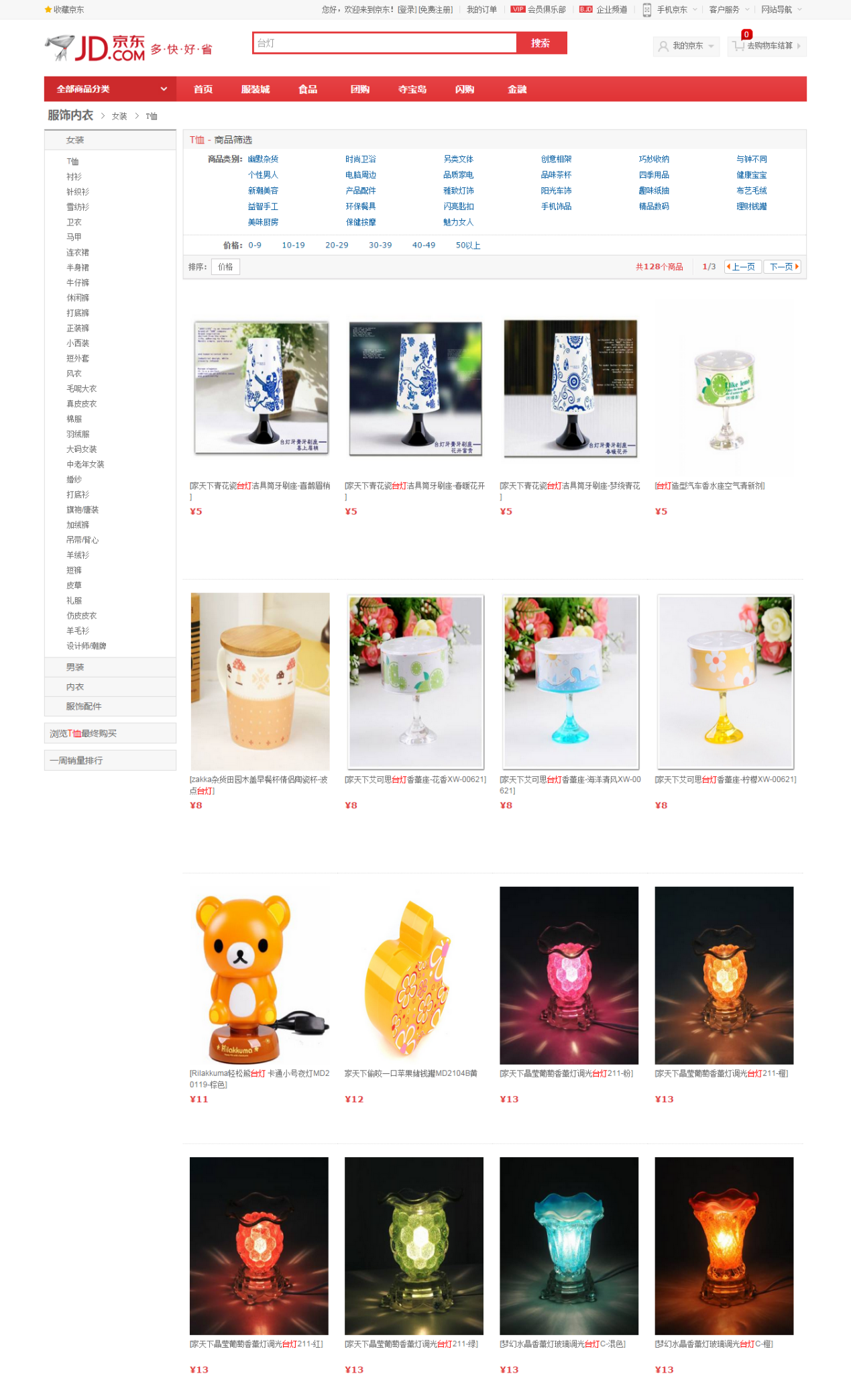
全文搜索技术-Solr

# 今天内容安排

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。



# 需求分析

## 实现方法

### 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。

### 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

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是一个**全文检索引擎系统**,部署到tomcat下就可以独立运行,通过http协议对外提供全文检索服务。

## 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，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

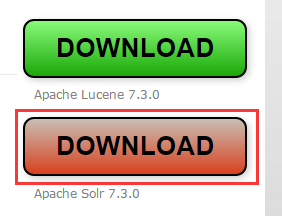
**Solr**的目标是打造一款企业级的搜索引擎系统，它是一个搜索引擎服务，可以独立运行，通过Solr可以非常快速的构建企业的搜索引擎，通过Solr也可以高效的完成站内搜索功能。



# Solr安装及配置

## Solr的下载

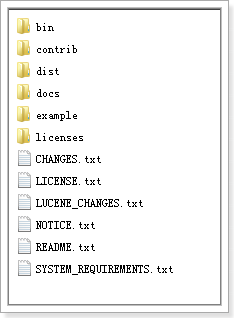
从Solr官方网站（<http://lucene.apache.org/> ）[下载](http://archive.apache.org/dist/lucene/solr/)Solr4.10.3



根据Solr的运行环境，Linux下需要下载solr-4.10.3.tgz，windows下需要下载solr-4.10.3.zip。

## Solr的文件夹结构

将solr-4.10.3.zip解压：



bin：solr的运行脚本

contrib：solr的一些软件/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

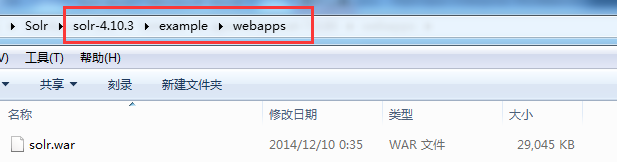
**example**：solr工程的例子目录：

licenses：solr相关的一些许可信息

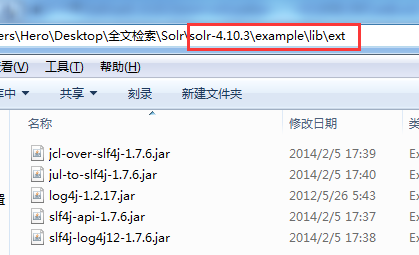
## Solr整合tomcat

### 整合步骤：

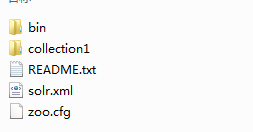
1. 把solr的war包复制到tomcat 的webapp目录下，**解压之后删除solr.war**



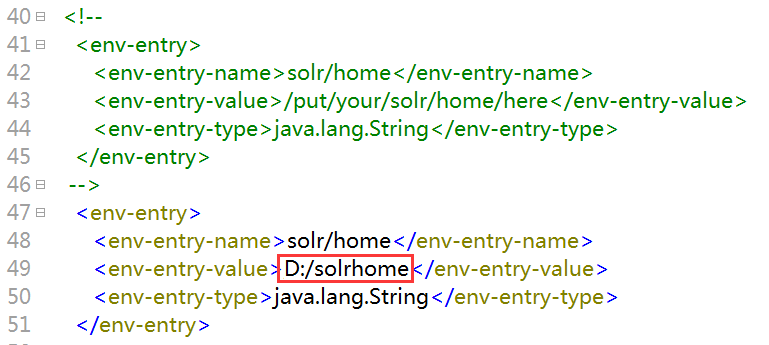
1. 给solr添加日志jar包

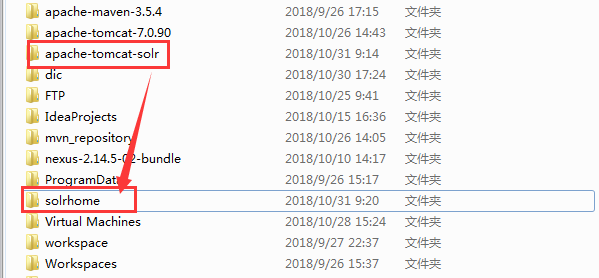


1. **配置solrHome和solrCore：**
   1. 把\solr-4.10.3\example\solr文件夹复制到D:\路径下，改名为solrhome，改名不是必须的，是为了便于理解。
   2. 在solrhome下有一个文件夹叫做collection1这就是一个solrcore。就是一个solr的实例。一个solrcore相当于mysql中一个仓库。

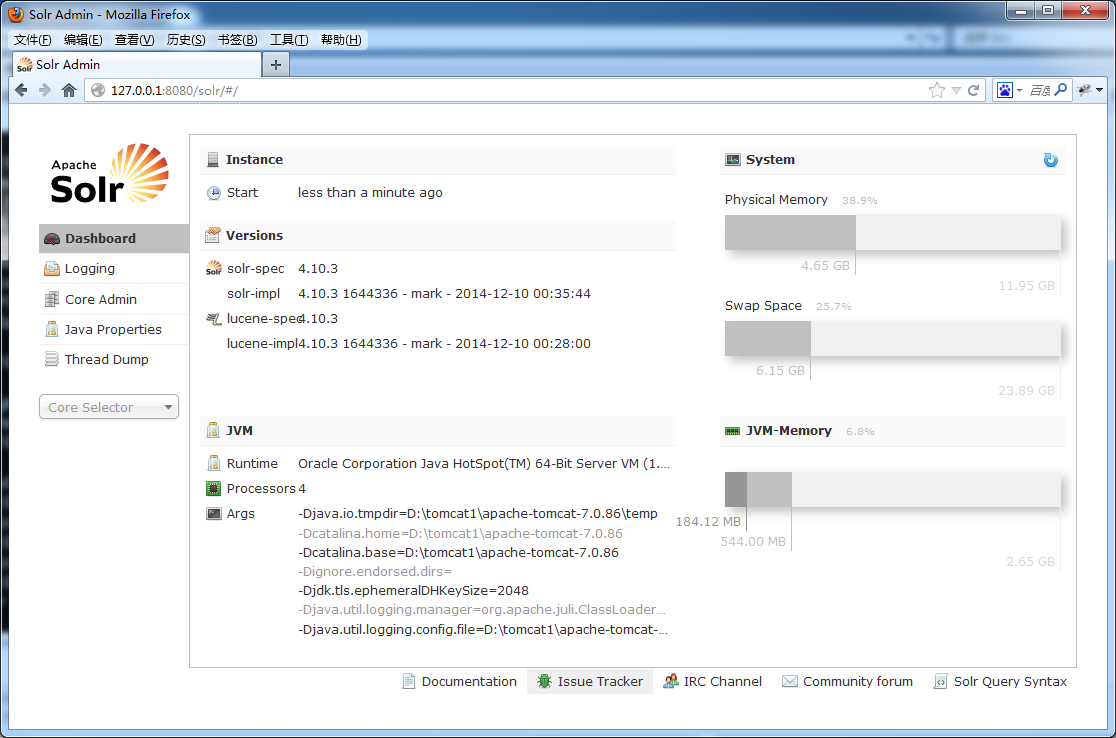


1. **告诉服务器solrHome的位置**：修改tomcat 的webapp目录下solr的web.xml配置文件





1. 启动tomcat，访问<http://localhost:8080/solr/>

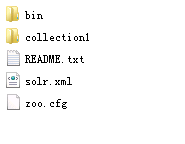


### Solr Home与SolrCore

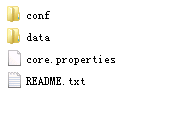
**solrHome**：SolrHome是Solr运行的主目录, 目录中包括了运行Solr实例所有的配置文件和数据文件

**solrCore**：一个solrCore就是一个solr实例, 一个SolrHome可以包括多个SolrCore，每个SolrCore提供单独的搜索和索引服务

1、example\solr是一个solr home目录结构，如下：



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



说明：

默认实例叫做collection1

conf是SolrCore的配置文件目录 。

data目录存放索引文件

## Solr后台管理

### 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

### 2、Logging

Solr运行日志信息

### 3、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。

添加solrcore：

第一步：复制collection1改名为collection2

第二步：修改core.properties。name=collection2

第三步：重启tomcat

### 4、java properties

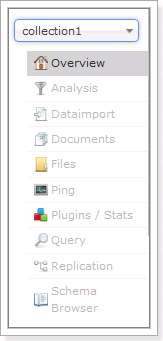
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

### 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。

### 6、Core selector

选择一个SolrCore进行详细操作，如下：



#### Analysis



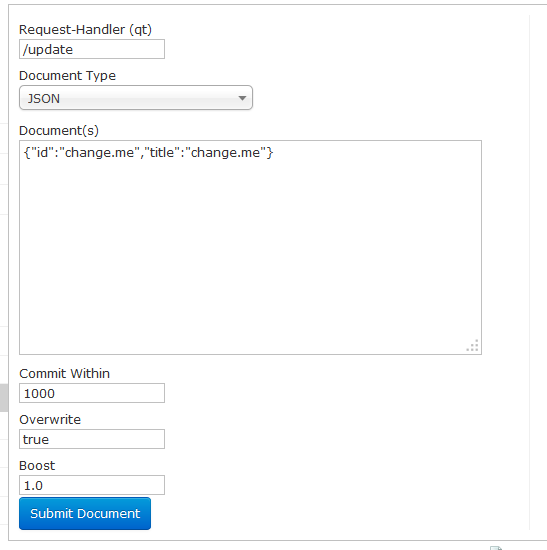
通过此界面可以测试索引分析器和搜索分析器的执行情况。

#### 6.2、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。

#### 6.3、Documents

通过此菜单可以创建索引、更新索引、删除索引等操作，



注意：

1. 索引的增加和修改必须要有id, 主键域,没有会报错



1. 域名必须先定义后使用,如果没有定义就使用会报错



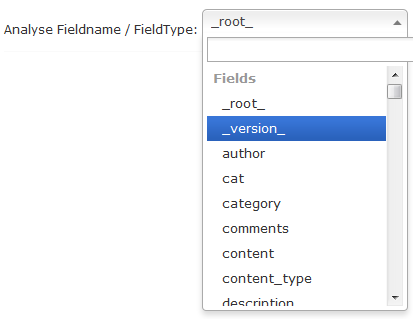
#### 6.4、Query

通过/select执行搜索索引，必须指定“q”查询条件方可搜索。

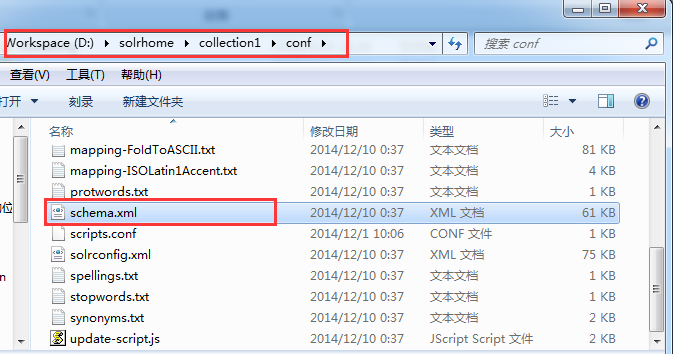


## 配置中文分析器

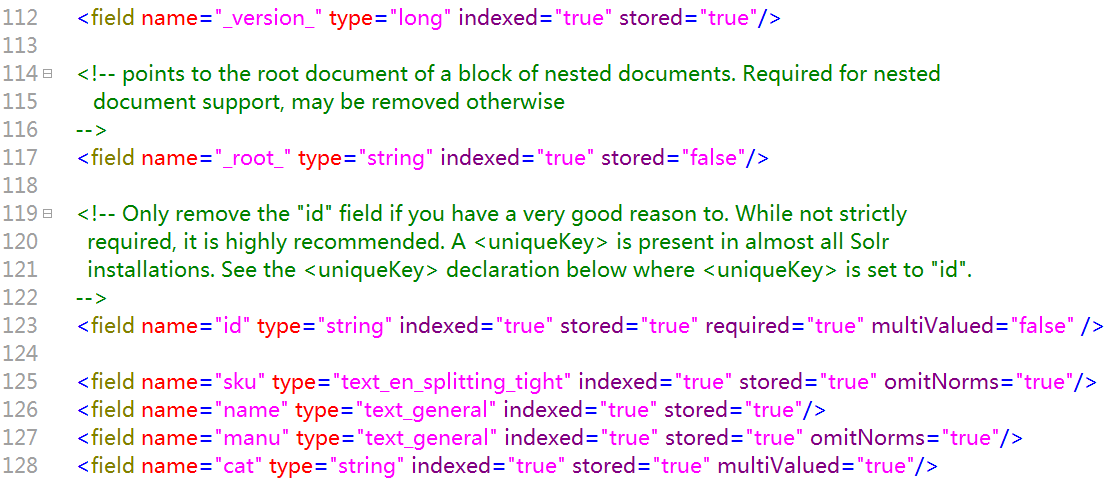
### field的类型（重点）



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。



#### 普通域



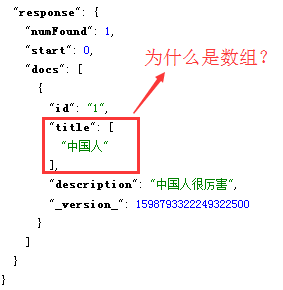
练习一：

1. 创建索引：{"id":"1","title":"中国人","content":"中国人很厉害"}
2. 查询索引：context:中国人

练习二：

1. 创建索引：{"id":"2","title":"中国人"," description":"中国人很厉害"}
2. 查询索引：description:中国人

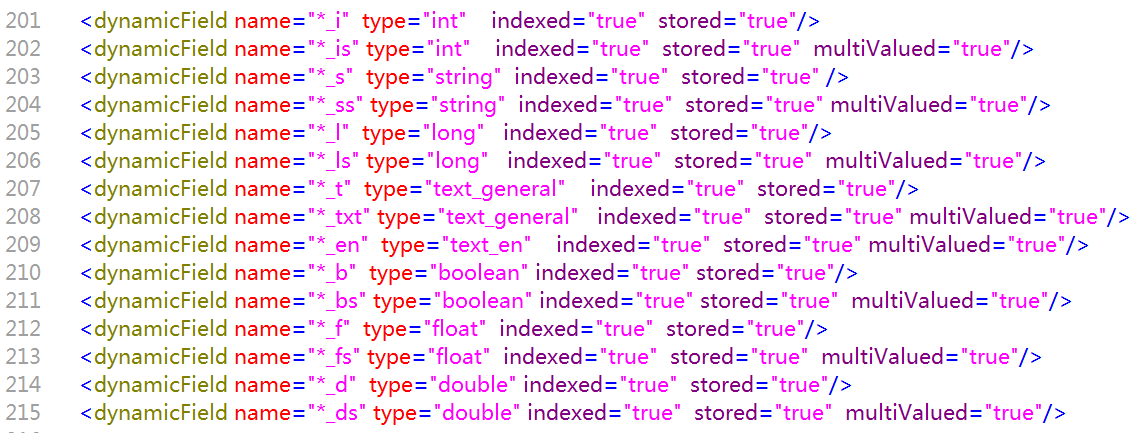
思考：



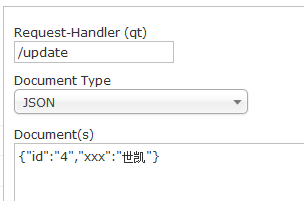
#### 动态域

动态域：起到模糊匹配的效果,可以模糊匹配没有定义过的域名

例如:xxxx这个域名没有定义,但是xxxx\_s这个域名模糊匹配了\*\_s这个域,所以相当于xxxx\_s这个域定义了

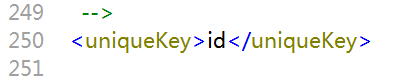


练习：



#### 主键域

一般主键域就用默认的这个就可以不需要更改或者添加



#### 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**和**内容content**，

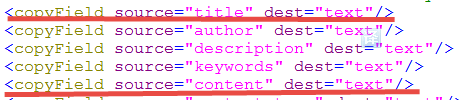
定义title、content、text的域：







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：

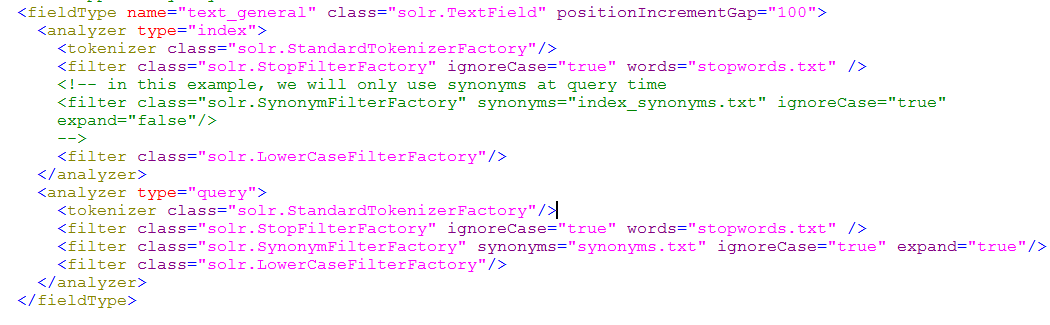


练习：



### 域类型定义(FieldType)

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：

1. name：是这个FieldType的名称
2. class：是Solr提供的包solr.TextField
3. 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

例如:搜索big car,如果document中存的是big red car,就无法搜索到了, positionIncrementGap就是设置big和car中间最大的间隔距离,只要在距离内就能搜索到.

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

例如上图中：

**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。

### 安装中文分词器

#### 使用IKAnalyzer中文分析器步骤：

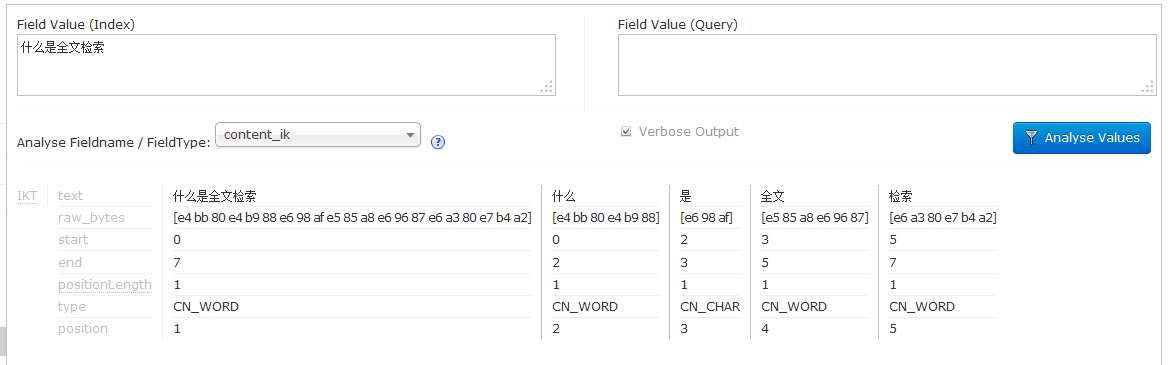
1. 把IKAnalyzer2012FF\_u1.jar添加到solr/WEB-INF/lib目录下。
2. 复制IKAnalyzer的配置文件和自定义词典和停用词词典到solr的WEB-INF目录下新建classes。
3. 在schema.xml中添加一个自定义的fieldType，使用中文分析器。

|  |
| --- |
| <!-- IKAnalyzer-->  <fieldType name="text\_ik" class="solr.TextField">  <analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>  </fieldType> |

1. 自定义field，指定field的type属性为text\_ik

|  |
| --- |
| <!--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. 重启tomcat测试：





练习一：

1. 创建索引：{"id":"2","title\_ik":"岛国","content\_ik":"岛国拍电影很厉害"}
2. 查询索引：content\_ik:电影

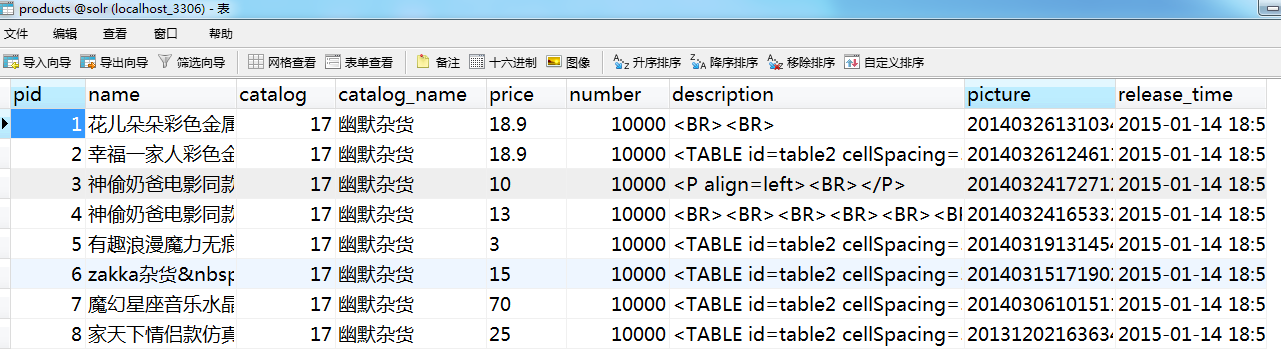
思考：



#### 总结：

* 1. **索引后就能查询,不索引就不能根据这个域搜索,**
  2. **存储后就能取出来里面的内容,不存储就取不出这个域内容**

## 分析业务数据并设置业务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" />

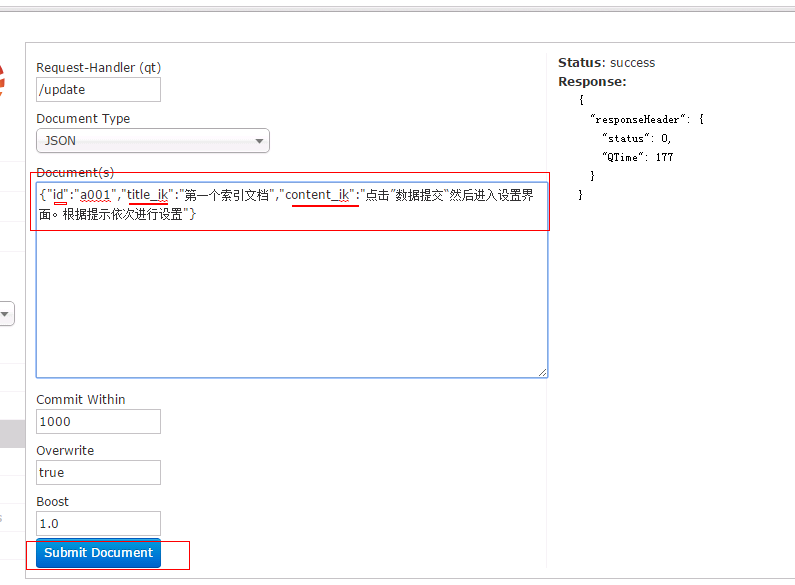
<field name="product\_keywords" type="text\_ik" indexed="true" stored="false" multiValued="true"/>

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

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

## 维护索引

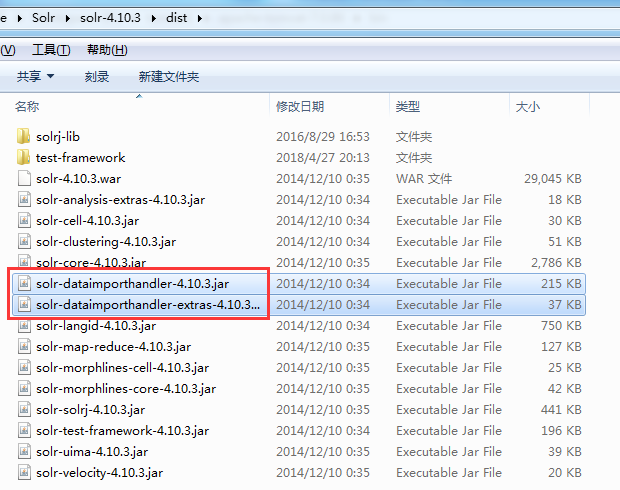
### 添加/更新单个文档



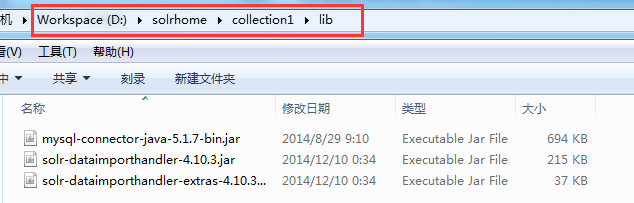
### 批量导入数据

使用dataimport插件批量导入数据。

第一步：把dataimport插件依赖的jar包添加到collection1\lib(新建)中还有数据库的包



还需要mysql的数据库驱动。



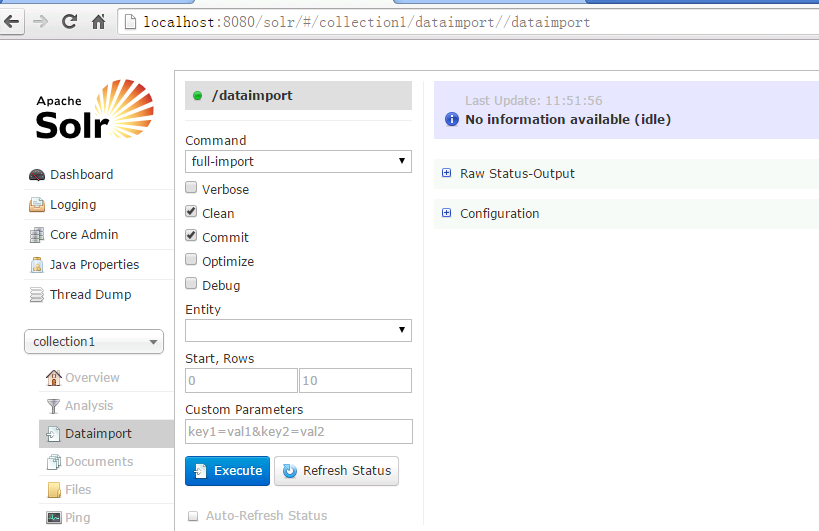
第二步：配置solrconfig.xml文件，添加一个requestHandler。

|  |
| --- |
| <requestHandler name="/dataimport"  class="org.apache.solr.handler.dataimport.DataImportHandler">  <lst name="defaults">  <str name="config">data-config.xml</str>  </lst>  </requestHandler> |

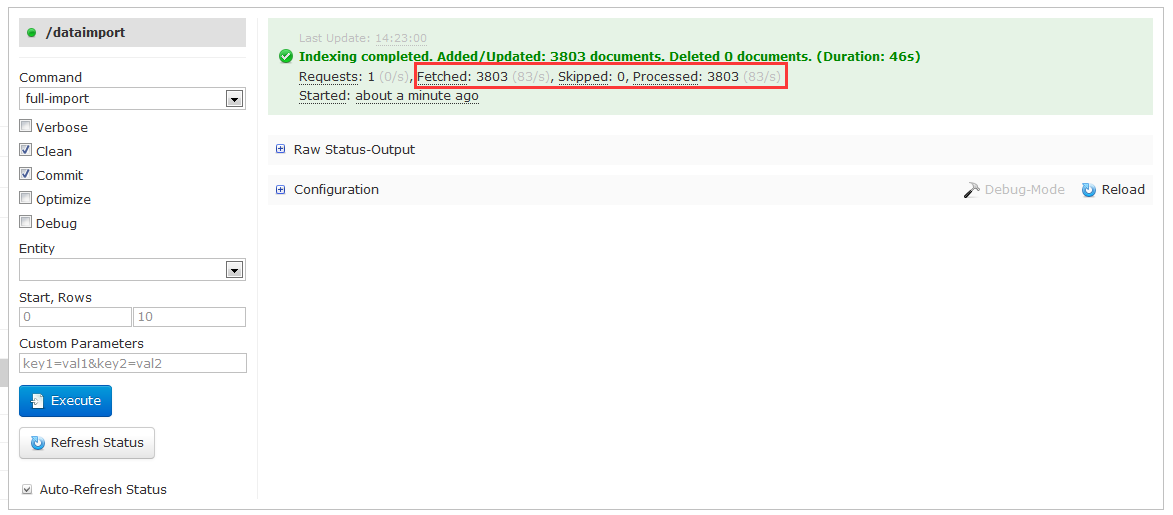
第三步：创建一个data-config.xml，保存到collection1\conf\目录下

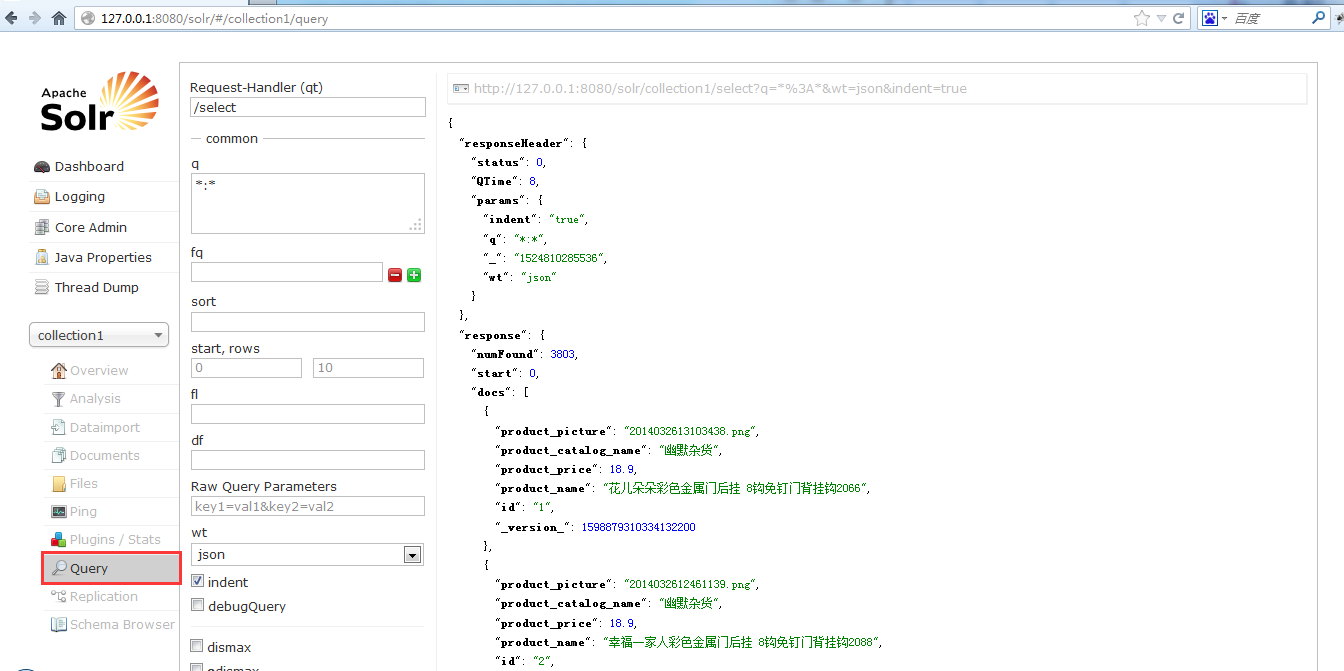
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <dataConfig>  <dataSource type="JdbcDataSource"  driver="com.mysql.jdbc.Driver"  url="jdbc:mysql://localhost:3306/solr"  user="root"  password="1111"/>  <document>  <entity name="product" query="SELECT pid,name,catalog\_name,price,description,picture 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：重启tomcat



第五步：点击“execute”按钮导入数据





### 删除文档

删除索引格式如下：

1） 删除制定ID的索引

<delete>

<id>1</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>

**注意：删除操作必须<commit/>才能生效**

## 查询索引（重点）

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\*:\*。

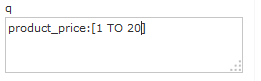


1. fq - （filter query）过虑查询，作用：在q查询符合结果中同时是fq查询符合的，例如：：



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。

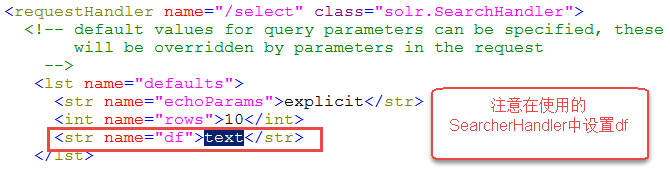


显示商品图片、商品名称、商品价格

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。



# 使用SolrJ管理索引库

## 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

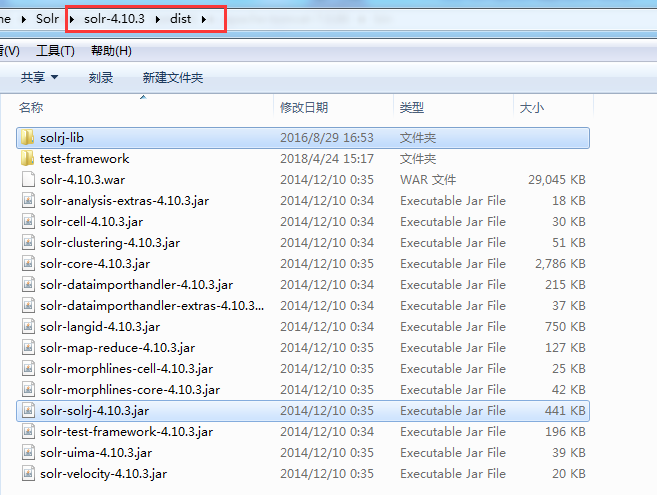
Solr服务

Tomcat

通过SolrJ请求Solr服务

最终Solr完在索引和搜索

## 依赖的jar包



pom.xml依赖：

<dependency>

<groupId>org.apache.solr</groupId>

<artifactId>solr-solrj</artifactId>

<version>4.10.3</version>

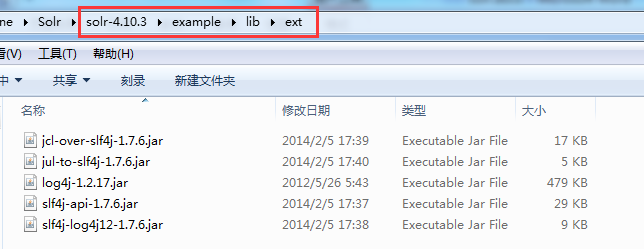
</dependency>

## 添加文档

### 实现步骤

第一步：创建一个java工程

第二步：导入jar包。包括solrJ的jar包。还需要



pom.xml依赖：

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jcl-over-slf4j</artifactId>

<version>1.7.6</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jul-to-slf4j</artifactId>

<version>1.7.6</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.6</version>

</dependency>

第三步：和Solr服务器建立连接。HttpSolrServer对象建立连接。

第四步：创建一个SolrInputDocument对象，然后添加域。

第五步：将SolrInputDocument添加到索引库。

第六步：提交。

### 代码实现

|  |
| --- |
| //向索引库中添加索引  @Test  **public** **void** addDocument() **throws** Exception {  //和solr服务器创建连接  //参数：solr服务器的地址  SolrServer solrServer = **new** HttpSolrServer("http://localhost:8080/solr");  //创建一个文档对象  SolrInputDocument document = **new** SolrInputDocument();  //向文档中添加域  //第一个参数：域的名称，域的名称必须是在schema.xml中定义的  //第二个参数：域的值  document.addField("id", "c0001");  document.addField("title\_ik", "使用solrJ添加的文档");  document.addField("content\_ik", "文档的内容");  document.addField("product\_name", "商品名称");  //把document对象添加到索引库中  solrServer.add(document);  //提交修改  solrServer.commit();    } |

## 删除文档

### 根据id删除

|  |
| --- |
| //删除文档，根据id删除  @Test  **public** **void** deleteDocumentByid() **throws** Exception {  //创建连接  SolrServer solrServer = **new** HttpSolrServer("http://localhost:8080/solr");  //根据id删除文档  solrServer.deleteById("c0001");  //提交修改  solrServer.commit();  } |

### 根据查询删除

查询语法完全支持Lucene的查询语法。

|  |
| --- |
| //根据查询条件删除文档  @Test  **public** **void** deleteDocumentByQuery() **throws** Exception {  //创建连接  SolrServer solrServer = **new** HttpSolrServer("http://localhost:8080/solr");  //根据查询条件删除文档  solrServer.deleteByQuery("\*:\*");  //提交修改  solrServer.commit();  } |

## 修改文档

在solrJ中修改没有对应的update方法，只有add方法，只需要添加一条新的文档，和被修改的文档id一致就，可以修改了。本质上就是先删除后添加。

## 查询文档

### 简单查询

|  |
| --- |
| //查询索引  @Test  **public** **void** queryIndex() **throws** Exception {  //创建连接  SolrServer solrServer = **new** HttpSolrServer("http://localhost:8080/solr");  //创建一个query对象  SolrQuery query = **new** SolrQuery();  //设置查询条件  query.setQuery("\*:\*");  //执行查询  QueryResponse queryResponse = solrServer.query(query);  //取查询结果  SolrDocumentList solrDocumentList = queryResponse.getResults();  //共查询到商品数量  System.*out*.println("共查询到商品数量:" + solrDocumentList.getNumFound());  //遍历查询的结果  **for** (SolrDocument solrDocument : solrDocumentList) {  System.*out*.println(solrDocument.get("id"));  System.*out*.println(solrDocument.get("product\_name"));  System.*out*.println(solrDocument.get("product\_price"));  System.*out*.println(solrDocument.get("product\_catalog\_name"));  System.*out*.println(solrDocument.get("product\_picture"));    }  } |

### 复杂查询

其中包含查询、过滤、分页、排序、高亮显示等处理。

|  |
| --- |
| //复杂查询索引  @Test  **public** **void** queryIndex2() **throws** Exception {  //创建连接  SolrServer solrServer = **new** HttpSolrServer("http://localhost:8080/solr");  //创建一个query对象  SolrQuery query = **new** SolrQuery();  //设置查询条件  query.setQuery("钻石");  //过滤条件  query.setFilterQueries("product\_catalog\_name:幽默杂货");  //排序条件  query.setSort("product\_price", ORDER.*asc*);  //分页处理  query.setStart(0);  query.setRows(10);  //结果中域的列表  query.setFields("id","product\_name","product\_price","product\_catalog\_name","product\_picture");  //设置默认搜索域  query.set("df", "product\_keywords");  //高亮显示  query.setHighlight(**true**);  //高亮显示的域  query.addHighlightField("product\_name");  //高亮显示的前缀  query.setHighlightSimplePre("<em>");  //高亮显示的后缀  query.setHighlightSimplePost("</em>");  //执行查询  QueryResponse queryResponse = solrServer.query(query);  //取查询结果  SolrDocumentList solrDocumentList = queryResponse.getResults();  //共查询到商品数量  System.*out*.println("共查询到商品数量:" + solrDocumentList.getNumFound());  //遍历查询的结果  **for** (SolrDocument solrDocument : solrDocumentList) {  System.*out*.println(solrDocument.get("id"));  //取高亮显示  String productName = "";  ~~Map<String, Map<String, List<String>>> highlighting = queryResponse.getHighlighting();~~  List<String> list = highlighting.get(solrDocument.get("id")).get("product\_name");  //判断是否有高亮内容  **if** (**null** != list) {  productName = list.get(0);  } **else** {  productName = (String) solrDocument.get("product\_name");  }    System.*out*.println(productName);  System.*out*.println(solrDocument.get("product\_price"));  System.*out*.println(solrDocument.get("product\_catalog\_name"));  System.*out*.println(solrDocument.get("product\_picture"));    }  } |

# 案例实现

## 原型分析



## 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

## pom.xml

创建一个web工程导入jar包

1. springmvc的相关jar包
2. solrJ的jar包
3. Example\lib\ext下的jar包

pom文件：

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.9.RELEASE</version>

</dependency>

<dependency>

<groupId>org.apache.solr</groupId>

<artifactId>solr-solrj</artifactId>

<version>4.10.3</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jcl-over-slf4j</artifactId>

<version>1.7.6</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jul-to-slf4j</artifactId>

<version>1.7.6</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.6</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<path>/</path>

<port>8080</port>

</configuration>

</plugin>

</plugins>

</build>

## web.xml

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

<web-app version=*"2.5"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<display-name></display-name>

<!-- 启动spring -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<!-- 处理中文乱码-->

<filter>

<filter-name>encoder</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>encoder</filter-name>

<url-pattern>\*.do</url-pattern>

</filter-mapping>

<!-- 启动springMVC -->

<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>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springMVC</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>product\_list.jsp</welcome-file>

</welcome-file-list>

</web-app>

## applicationContext.xml

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd "*>

<context:component-scan base-package=*"com.sxt.service,com.sxt.dao"*></context:component-scan>

</beans>

## springmvc.xml

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<context:component-scan base-package=*"com.sxt.controller"*></context:component-scan>

<mvc:annotation-driven></mvc:annotation-driven>

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

<!-- 把 solrServer交给spring去管理 -->

<bean id=*"solrServer"* class=*"org.apache.solr.client.solrj.impl.HttpSolrServer"*>

<!-- 给solrServer注入baseURL -->

<constructor-arg name=*"baseURL"* value=*"http://192.168.80.132:8080/solr"*/>

</bean>

</beans>

## Dao

功能：接收service层传递过来的参数，根据参数查询索引库，返回查询结果。

参数：SolrQuery对象

返回值：一个商品列表List<ProductModel>，还需要返回查询结果的总数量。返回：ResultModel

方法定义：ResultModel queryProduct(SolrQuery query) throws Exception;

商品对象模型：

**public** **class** ProductModel {

// 商品编号

**private** String pid;

// 商品名称

**private** String name;

// 商品分类名称

**private** String catalog\_name;

// 价格

**private** **float** price;

// 商品描述

**private** String description;

// 图片名称

**private** String picture;

}

返回值对象模型

**public** **class** ResultModel {

// 商品列表

**private** List<ProductModel> productList;

// 商品总数

**private** Long recordCount;

// 总页数

**private** **int** pageCount;

// 当前页

**private** **int** curPage;

}

|  |
| --- |
| @Repository  **public** **class** ProductDaoImpl **implements** ProductDao {  @Autowired  **private** SolrServer solrServer;    @Override  **public** ResultModel queryProduct(SolrQuery query) **throws** Exception {    ResultModel resultModel = **new** ResultModel();  //根据query对象查询商品列表  QueryResponse queryResponse = solrServer.query(query);  SolrDocumentList solrDocumentList = queryResponse.getResults();  //取查询结果的总数量  resultModel.setRecordCount(solrDocumentList.getNumFound());  List<ProductModel> productList = **new** ArrayList<>();  //遍历查询结果  **for** (SolrDocument solrDocument : solrDocumentList) {  //取商品信息  ProductModel productModel = **new** ProductModel();  productModel.setPid((String) solrDocument.get("id"));  //取高亮显示  String productName = "";  Map<String, Map<String, List<String>>> highlighting = queryResponse.getHighlighting();  List<String> list = highlighting.get(solrDocument.get("id")).get("product\_name");  **if** (**null** != list) {  productName = list.get(0);  } **else** {  productName = (String) solrDocument.get("product\_name");  }  productModel.setName(productName);  productModel.setPrice((**Float**) solrDocument.get("product\_price"));  productModel.setCatalog\_name((String) solrDocument.get("product\_catalog\_name"));  productModel.setPicture((String) solrDocument.get("product\_picture"));  //添加到商品列表  productList.add(productModel);  }  //商品列表添加到resultmodel中  resultModel.setProductList(productList);  **return** resultModel;  }  } |

## Service

功能：接收action传递过来的参数，根据参数拼装一个查询条件，调用dao层方法，查询商品列表。接收返回的商品列表和商品的总数量，根据每页显示的商品数量计算总页数。

参数：

1、查询条件：字符串

2、商品分类的过滤条件：商品的分类名称，字符串

3、商品价格区间：传递一个字符串，满足格式：“0-100、101-200、201-\*”

4、排序条件：页面传递过来一个升序或者降序就可以，默认是价格排序。0：升序1：降序

5、分页信息：每页显示的记录条数创建一个常量60条。传递一个当前页码就可以了。

返回值：ResultModel

方法定义：ResultModel queryProduct(String queryString, String caltalog\_name, String price, String sort, Integer page) throws Exception;

|  |
| --- |
| @Service  **public** **class** ProductServiceImpl **implements** ProductService {  @Autowired  **private** ProductDao productDao;    @Override  **public** ResultModel queryProduct(String queryString, String caltalog\_name,  String price, String sort, Integer page) **throws** Exception {  //拼装查询条件  SolrQuery query = **new** SolrQuery();  //查询条件  **if** (**null** != queryString && !"".equals(queryString)) {  query.setQuery(queryString);  } **else** {  query.setQuery("\*:\*");  }  //商品分类名称过滤  **if** (**null** != caltalog\_name && !"".equals(caltalog\_name)) {  query.addFilterQuery("product\_catalog\_name:" + caltalog\_name);  }  //价格区间过滤  **if** (**null** != price && !"".equals(price)) {  String[] strings = price.split("-");  query.addFilterQuery("product\_price:["+strings[0]+" TO "+strings[1]+"]");  }  //排序条件  **if** ("1".equals(sort)) {  query.setSort("product\_price", ORDER.*desc*);  } **else** {  query.setSort("product\_price", ORDER.*asc*);  }  //分页处理  **if** (**null** == page) {  page = 1;  }  //start  **int** start = (page-1) \* Commons.*PAGE\_SIZE*;  query.setStart(start);  query.setRows(Commons.*PAGE\_SIZE*);  //设置默认搜索域  query.set("df", "product\_keywords");  //高亮设置  query.setHighlight(**true**);  query.addHighlightField("product\_name");  query.setHighlightSimplePre("<span style=\"color:red\">");  query.setHighlightSimplePost("</span>");    //查询商品列表  ResultModel resultModel = productDao.queryProduct(query);  //计算总页数  **long** recordCount = resultModel.getRecordCount();  **int** pages = (**int**) (recordCount/Commons.*PAGE\_SIZE*);  **if** (recordCount % Commons.*PAGE\_SIZE* > 0) {  pages ++;  }  resultModel.setPageCount(pages);  resultModel.setCurPage(page);    **return** resultModel;  }  } |

## controller

功能：接收页面传递过来的参数调用service查询商品列表。将查询结果返回给jsp页面，还需要查询参数的回显。

参数：

1、查询条件：字符串

2、商品分类的过滤条件：商品的分类名称，字符串

3、商品价格区间：传递一个字符串，满足格式：“0-100、101-200、201-\*”

4、排序条件：页面传递过来一个升序或者降序就可以，默认是价格排序。0：升序1：降序

5、分页信息：每页显示的记录条数创建一个常量60条。传递一个当前页码就可以了。

6、Model：相当于request。

返回结果：String类型，就是一个jsp的名称。

String queryProduct(String queryString, String caltalog\_name, String price, String sort, Integer page, Model model) throws Exception;

|  |
| --- |
| @Controller  **public** **class** ProductAction {    @Autowired  **private** ProductService productService;    @RequestMapping("/list")  **public** String queryProduct(String queryString, String catalog\_name, String price,  String sort, Integer page, Model model) **throws** Exception {  //查询商品列表  ResultModel resultModel = productService.queryProduct(queryString, catalog\_name, price,  sort, page);  //列表传递给jsp  model.addAttribute("result", resultModel);  //参数回显  model.addAttribute("queryString", queryString);  model.addAttribute("caltalog\_name", catalog\_name);  model.addAttribute("price", price);  model.addAttribute("sort", sort);  model.addAttribute("page", page);    **return** "product\_list";  }  } |

## Jsp

参考资料。