Solr

全文搜索服务

# Solr介绍

## 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可以独立运行在Jetty、Tomcat等这些Servlet容器中。**

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**。

使用Solr 进行创建索引和搜索索引的实现方法很简单，如下：

* **创建索引：客户端（可以是浏览器可以是Java程序）**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服务器**根据xml文档添加、删除、更新索引 。
* **搜索索引：客户端（可以是浏览器可以是Java程序）**用 GET方法向 **Solr 服务器**发送请求，然后对 **Solr服务器**返回**Xml、json等格式的查询结果**进行解析，组织页面布局。Solr不提供构建页面UI的功能，但是**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和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应用。Lucene仅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应用。

**Solr的目标是打造一款企业级的搜索引擎系统，它是基于Lucene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solr

Solr和lucene的版本是同步更新的，最新的版本是5.2.1

本课程使用的版本：4.10.3

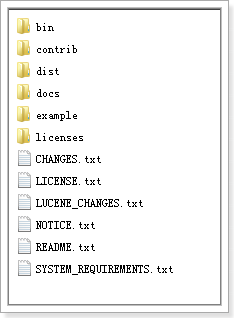
下载地址：<http://archive.apache.org/dist/lucene/solr/>

下载版本：4.10.3

<http://iamyida.iteye.com/blog/2210621> 数据库导入参考资料

Linux下需要下载lucene-4.10.3.tgz，windows下需要下载lucene-4.10.3.zip。

下载lucene-4.10.3.zip并解压：



**bin**：solr的运行脚本

**contrib**：solr的一些扩展jar包，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**：

该目录是一个标准的SolrHome，它包含一个默认的SolrCore

* **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，相关环境如下：

* Solr：4.10.3
* Jdk环境：1.7.0\_72（solr4.10 不能使用jdk1.7以下）
* Web服务器（servlet容器）：Tomcat 7X

## SolrCore配置

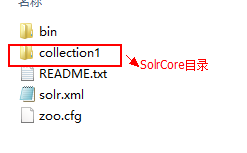
### SolrHome和SolrCore

**SolrHome是Solr运行的主目录**，该**目录中包括了多个SolrCore目录。SolrCore目录中包含了运行Solr实例所有的配置文件和数据文件，Solr实例就是SolrCore**。

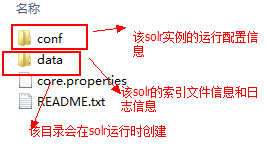
一个SolrHome可以包括多个SolrCore（Solr实例），每个SolrCore提供单独的搜索和索引服务。

### 目录结构

SolrHome目录：



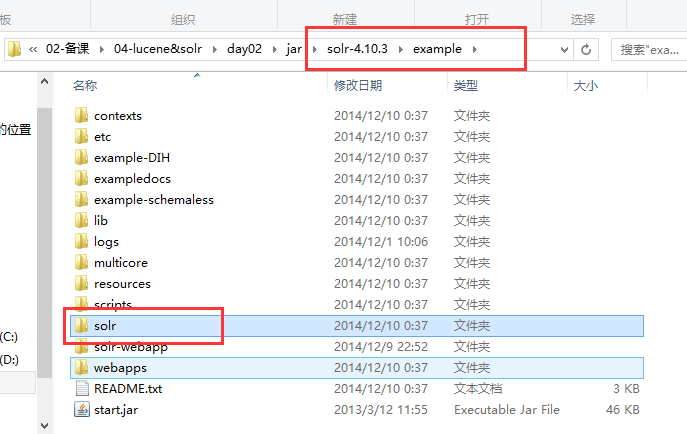
SolrCore目录：



### 创建SolrCore

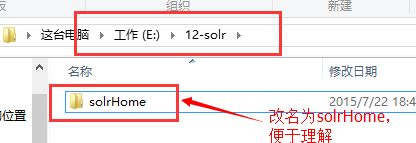
创建SolrCore先要创建SolrHome。在solr解压包下**solr-4.10.3\example\solr**文件夹就是一个标准的SolrHome。

* 拷贝solr解压包下**solr-4.10.3\example\solr**文件夹。

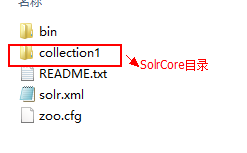


* 复制该文件夹到本地的一个目录，把文件名称改为solrhome。

注：改名不是必须的，只是为了便于理解



* 打开SolrHome目录



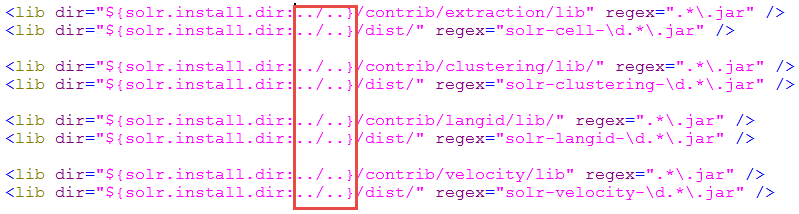
**SolrCore创建成功。**

### 配置SolrCore

在conf文件夹下有一个solrconfig.xml。这个文件是来配置SolrCore实例的相关信息。**如果使用默认配置可以不用做任何修改。**它里面包含了不少标签，但是我们关注的标签为：**lib标签、datadir标签、requestHandler标签**。

#### lib 标签

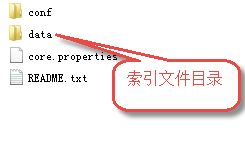
在solrconfig.xml中可以加载一些扩展的jar，solr.install.dir表示solrCore的目录位置，需要如下修改：



**然后将contrib和dist两个目录拷贝到E:\12-solr\0505下**

#### datadir标签

每个SolrCore都有自己的索引文件目录 ，默认在SolrCore目录下的data中。



data数据目录下包括了index索引目录 和tlog日志文件目录。

如果不想使用默认的目录也可以通过solrConfig.xml更改索引目录 ，如下：



#### requestHandler标签

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-->

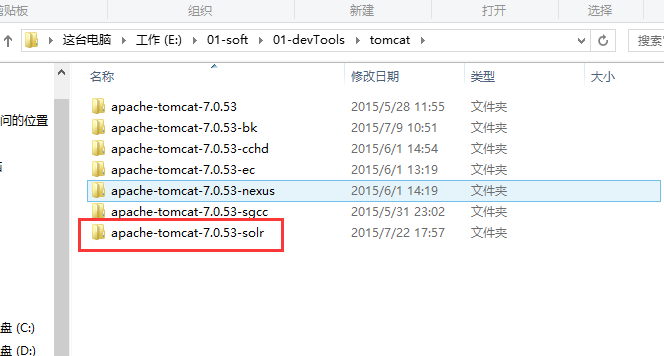
</lst>

</requestHandler>

## Solr工程部署

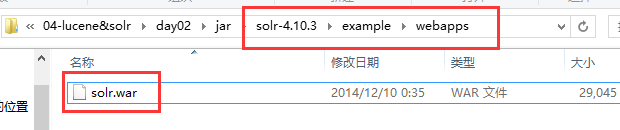
由于在项目中用到的web服务器大多数是用的Tomcat，所以就讲solr和Tomcat的整合。

### 安装Tomcat

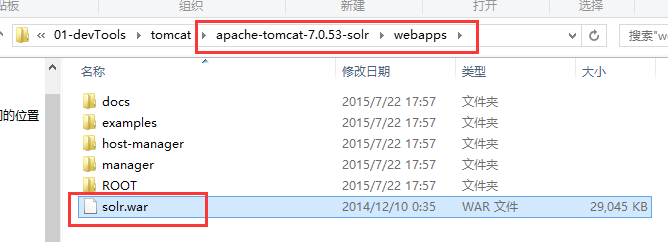


### 把solr.war部署到Tomcat中

1. 从solr解压包下的**solr-4.10.3\example\webapps**目录中拷贝solr.war

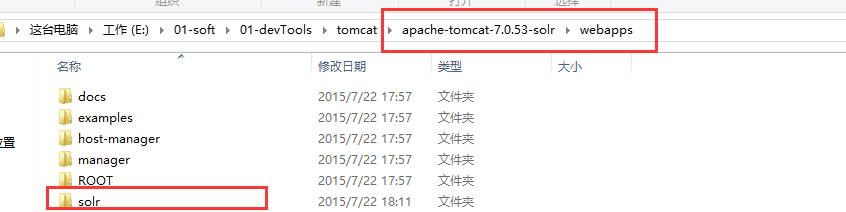


1. 复制到tomcat安装目录的**webapps**文件夹下



### 解压缩solr.war

使用压缩工具解压或者启动tomcat自动解压。解压之后删除solr.war

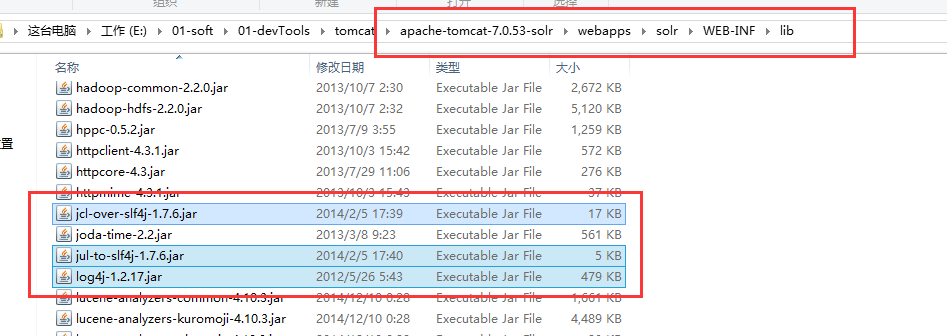


### 添加solr服务的扩展依赖包（日志包）

* 把solr解压包下的**solr-4.10.3\example\lib\ext**目录下的所有jar包拷贝。

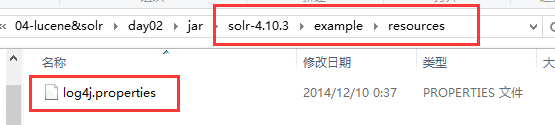


* 复制到解压缩后的solr工程的**WEB-INF\lib**目录

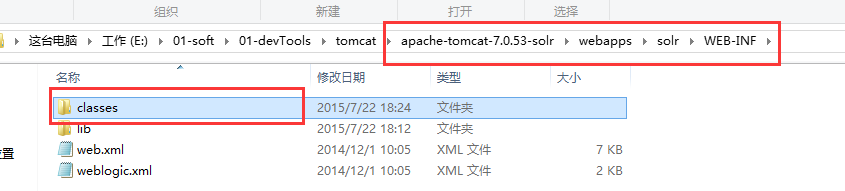


### 添加log4j.properties

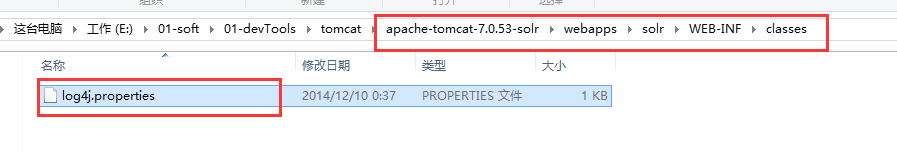
1. 把solr解压包下**solr-4.10.3\example\resources\log4j.properties**文件进行拷贝



1. 在解压缩后的solr工程中的**WEB-INF**目录中创建classes文件夹



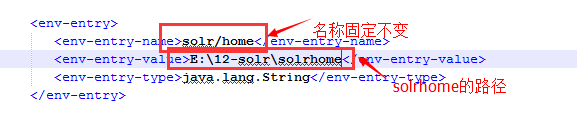
1. 复制log4j.properties文件到刚创建的classes目录



### 在solr应用的web.xml文件中，加载SolrHome

修改web.xml使用jndi的方式告诉solr服务器。

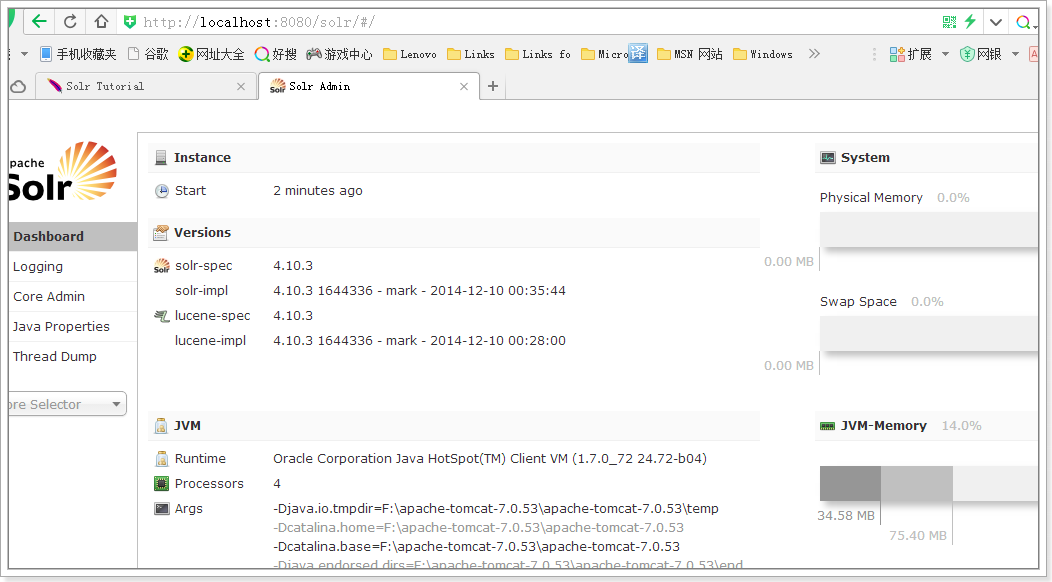
**Solr/home名称必须是固定的。**



### 启动Tomcat进行访问

访问<http://localhost:8080/solr/>

出现以下界面则说明solr安装成功！！！



## 管理界面功能介绍

### 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，该部分功能在第二个项目，即电商项目会讲解。

### Core Admin

Solr Core的管理界面。在这里可以添加SolrCore实例。

### java properties

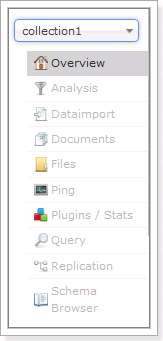
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

### 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（重点）

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。

注：solr中，**分析器是绑定在域的类型中的**。

#### dataimport

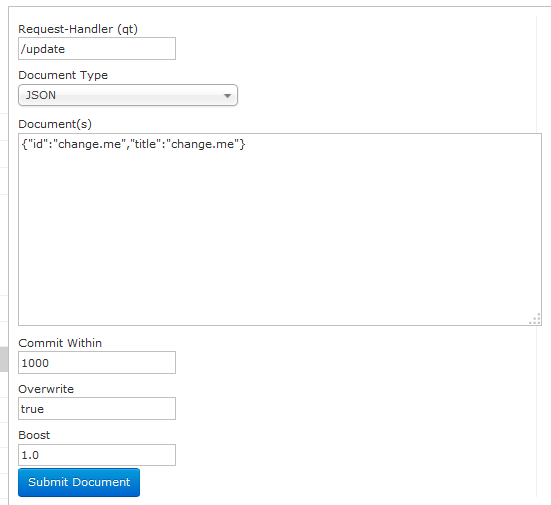
可以定义数据导入处理器，从关系数据库将数据导入到Solr索引库中。

默认没有配置，需要手工配置。

#### Document（重点）

通过/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**。

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：



* 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="1000" ： solr 在做索引的时候，每个1000（1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，</doc>后添加“<commit/>”

#### 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。



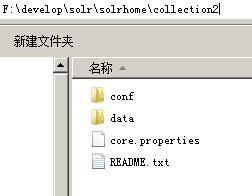
## 多solrcore配置

配置多SolrCore的好处：

1. 一个solr工程对外通过SorlCore 提供服务，每个SolrCore相当于一个数据库，这个功能就相当于一个mysql可以运行多个数据库。
2. 将索引数据分SolrCore存储，方便对索引数据管理维护。
3. SolrCloud集群需要使用多core。

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：

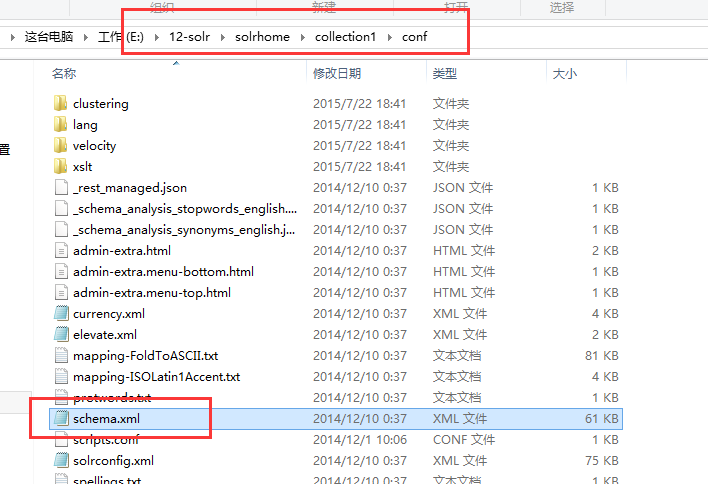


演示多core的使用，在collection1和collection2中分别创建索引、搜索索引。

# solr基本使用

## schema.xml

schema.xml文件在SolrCore的conf目录下，它是Solr数据表配置文件，在此配置文件中定义了域以及域的类型还有其他一些配置，**在solr中域必须先定义后使用**。



### field

|  |
| --- |
| <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" /> |

* Name：域的名称
* Type：域的类型
* Indexed：是否索引
* Stored：是否存储
* Required：是否必须
* multiValued：是否是多值，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）

### fieldType（域类型）

|  |
| --- |
| <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> |

* name：域类型的名称
* class：指定域类型的solr类型。
* analyzer：指定分词器。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。
* type：index和query。Index 是创建索引，query是查询索引。
* tokenizer：指定分词器
* filter：指定过滤器

### uniqueKey

|  |
| --- |
| <uniqueKey>id</uniqueKey> |

相当于主键，每个文档中必须有一个id域。

### copyField（复制域）

|  |
| --- |
| <copyField source="cat" dest="text" /> |

可以将多个Field复制到一个Field中，以便进行统一的检索。当创建索引时，solr服务器会自动的将源域的内容复制到目标域中。

* source：源域
* dest：目标域，搜索时，指定目标域为默认搜索域，可以提供查询效率。

定义目标域：



必须要使用：multiValued="true"

### dynamicField（动态域）

|  |
| --- |
| <dynamicField name="\*\_i" type="int" indexed="true" stored="true"/> |

* Name：动态域的名称，是一个表达式，\*匹配任意字符，只要域的名称和表达式的规则能够匹配就可以使用。

例如：搜索时查询条件【product\_i：钻石】就可以匹配这个动态域，可以直接使用，不用单独再定义一个product\_i域。

## 配置中文分析器

使用IKAnalyzer中文分析器。

第一步：把IKAnalyzer2012FF\_u1.jar添加到solr/WEB-INF/lib目录下。

第二步：复制IKAnalyzer的配置文件和自定义词典和停用词词典到solr的classpath下。

第三步：在schema.xml中添加一个自定义的fieldType，使用中文分析器。

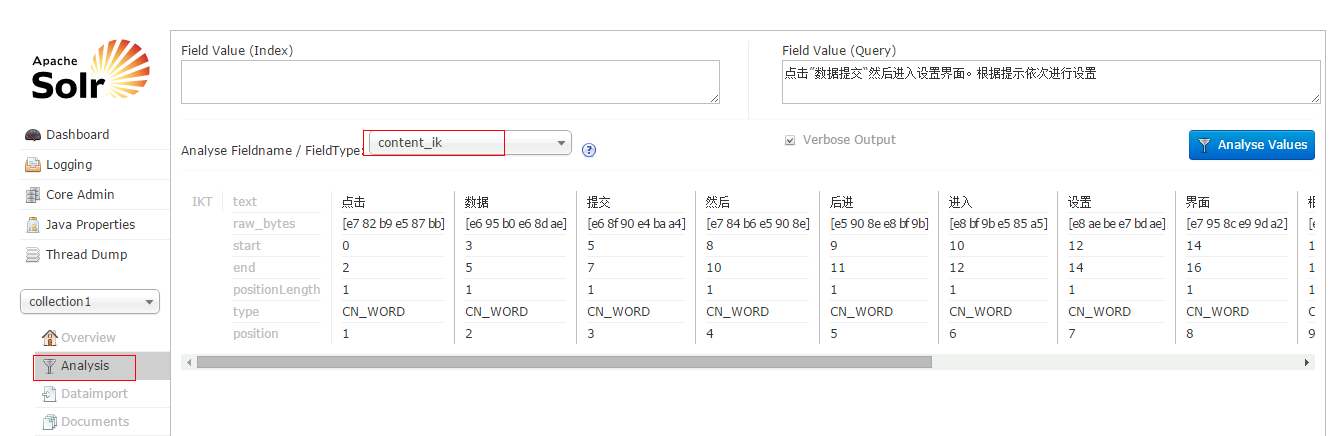
|  |
| --- |
| <!-- IKAnalyzer-->  <fieldType name="text\_ik" class="solr.TextField">  <analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>  </fieldType> |

第四步：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"/> |

第五步：重启tomcat

测试：



## 配置业务field

### 需求

要使用solr实现电商网站中商品搜索。

电商中商品信息在mysql数据库中存储了，将mysql数据库中数据在solr中创建索引。

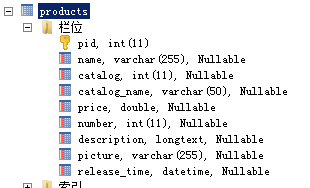
需要在solr的schema.xml文件定义商品Field。

### 定义步骤

先确定定义的商品document的field有哪些？

可以根据mysql数据库中商品表的字段来确定：

products商品表：



商品document的field包括：pid、name、catalog、catalog\_name、price、description、picture

先定义Fieldtype：

solr本身提供的fieldtype类型够用了不用定义新的了。

再定义Field：

pid：商品id主键

使用solr本身提供的：

<field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />

name：商品名称

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

catalog：商品分类

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

catalog\_name：商品分类名称

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

price：商品价格

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

description：商品描述

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

picture：商品图片

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

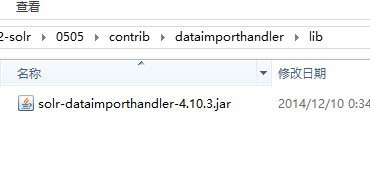
|  |
| --- |
| <!--product-->  <field name="product\_name" type="text\_ik" indexed="true" stored="true"/>  <field name="product\_catalog" type="string" indexed="true" stored="true"/>  <field name="product\_catalog\_name" type="string" indexed="true" stored="true" />  <field name="product\_price" type="float" 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\_keywords" type="text\_ik" indexed="true" stored="false" multiValued="true"/>  <copyField source="product\_name" dest="product\_keywords"/>  <copyField source="product\_description" dest="product\_keywords"/> |

## dataimportHandler插件

### 第一步：添加jar包

* Dataimport的jar

从solr-4.10.3\dist目录下拷贝solr-dataimporthandler-4.10.3.jar，复制到以下目录：

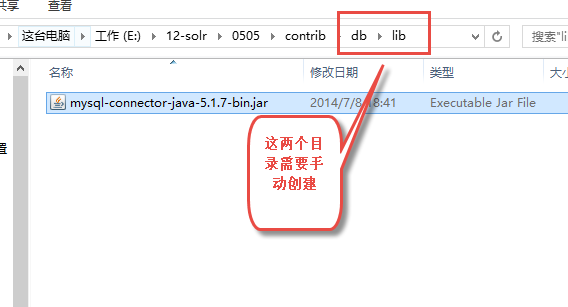


修改schema.xml如下：



* 数据库驱动包

把mysql数据库驱动包，拷贝到以下目录：



修改solrconfig.xml，如下：



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

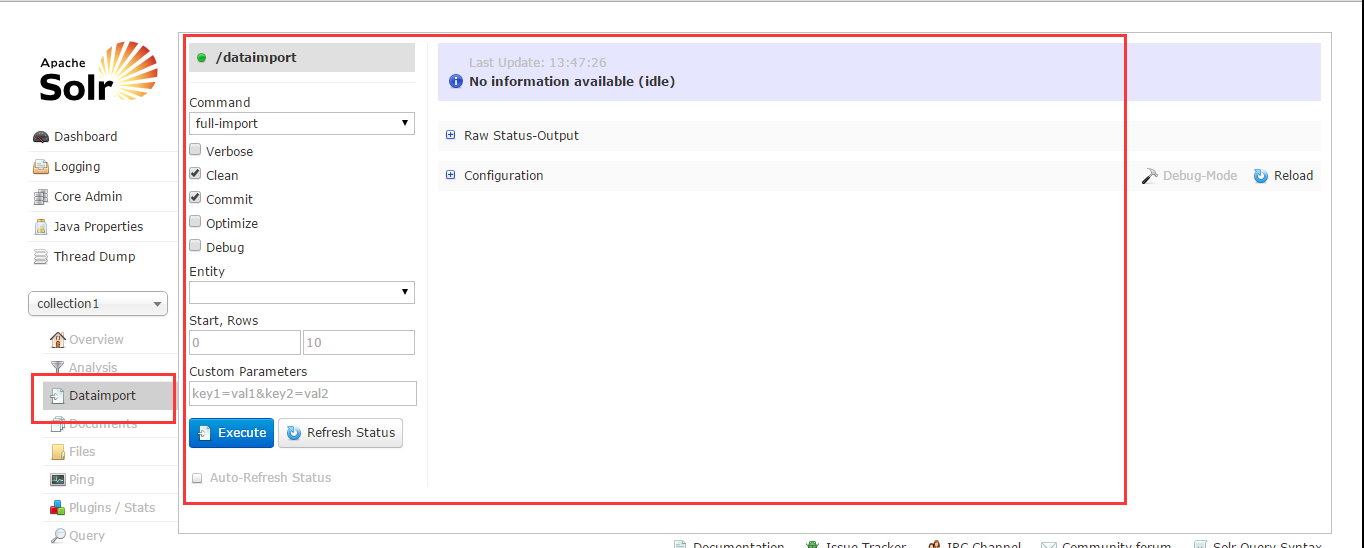
|  |
| --- |
| <requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">  <lst name="defaults">  <str name="config">data-config.xml</str>  </lst>  </requestHandler> |

### 第三步：创建一个data-config.xml

在collection1\conf\目录下创建data-config.xml文件

|  |
| --- |
| <?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="root"/>  <document>  <entity name="product" query="SELECT pid,name,catalog,catalog\_name,price,description,picture FROM products ">  <field column="pid" name="id"/>  <field column="name" name="product\_name"/>  <field column="catalog" name="product\_catalog"/>  <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

**注意：到入数据前会先清空索引库，然后再导入。**

# Solrj的使用

## 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

Solrj和图形界面操作的区别就类似于数据库中你使用jdbc和mysql客户端的区别一样。

## 需求

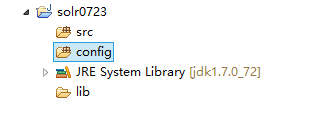
使用solrj调用solr服务实现对索引库的增删改查操作。

## 环境准备

* Solr：4.10.3
* Jdk环境：1.7.0\_72（solr4.10 不能使用jdk1.7以下）
* Ide环境：eclipse indigo

## 工程搭建

### 第一步：创建java工程



### 第二步：添加jar

* Solrj的包



* Solr服务的依赖包



## 代码实现

### 添加\修改索引

#### 步骤

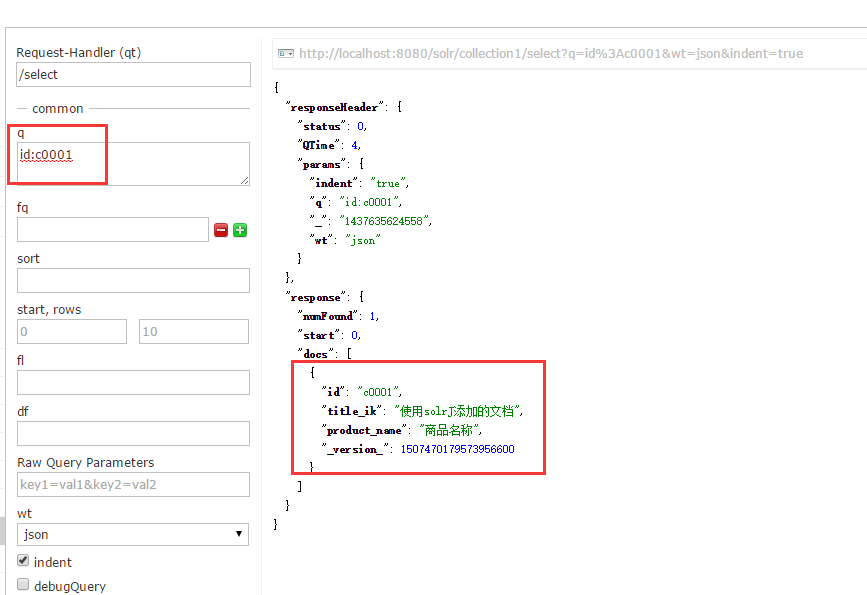
1. 创建HttpSolrServer对象，通过它和Solr服务器建立连接。
2. 创建SolrInputDocument对象，然后通过它来添加域。
3. 通过HttpSolrServer对象将SolrInputDocument添加到索引库。
4. 提交。

#### 代码

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**。

|  |
| --- |
| @Test  **public** **void** addDocument() **throws** Exception {  // 1、 创建HttpSolrServer对象，通过它和Solr服务器建立连接。  // 参数：solr服务器的访问地址  HttpSolrServer server = **new** HttpSolrServer("http://localhost:8080/solr/");  // 2、 创建SolrInputDocument对象，然后通过它来添加域。  SolrInputDocument document = **new** SolrInputDocument();  // 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  // 注意：id的域不能少  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");  // 3、 通过HttpSolrServer对象将SolrInputDocument添加到索引库。  server.add(document);  // 4、 提交。  server.commit();  } |

#### 查询测试



### 删除索引

#### 根据ID删除

##### 代码

|  |
| --- |
| @Test  **public** **void** deleteDocument() **throws** Exception {  // 1、 创建HttpSolrServer对象，通过它和Solr服务器建立连接。  // 参数：solr服务器的访问地址  HttpSolrServer server = **new** HttpSolrServer(  "http://localhost:8080/solr/");  // 根据ID删除  server.deleteById("c0001");  // 提交  server.commit();  } |

##### 查询测试



#### 根据条件删除

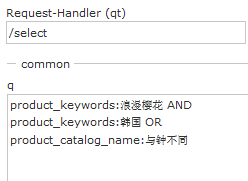
|  |
| --- |
| @Test  **public** **void** deleteDocumentByQuery() **throws** Exception {  // 1、 创建HttpSolrServer对象，通过它和Solr服务器建立连接。  // 参数：solr服务器的访问地址  HttpSolrServer server = **new** HttpSolrServer(  "http://localhost:8080/solr/");  // 根据ID删除  server.deleteByQuery("id:c0001");  // 全部删除  // server.deleteByQuery("\*:\*");  // 提交  server.commit();  } |

### 查询索引

#### solr的查询语法

1. q - 查询关键字，必须的，如果查询所有使用\*:\*。

请求的q是字符串



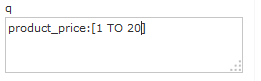
1. fq - （filter query）过虑查询，作用：在q查询符合结果中同时是fq查询符合的，例如：：

请求fq是一个数组（多个值）



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。

实际开发时，知道当前页码和每页显示的个数最后求出开始下标。

1. fl - 指定返回那些字段内容，用逗号或空格分隔多个。

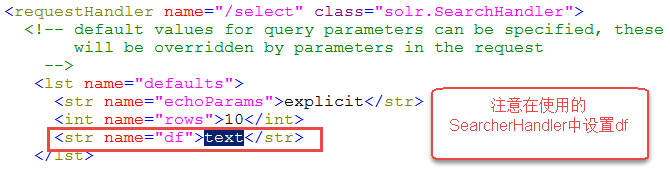


显示商品图片、商品名称、商品价格

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。



#### 简单查询

|  |
| --- |
| @Test  **public** **void** queryIndex() **throws** Exception {  // 创建HttpSolrServer对象，通过它和Solr服务器建立连接。  // 参数：solr服务器的访问地址  HttpSolrServer server = **new** HttpSolrServer(  "http://localhost:8080/solr/");  // 创建SolrQuery对象  SolrQuery query = **new** SolrQuery();  // 设置查询条件,名称“q”是固定的且必须 的  query.set("q", "id:2");  // 调用server的查询方法，查询索引库  QueryResponse response = server.query(query);  // 查询结果  SolrDocumentList results = response.getResults();  // 查询结果总数  **long** cnt = results.getNumFound();  System.*out*.println("查询结果总数:" + cnt);  **for** (SolrDocument solrDocument : results) {  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

复杂查询中包括高亮的处理

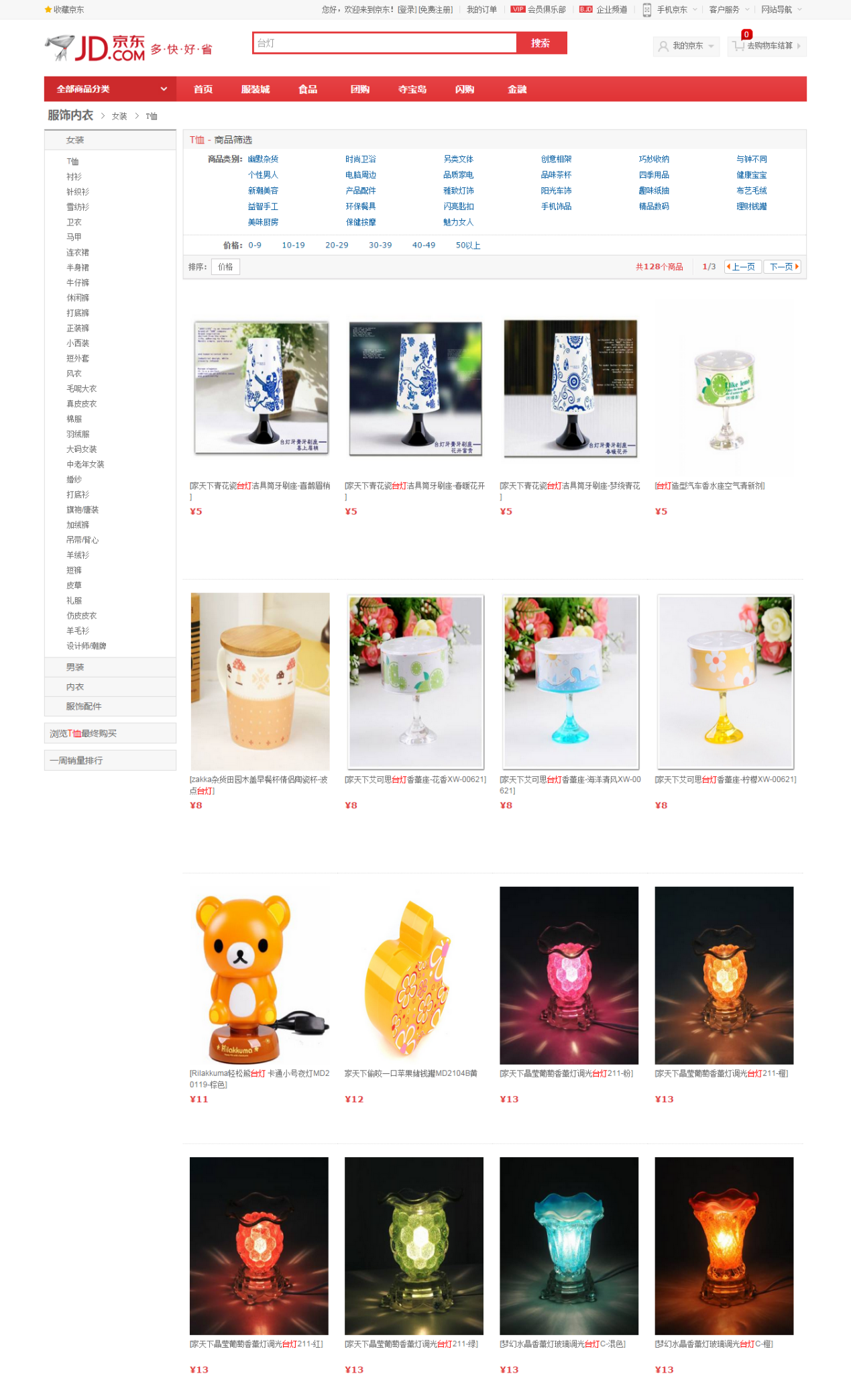
|  |
| --- |
| @Test  **public** **void** queryIndex2() **throws** Exception {  // 创建HttpSolrServer对象，通过它和Solr服务器建立连接。  // 参数：solr服务器的访问地址  HttpSolrServer server = **new** HttpSolrServer("http://localhost:8080/solr/");  // 创建SolrQuery对象  SolrQuery query = **new** SolrQuery();  // 设置查询条件  query.setQuery("钻石");  // 设置过滤条件  query.setFilterQueries("product\_catalog\_name:幽默杂货");  // 设置排序  query.setSort("product\_price", ORDER.*desc*);  // 设置分页信息  query.setStart(0);  query.setRows(10);  // 设置显得的域的列表  query.setFields("id", "product\_name", "product\_price",  "product\_catalog\_name", "product\_picture");  // 设置默认搜索域  query.set("df", "product\_name");  // 设置高亮  query.setHighlight(**true**);  query.addHighlightField("product\_name");  query.setHighlightSimplePre("<em>");  query.setHighlightSimplePost("</em>");  // 调用server的查询方法，查询索引库  QueryResponse response = server.query(query);  // 查询结果  SolrDocumentList results = response.getResults();  // 查询结果总数  **long** cnt = results.getNumFound();  System.*out*.println("查询结果总数:" + cnt);  **for** (SolrDocument solrDocument : results) {  System.*out*.println(solrDocument.get("id"));  String productName = (String) solrDocument.get("product\_name");  //获取高亮列表  Map<String, Map<String, List<String>>> highlighting = response  .getHighlighting();  //获得本文档的高亮信息  List<String> list = highlighting.get(solrDocument.get("id")).get(  "product\_name");  //如果有高亮，则把商品名称赋值为有高亮的那个名称  **if** (list != **null**) {  productName = list.get(0);  }  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

## 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：



## 分析

开发人员需要的文档：静态页面（根据UI设计由美工给出）、数据库设计、原型设计

### UI分析



### 架构分析

spring容器

表现层

springmvc

Service层

Dao层

Solr索引库

mysql数据库

商品信息

Tomcat

Solr服务

索引、搜索请求

架构分为：

* 1. solr服务器
  2. 自己开发的应用（重点）
  3. 数据库mysql

自己开发的应用

* 1. Controller

和前端页面进行请求和响应的交互

* 1. Service

使用solrj来调用solr的服务进行索引和搜索

Service调用dao进行商品数据的维护时，要同步更新索引库

* 1. Dao

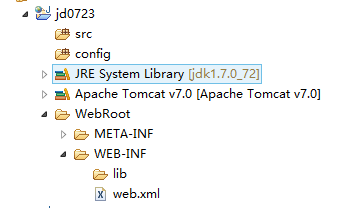
对商品数据进行维护和查询

## 环境准备

* Solr：4.10.3
* Jdk环境：1.7.0\_72（solr4.10 不能使用jdk1.7以下）
* Ide环境：eclipse indigo
* Web服务器（servlet容器）：Tomcat 7X

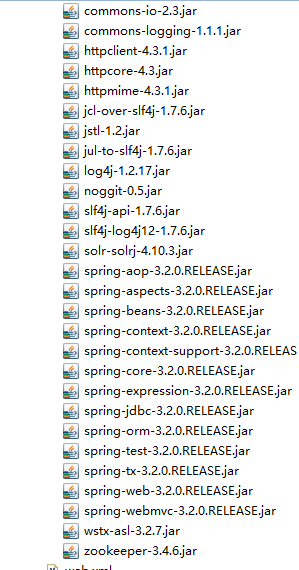
## 工程搭建

### 第一步：创建web工程



### 第二步：添加jar包

* Solrj的包
* Solr服务的日志包
* Spring的包（包含springmvc）
* Jstl包
* Commons日志包

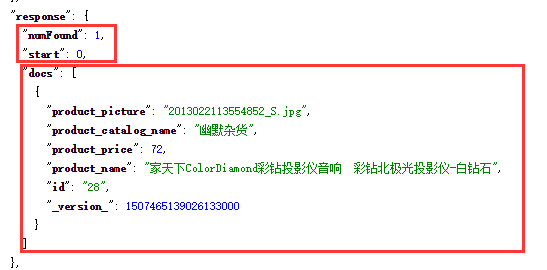


## 代码实现

### Pojo

#### 分析

索引查询结果



通过分析得出：

* 需要一个商品的pojo（Product），存放商品信息
* 还需要一个包装pojo（ResultModel），它包括商品列表信息、商品分页信息

#### 代码

Product.java

|  |
| --- |
| **public** **class** Product {  // 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;  } |

ResultModel.java

|  |
| --- |
| **public** **class** ResultModel {  // 商品列表  **private** List<Product> productList;  // 商品总数  **private** Long recordCount;  // 总页数  **private** **int** pageCount;  // 当前页  **private** **int** curPage;  } |

### 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。

|  |
| --- |
| **public** **interface** ProductDao {  //查询商品信息，包括分页信息  **public** ResultModel queryProduct(SolrQuery query) **throws** Exception;  } |

|  |
| --- |
| **public** **class** ProductDaoImpl **implements** ProductDao {  @Autowired  **private** HttpSolrServer server;  @Override  **public** ResultModel queryProduct(SolrQuery query) **throws** Exception {  ResultModel result = **new** ResultModel();  // 通过server查询索引库  QueryResponse response = server.query(query);  // 获得查询结果  SolrDocumentList documentList = response.getResults();  // 把查询结果总数设置到ResultModel  result.setRecordCount(documentList.getNumFound());  List<Product> productList = **new** ArrayList<>();  Product product = **null**;  // 高亮信息  Map<String, Map<String, List<String>>> highlighting = response  .getHighlighting();  **for** (SolrDocument solrDocument : documentList) {  product = **new** Product();  product.setPid((String) solrDocument.get("id"));  String prodName = (String) solrDocument.get("product\_name");  List<String> list = highlighting.get(solrDocument.get("id")).get(  "product\_name");  **if** (list != **null**)  prodName = list.get(0);  product.setName(prodName);  product.setCatalog\_name((String) solrDocument  .get("product\_catalog\_name"));  product.setPrice((**float**) solrDocument.get("product\_price"));  product.setPicture((String) solrDocument.get("product\_picture"));  productList.add(product);  }  // 把商品列表放到ResultMap中  result.setProductList(productList);  **return** result;  }  } |

### 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。

|  |
| --- |
| **public** **interface** ProductService {  **public** ResultModel queryProduct(String queryString, String cataName,  String price, String sort, Integer curPage) **throws** Exception;  } |

|  |
| --- |
| @Service  **public** **class** ProductServiceImpl **implements** ProductService {  @Autowired  **private** ProductDao dao;  @Override  **public** ResultModel queryProduct(String queryString, String cataName,  String price, String sort, Integer curPage) **throws** Exception {  // 封装SolrQuery  SolrQuery query = **new** SolrQuery();  // 设置查询条件  **if** (queryString != **null** && !"".equals(queryString)) {  query.setQuery(queryString);  }**else**{  query.setQuery("\*:\*");  }  // 设置过滤条件  **if** (cataName != **null** && !"".equals(cataName)) {  query.addFilterQuery("product\_catalog\_name:" + cataName);  }  **if** (price != **null** && !"".equals(price)) {  String[] strings = price.split("-");  **if** (strings.length == 2) {  query.addFilterQuery("product\_price:[" + strings[0] + " TO "  + strings[1] + "]");  }  }  // 设置排序（1：降序 0：升序）  **if** ("1".equals(sort)) {  query.setSort("product\_price", ORDER.*desc*);  } **else** {  query.setSort("product\_price", ORDER.*asc*);  }  // 设置分页信息  **if** (curPage == **null**)  curPage = 1;  query.setStart((curPage - 1) \* Constants.*rows*);  query.setRows(Constants.*rows*);  // 设置默认搜索域  query.set("df", "product\_name");  // 设置高亮  query.setHighlight(**true**);  query.addHighlightField("product\_name");  query.setHighlightSimplePre("<font style=\"color:red\">");  query.setHighlightSimplePost("</font>");  ResultModel resultModel = dao.queryProduct(query);  resultModel.setCurPage(curPage);  **return** resultModel;  }  } |

### Controller

#### 代码

|  |
| --- |
| @Controller  **public** **class** ProductController {  @Autowired  **private** ProductService service;  @RequestMapping("/list")  **public** String queryProduct(String queryString, String catalog\_name,  String price, String sort, Integer page, Model model)  **throws** Exception {  ResultModel resultModel = service.queryProduct(queryString, catalog\_name,  price, sort, page);  model.addAttribute("result", resultModel);  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";  }  } |

#### 配置

##### Web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"WebApp\_ID"* version=*"2.5"*>  <display-name>jd0723</display-name>  <!-- SpringMVC配置 -->  <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>  <filter>  <filter-name>Character Encoding</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>Character Encoding</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  </web-app> |

##### Springmvc.xml

在config包下，创建springmvc.xml文件

|  |
| --- |
| <?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:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-3.1.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.1.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-3.1.xsd "*>  <!-- 配置扫描包 -->  <context:component-scan base-package=*"cn.itcast"* />  <!-- 配置注解驱动 -->  <mvc:annotation-driven />  <!-- jsp视图解析器 -->  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <!-- 前缀 -->  <property name=*"prefix"* value=*"/WEB-INF/jsp/"*></property>  <!-- 后缀 -->  <property name=*"suffix"* value=*".jsp"*></property>  </bean>  <bean class=*"org.apache.solr.client.solrj.impl.HttpSolrServer"*>  <constructor-arg value=*"http://localhost:8080/solr/"*></constructor-arg>  </bean>  </beans> |

### 启动Tomcat

访问：<http://localhost:8180/jd0723/list.action>

