Lucene复习：

1. 什么是lucene：全文检索工具包
2. Lucene的工作原理：

索引数据的创建

从原始文件中提取一些可以用来搜索的数据（封装成各种Field），把各field再封装成document，然后对document进行分析（对各字段分词），得到一些索引目录写入索引库，document本身也会被写入一个文档信息库；

索引数据的查询

根据关键词解析（queryParser）出查询条件query，利用搜索工具（indexSearcher）去索引库获取文档id，然后再根据文档id去文档信息库获取文档信息

分词器不同，建立的索引数据就不同；比较通用的一个中文分词器IKAnalyzer的用法

1. 相关度得分
   1. 在建立索引的时候，给指定文档的指定域设置一个权重

Field.setBoosts()

* 1. 在搜索的时候，可以给不同的搜索域设置不同的权重

Boosts = new HashMap<String,Float>

MultiFieldsQueryParser（fields,analyzer,boosts）

Solr笔记

# 课程计划

1. 站内搜索技术选型。
2. Solr相关概念
3. **Solr的安装及配置**

Solr整合tomcat

1. solr后台管理界面的功能介绍
   1. 维护索引
   2. 查询索引
2. solr实现索引库的维护
   1. 增删改操作
   2. 批量添加数据，从数据库中把数据导入到索引库。Dataimport插件。
3. 索引库的查询
4. SolrJ客户端，
   1. 索引库的维护
   2. 索引库的查询
5. 综合案例，电商网站的搜索。

# 站内搜索技术选型

1. Lucene：可以实现站内搜索。需要大量的开发工作。索引库的维护及优化。查询的优化等问题都需要我们自己来解决。不推荐使用。
2. 使用第三方搜素引擎实现。使用百度实现站内搜索。免费的。索引库无法维护。适合一些小的网站。不推荐使用。
3. Solr：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。提供了完整的集群方案，和索引库优化方案。

# 什么是solr

为什么要solr：

1、solr已经将整个索引操作功能封装好了的搜索引擎系统(企业级搜索引擎产品)

2、solr可以部署到单独的服务器上(WEB服务)，它可以提供服务，我们的业务系统就只要发送请求，接收响应即可，降低了业务系统的负载

3、solr部署在专门的服务器上，它的索引库就不会受业务系统服务器存储空间的限制

4、solr支持分布式集群，索引服务的容量和能力可以线性扩展

solr的工作机制：

1、solr就是在lucene工具包的基础之上进行了封装，而且是以web服务的形式对外提供索引功能

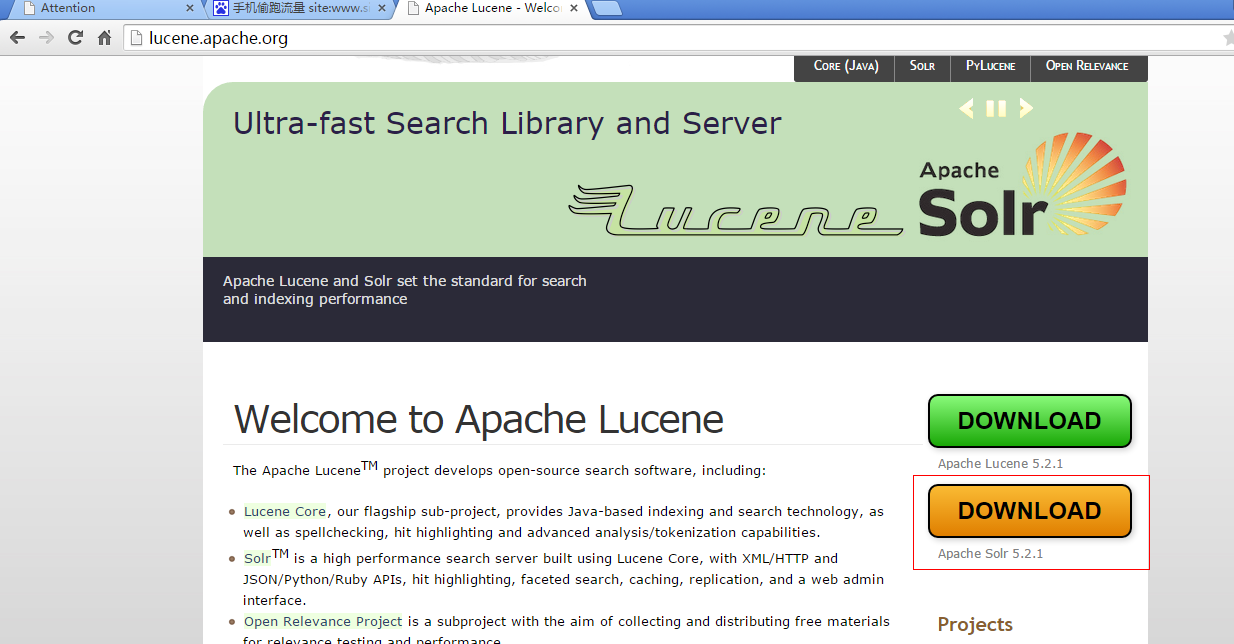
2、业务系统需要使用到索引的功能（建索引，查索引）时，只要发出http请求，并将返回数据进行解析即可

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。

**就是一个web工程**

## Solr的下载



<http://lucene.apache.org/>

最新版：5.3.1

本课程的版本：4.10.3



## Solr和lucene区别

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

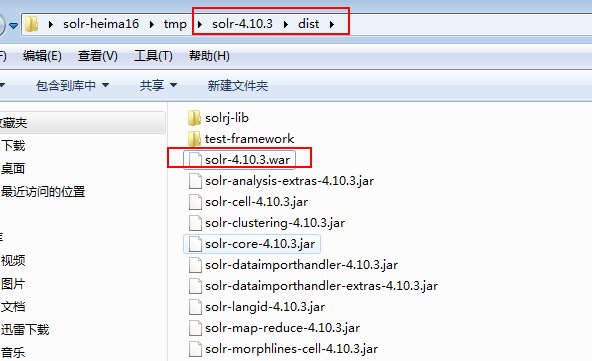
Solr的目标是打造一款企业级的搜索引擎系统，它是一个搜索引擎服务，可以独立运行，通过Solr可以非常快速的构建企业的搜索引擎，通过Solr也可以高效的完成站内搜索功能。



# Solr的安装及配置



安装solr，就是去部署它的war包，war包所在的位置如图：



## 开发环境

Jdk：1.7以上。1.7.0\_72

Tomcat：7以上。apache-tomcat-7.0.53

Solr：4.10.3

## Solr集成tomcat

第一步：安装tomcat，建议安装一个全新的tomcat。

第二步：把solr的war包\solr-4.10.3\dist\solr-4.10.3.war复制到webapps下，改名为solr.war，改名不是必须的，为了方便访问。

第三步：解压solr.war。启动tomcat会自动解压。也可以使用解压缩软件。

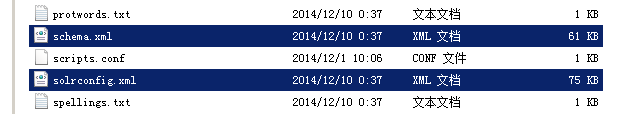
第四步：把\solr-4.10.3\example\lib\ext目录下所有的jar包添加到webapps中解压好的solr工程中webapps\solr\WEB-INF\lib下

第五步：创建一个solrhome（就是solr所有配置文件存放的目录）solr-4.10.3\example\solr文件夹就是一个标准的solrhome，可以直接使用。把solr文件夹复制到D:\temp\heima16下，改名为solrhome。

**了解solrhome：**

1、collection1：是一个solrcore，一个solrcore就是一个索引库。一个solr服务器上可以有多solrcore。每个索引库之间是相互独立的。

2、\solrhome\collection1\conf：是存放每个solrcore的个性配置。



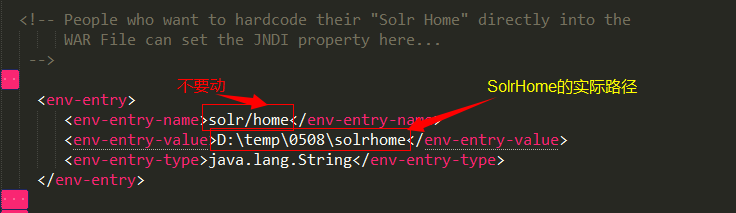
1. Solrconfig.xml
   1. luceneMatchVersion：匹配lucene的版本信息
   2. Lib：solrcore扩展使用的jar包。默认值是collection1\lib，如果没有此文件夹就创建一个。
   3. dataDir：索引库存放的目录。默认是collection1\data文件夹。如果没有solr会自动创建。如果想修改为其他位置，需要配置此节点。
   4. requestHandler：配置solr对外提供服务的url
      1. <requestHandler name="/select" class="solr.SearchHandler">：查询索引库使用的url
      2. <requestHandler name="/update" class="solr.UpdateRequestHandler">

维护索引库使用的url

* 1. defaultQuery：管理页面默认的查询条件 \*:\*

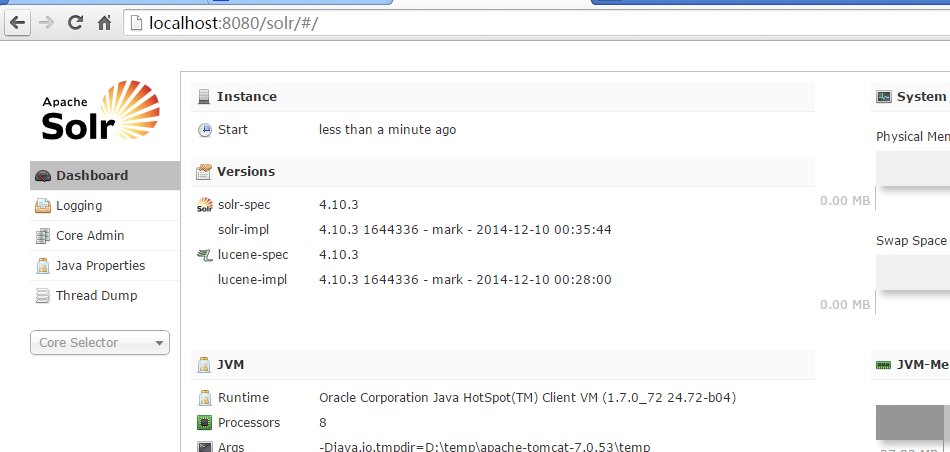
1. Core.properties：配置了solrcore的名字。

第六步：告诉solr工程solrhome的位置。修改solr/WEB-INF/web.xml文件。



第七步：启动tomcat

访问http://localhost:8080/solr



# 管理界面功能介绍

## Core Admin

solrcore的管理 功能。

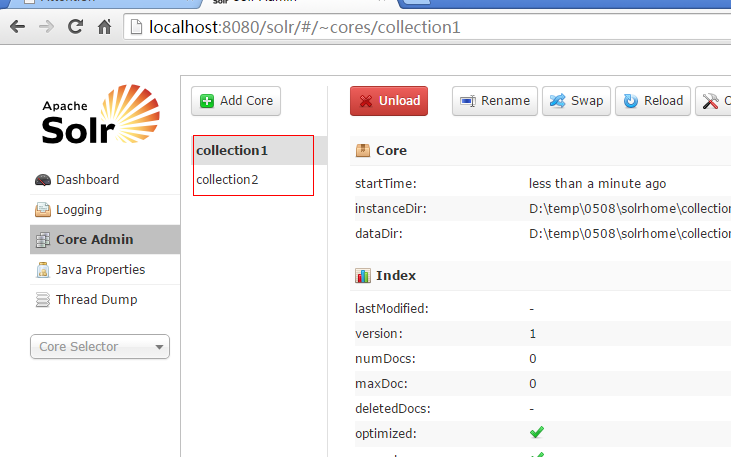
### 添加一个solrcore

添加步骤：

第一步：把collection1复制一份改名为collection2

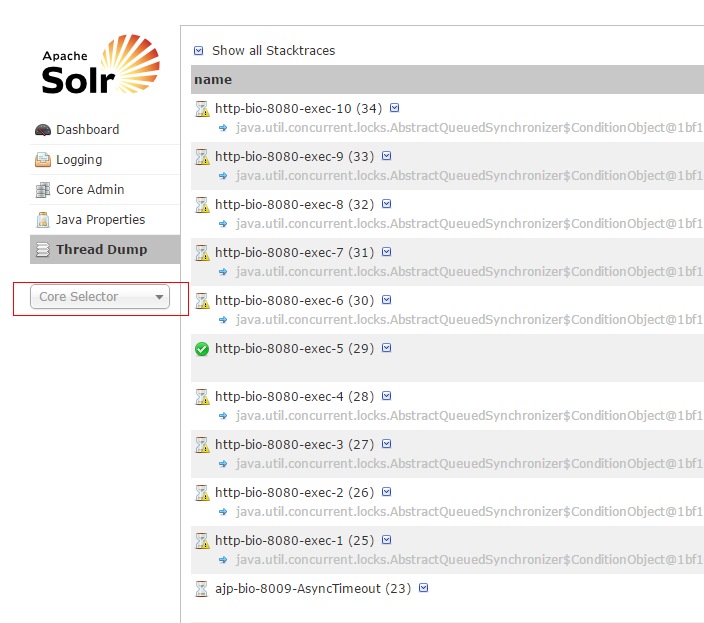
第二步：修改core.properties。name=collection2

第三步：重启tomcat

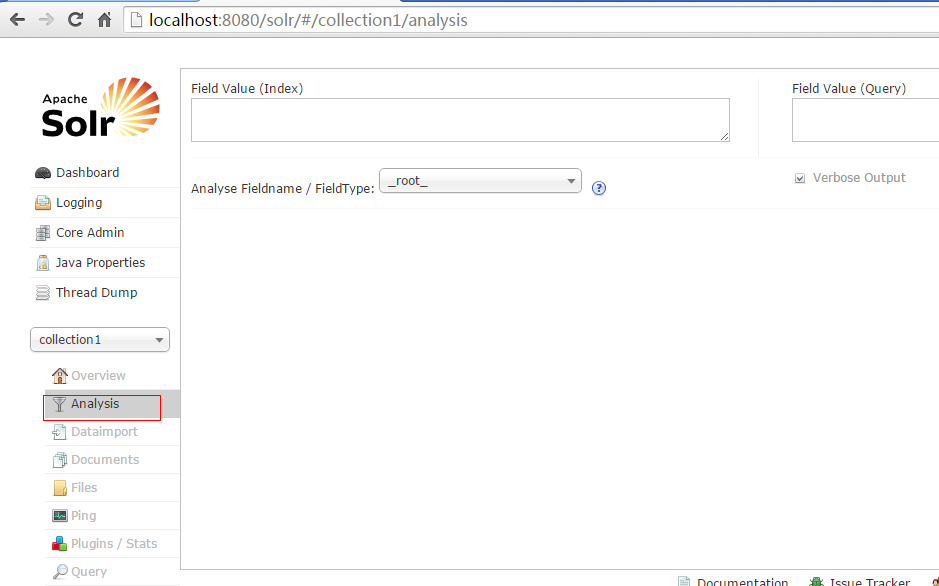


## Core Selecter

选择要管理哪个solrcore。



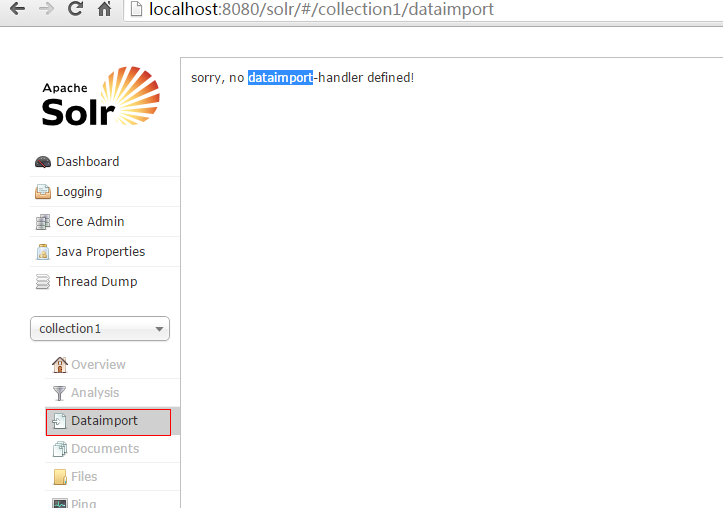
## Analysis



查看域的分词效果。

## Dataimport

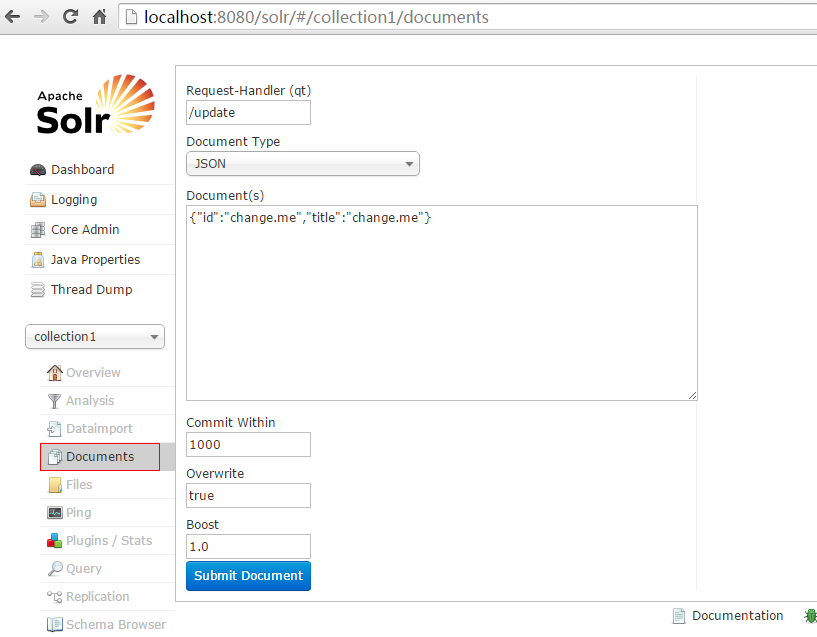
可以实现把数据库中的数据导入到索引库中。



## Documents

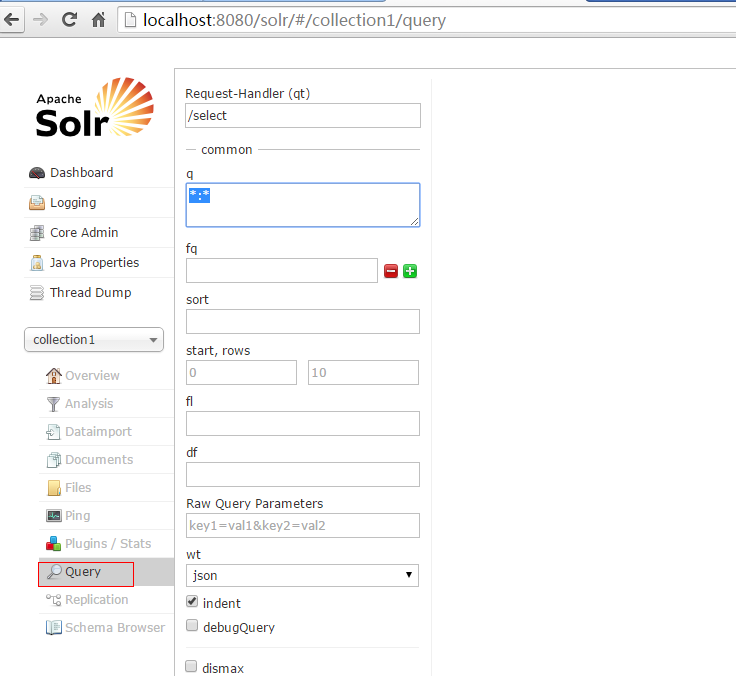
索引库维护功能。

增删改查



## Query

索引查询功能。



# 索引的维护

在solr中域必须先定义后使用。而且每个document中必须有一个id域。

## Schema.xml

### Field:域的定义。

Name：域的名称

Type：域的类型

Indexed：是否索引

Stored：是否存储

multiValued：是否多值，如果是多值在一个域中可以保持多个值。

### dynamicField动态域

Name：域的名称，是一个表达式。如果域的名称和表达式相匹配，此域名就可以使用。

Type：域的类型

Indexed：是否索引

Stored：是否存储

multiValued：是否多值，如果是多值在一个域中可以保持多个值。

### uniqueKey

每个文档必须有一个uniqueKey，而且不能重复。相当于表中的主键。

### copyField

复制域。

Source：源域

Dest：目标域。

创建文档时，solr会自动把源域的内容复制到目标域。使用复制域可以提供查询的性能。

### fieldType

域的类型。

Name：域类型名。

Class：对应的实现类。solr.TextField类似于Lucene中的TextField。可以配置用户自定义的分析器。

### 自定义fieldType使用中文分析器

#### 配置中文分析器

配置步骤：

第一步：把IKAnalyzer2012FF\_u1.jar添加到solr工程的lib库中。

第二步：把配置文件和扩展词典、停用词词典添加到solr工程classpath下。Solr/WEB-INF/classes。保证字典的字符集是utf-8.

#### 配置自定义fieldtype

在schema.xml中添加如下内容：

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

#### 配置自定义的域

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

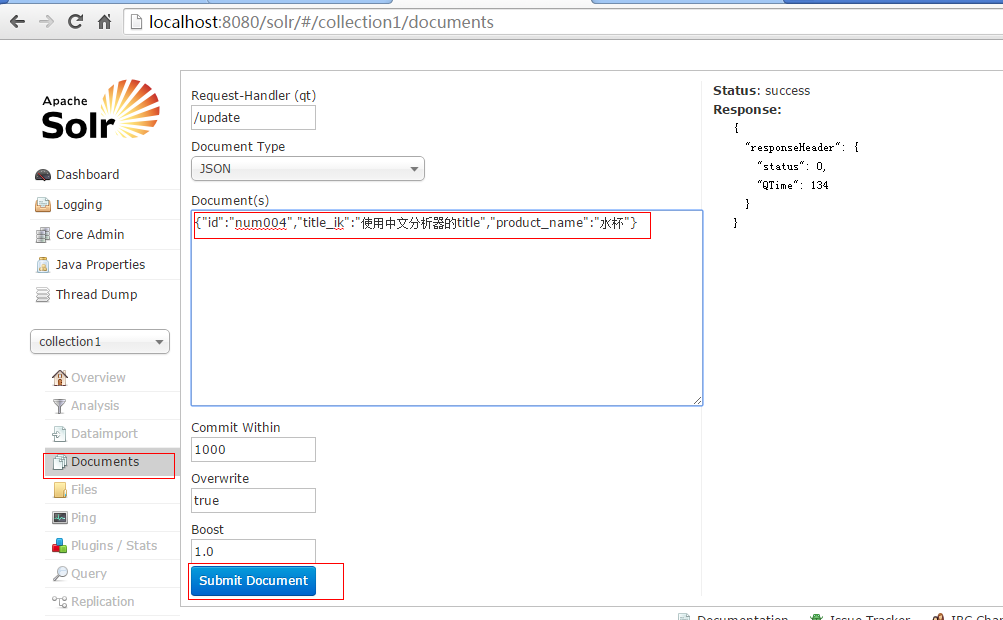
重启tomcat



#### 淘淘商城商品信息搜索域定义

|  |
| --- |
| <!--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"/> |

## 添加文档



注意：每个文档必须有一个id域。而且域名必须在schema.xml中定义。

## 删除文档

### 根据id删除

|  |
| --- |
| <delete>  <id>num001</id>  </delete>  <commit/> |

### 根据查询删除

|  |
| --- |
| <delete>  <query>\*:\*</query>  </delete>  <commit/> |

查询语法完全支持Lucene的语句。

### 修改文档

本质就是先删除后添加。

可以再添加一个新的document对象，只要是id和被修改的document的id一致就可以。

## Dataimport插件

可以批量把数据库中的数据导入到索引库中。

### 需要的jar包



### 安装步骤：

第一步：把dataimport插件依赖的jar包添加到collection1\lib文件夹下。

第二步：把mysql的数据库驱动也放到collection1\lib文件夹下

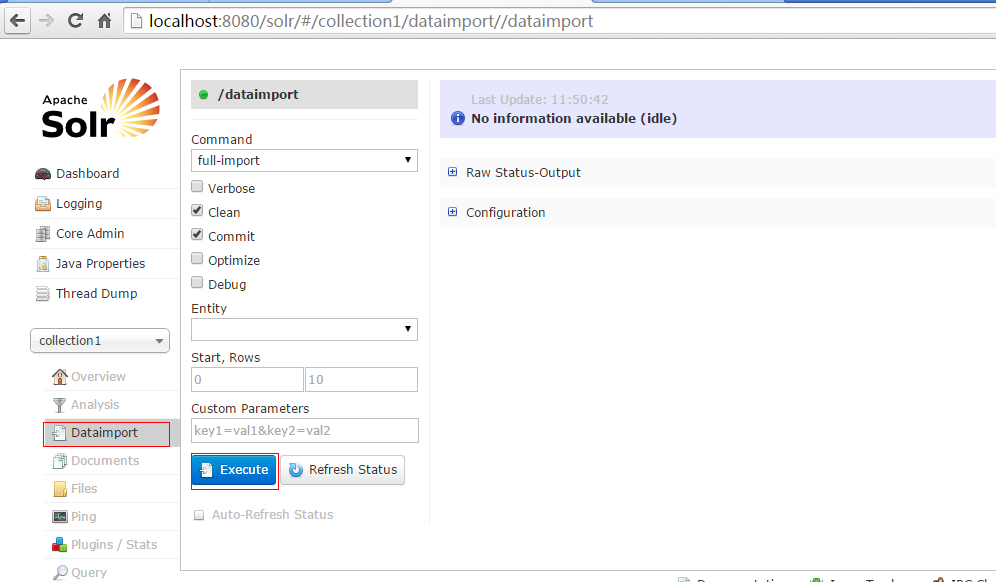
第三步：修改solrhome/collection1/conf/solrconfig.xml，添加一个requestHandler。

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

第四步：创建一个data-config.xml。目录和solrconfig.xml在同一个目录下collection1\conf。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <dataConfig>  <dataSource type="JdbcDataSource"  driver="com.mysql.jdbc.Driver"  url="jdbc:mysql://192.168.33.101:3306/taotao"  user="root"  password="root"/>  <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> |

第五步：重启tomcat后，进入solr管理页面，执行数据导入



## 索引库的查询

q：主查询条件。完全支持lucene语法。还进行了扩展。

fq：过滤查询。是在主查询条件查询结果的基础上进行过滤。

sort：排序条件。排序的域asc。如果有多个排序条件使用半角逗号分隔。

start, rows：分页处理。Start起始记录rows每页显示的记录条数。

fl：返回结果中域的列表。使用半角逗号分隔。

df：默认搜索域

wt：响应结果的数据格式，可以是json、xml等。

hl：开启高亮显示。

hl.fl：要高亮显示的域。

hl.simple.pre：高亮显示的前缀

hl.simple.post：高亮显示的后缀

# SolrJ客户端

可以实现对索引库的增删改查操作。

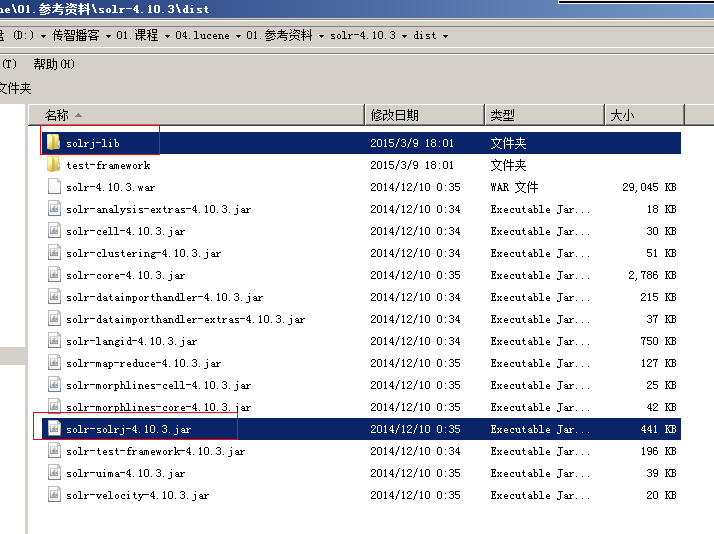
## 索引库的维护

### 添加文档

#### 使用步骤：

第一步：创建一java工程。

第二步：导入jar包。导入solrj的jar 包。





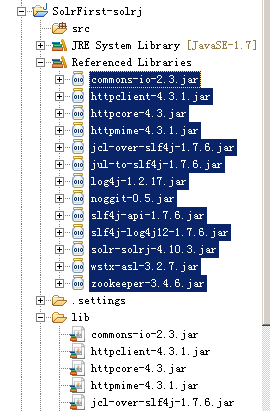
第三步：创建SolrServer对象和服务端建立连接。HttpSolrServer子类来完成。集群环境使用CloudSolrServer。

第四步：创建一文档对象。SolrInputDocument。

第五步：向文档对象中添加域。使用addField添加域。要求必须有id域，而且每个域必须在schema.xml中定义。

第六步：使用solrServer对象把文档提交到服务器。

#### 代码实现



|  |
| --- |
| //添加文档对象  @Test  **public** **void** addDocument() **throws** Exception {  //创建一个SolrServer对象  //参数：solr服务器的url  SolrServer server = **new** HttpSolrServer("http://localhost:8080/solr");  //创建文档对象  SolrInputDocument document = **new** SolrInputDocument();  //添加域  document.addField("id", "num001");  document.addField("title\_ik", "巧手DIY彩帘");  //把document对象写入索引库  server.add(document);  //提交修改  server.commit();    } |

### 删除文档

#### 根据id删除文档

|  |
| --- |
| //根据id删除文档  @Test  **public** **void** deleteDocumentById() **throws** Exception {  SolrServer server = **new** HttpSolrServer("http://localhost:8080/solr");  //删除文档  server.deleteById("num001");  //提交修改  server.commit();  } |

#### 根据查询删除文档

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

### 修改文档

Solrj并没有一个方法可以修改文档。还是使用add方法。只需要添加一个新的文档，保证新文档的id和被修改文档的id一致即可。

本质先删除后添加。

## 查询索引库

### 实现步骤

第一步：创建一个solrServer对象。

第二步：创建一个solrQuery对象。

第三步：向solrQuery对象中添加查询条件。

第四步：执行查询。返回文档列表。

第五步：遍历列表。

### 代码实现

|  |
| --- |
| //简单查询  @Test  **public** **void** queryIndex() **throws** Exception {  SolrServer server = **new** HttpSolrServer("http://localhost:8080/solr");  //创建一个查询对象  SolrQuery query = **new** SolrQuery();  //添加查询条件  //query.setQuery("\*:\*");  query.set("q", "\*:\*");    //执行查询  QueryResponse response = server.query(query);  //取文档列表  SolrDocumentList solrDocumentList = response.getResults();  System.*out*.println("查询结果的总数量：" + solrDocumentList.getNumFound());  //遍历列表  **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"));  }    } |

# 综合案例

## 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、价格范围搜索商品信息，也可以根据价格进行排序。

## 需求分析

### 开发需要的文档

1. 数据库的表结构。
2. 页面原型图。
3. 业务流程图

### 页面原型分析



### 流程图

Tomcat

Tomcat

当展示商品详细页时需要查询数据库

查询商品列表

使用dataimport插件

Mysql数据库

索引库

Solr服务

表现层：

Jsp+Controller

springmvc

Dao层

数据层 SolrJ

业务逻辑层

Service

## Dao层

功能：查询solr服务返回一个商品列表。需要一个query对象，执行query对象进行查询，返回商品列表。

参数：SolrQuery query

返回值：ResultModel 包含商品列表

商品的pojo：

|  |
| --- |
| **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Pojo：

|  |
| --- |
| **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;  } |

方法定义：

ResultModel queryProduct(SolrQuery query) throws Exception;

## Service

功能：接收controller传递过来的参数。创建一SolrQuery对象，拼装查询条件调用Dao层执行查询返回一个ResultModel对象。还需要计算商品列表的总页数。

参数：

1. 查询条件：String queryString
2. 商品分类名称：String catalog\_name
3. 价格区间过滤条件：使用一个字符串来描述一个区间例如：

\*-10,10-20,20-30,30-\*。

String price

1. 排序条件：只需要接收一个排序的方式就可以了。0：升序1：降序。 String sort
2. 分页条件：接收一个页码是一个Integer数据。需要我们确定每页显示商品的数量。可以定义在常量或者配置文件。每页显示60商品。Integer page

返回值：ResultModel

方法定义：

ResultModel queryProduct(String queryString, String catalog\_name, String price,String sort, Integer page);

## Controller

功能：接收页面传递过来的参数，调用service查询商品列表。把查询结果传递给页面。还需要参数回显。

参数：

1. 查询条件：String queryString
2. 商品分类名称：String catalog\_name
3. 价格区间过滤条件：使用一个字符串来描述一个区间例如：

\*-10,10-20,20-30,30-\*。

String price

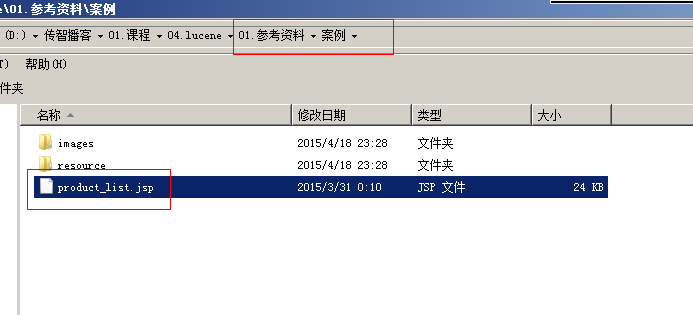
1. 排序条件：只需要接收一个排序的方式就可以了。0：升序1：降序。 String sort
2. 分页条件：接收一个页码是一个Integer数据。需要我们确定每页显示商品的数量。可以定义在常量或者配置文件。每页显示60商品。Integer page
3. Model，传递参数使用。

返回值：String（逻辑视图名）

方法定义：

Public String queryProduct(String queryString, String catalog\_name, String price,String sort, Integer page, Model model)

## Jsp



## 代码实现

### 实现步骤

第一步：创建一web工程

第二步：导入jar包。需要springmvc、spring、solrj、solrj依赖的jar、日志相关的jar包。

第三步：编写dao

第四步：编写Service

第五步：编写Controller

第六步：配置前端控制器，创建springmvc.xml配置三大件。

### 代码实现

#### Dao

|  |
| --- |
| @Repository  **public** **class** ProductDaoImpl **implements** ProductDao {  @Autowired  **private** SolrServer solrServer;    @Override  **public** ResultModel queryProduct(SolrQuery query) **throws** Exception {    //根据query对象查询索引库  QueryResponse response = solrServer.query(query);  //取商品列表  SolrDocumentList documentList = response.getResults();  List<ProductModel> productList = **new** ArrayList<>();  **for** (SolrDocument solrDocument : documentList) {  //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 = response.getHighlighting();  List<String> list = highlighting.get(solrDocument.get("id")).get("product\_name");  **if** (list != **null** && list.size() > 0) {  productName = list.get(0);  } **else** {  productName = (String) solrDocument.get("product\_name");  }  productModel.setName(productName);  productModel.setCatalog\_name((String) solrDocument.get("product\_catalog\_name"));  productModel.setPrice((**float**) solrDocument.get("product\_price"));  productModel.setPicture((String) solrDocument.get("product\_picture"));    //添加到商品列表  productList.add(productModel);  }    //返回值对象  ResultModel resultModel = **new** ResultModel();  resultModel.setProductList(productList);  //查询结果总数量  resultModel.setRecordCount(documentList.getNumFound());    **return** resultModel;  }  } |

#### Service

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

#### Controller

|  |
| --- |
| @Controller  **public** **class** ProductController {  @Autowired  **private** ProductService productService;  @RequestMapping("/list")  **public** String queryProduct(String queryString, String catalog\_name,  String price, String sort, Integer page, Model model) {  // 查询商品列表  ResultModel resultModel = **null**;  **try** {  resultModel = productService.queryProduct(queryString,  catalog\_name, price, sort, page);  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  // 传递给页面  model.addAttribute("result", resultModel);  // 参数回显  model.addAttribute("queryString", queryString);  model.addAttribute("catalog\_name", catalog\_name);  model.addAttribute("price", price);  model.addAttribute("sort", sort);  model.addAttribute("page", page);  // 返回jsp  **return** "product\_list";  }  } |

#### Springmvc

|  |
| --- |
| <?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.jd"* />  <!-- 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> |

#### 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>SolrJD0508</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  <!-- 配置前端控制器 -->  <!-- 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> |