Solr

全文检索服务器

# 课程计划

1. solr介绍
   1. 什么是solr
   2. Solr和lucene的区别
2. Solr的安装配置（重点）
3. Solr的基本使用（重点）
4. Solrj的使用（重点）
5. 京东案例（重点）

# Solr介绍

## 什么是solr

Solr是apache的顶级开源项目，它是使用java开发 ，基于lucene的全文检索服务器。

Solr比lucene提供了更多的查询语句，而且它可扩展、可配置，同时它对lucene的性能进行了优化。

Solr是如何实现全文检索的呢？

索引流程：solr客户端（浏览器、java程序）可以向solr服务端发送POST请求，请求内容是包含Field等信息的一个xml文档，通过该文档，solr实现对索引的维护（增删改）

搜索流程：solr客户端（浏览器、java程序）可以向solr服务端发送GET请求，solr服务器返回一个xml文档。

Solr同样没有视图渲染的功能。

## Solr和lucene的区别

Lucene是一个全文检索引擎工具包，它只是一个jar包，不能独立运行，对外提供服务。

Solr是一个全文检索服务器，它可以单独运行在servlet容器，可以单独对外提供搜索和索引功能。Solr比lucene在开发全文检索功能时，更快捷、更方便。



# Solr安装配置

## 下载solr

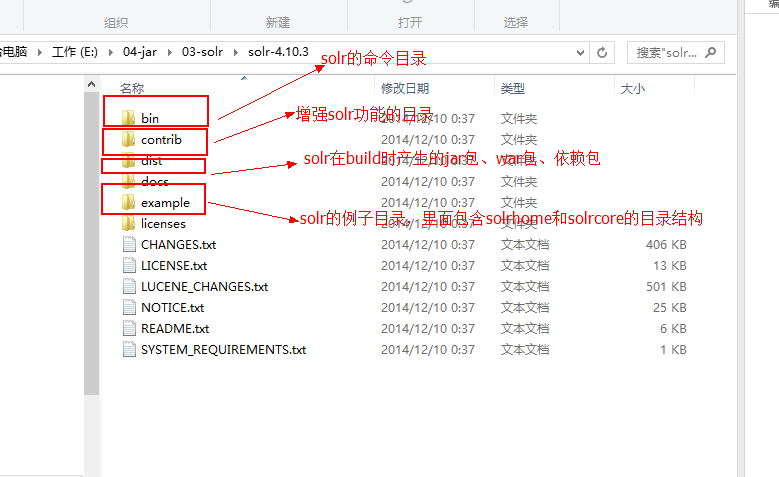
Solr和lucene的版本是同步更新的，最新的版本是5.2.1

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

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



**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

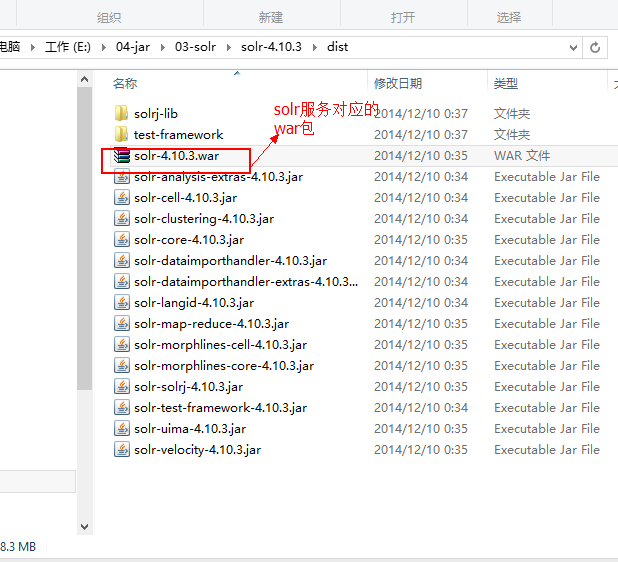
* **example/multicore**：

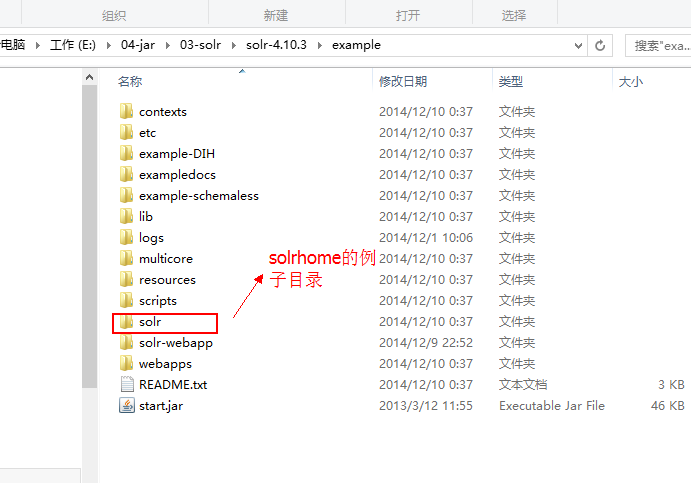
该目录包含了在Solr的multicore中设置的多个Core目录。

* **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

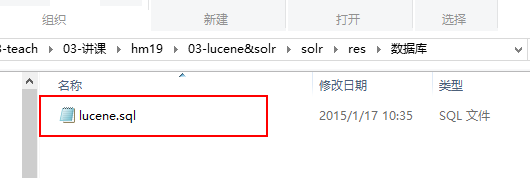




## 运行环境

* Jdk：1.7及以上
* Solr：4.10.3
* Mysql：5X
* Web服务器：tomcat 7

### 初始化数据库脚本

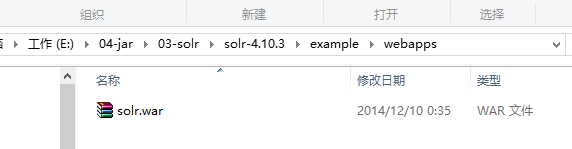


## Solr安装配置

### Solr的安装部署

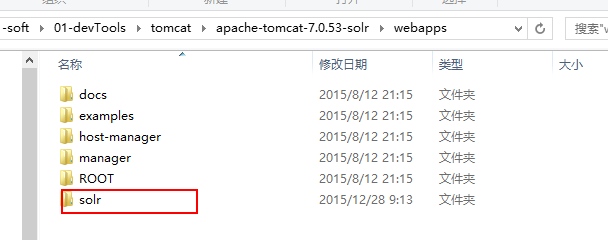
第一步：安装tomcat

第二步：将以下的war包，拷贝到tomcat的webapps目录下

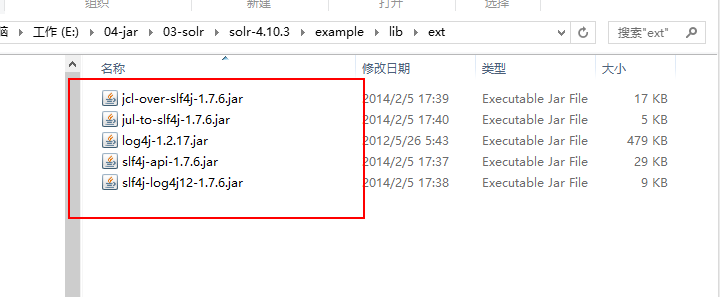


第三步：解压缩war包

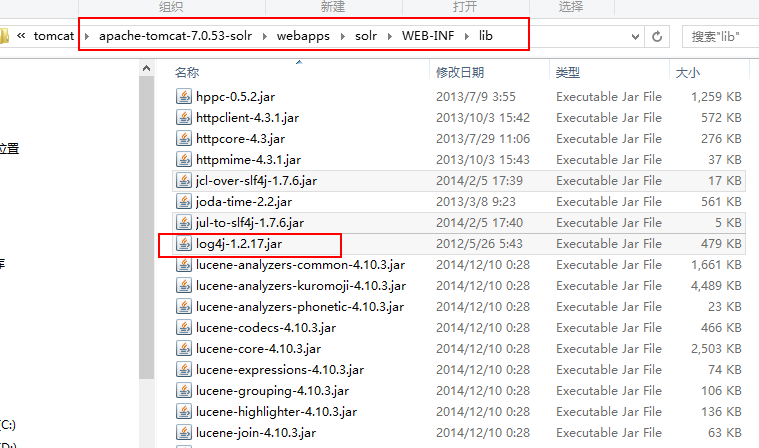
解压缩之后，将war包删掉



第四步：添加solr的扩展服务包

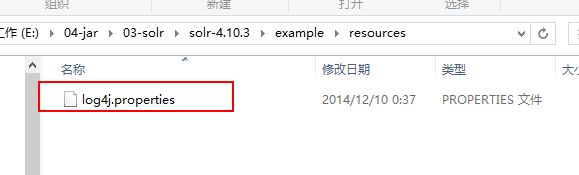


将以上jar包，添加到以下目录

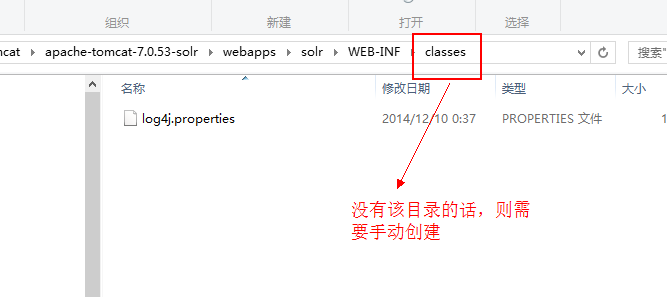


第五步：添加log4j.properties

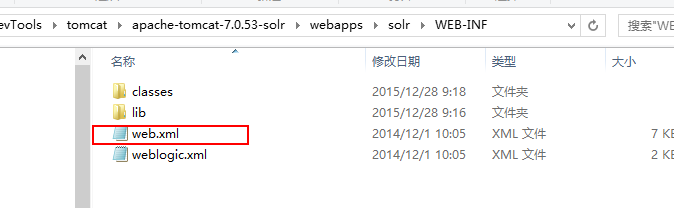
将以下目录的文件进行拷贝

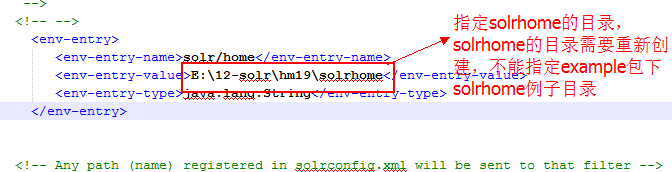


复制到以下目录



第六步：在web.xml中指定solrhome的目录





### Solrcore的安装

#### Solrcore和solrhome

Solrhome是solr服务运行的主目录，一个solrhome目录里面包含多个solrcore目录，一个solrcore目录里面了一个solr实例运行时所需要的配置文件和数据文件。

每一个solrcore都可以单独对外提供搜索和索引服务。

多个solrcore之间没有关系。

#### Solrcore和solrhome的目录结构

Solrhome的目录结构



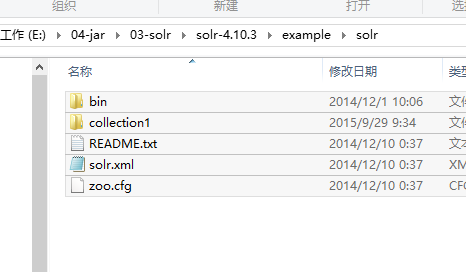
Solrcore目录



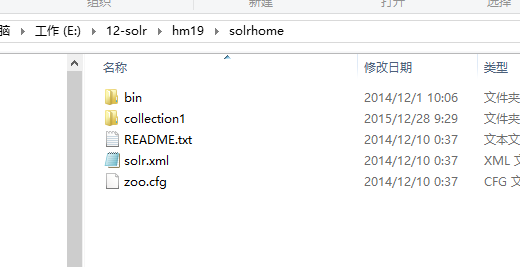
#### Solrcore的安装

安装solrcore需要先安装solrhome

将以下目录的文件进行拷贝



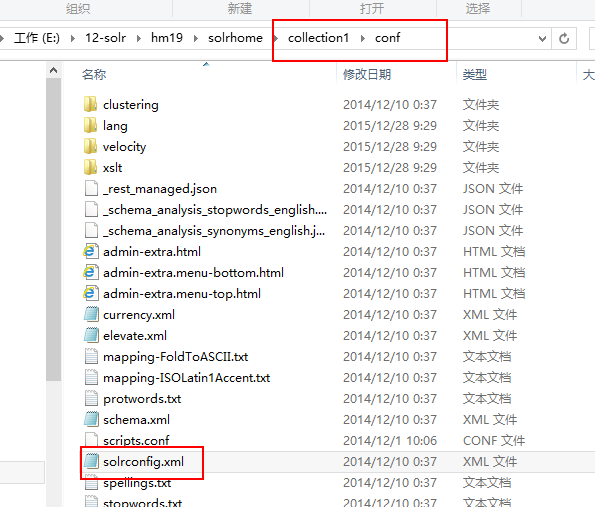
复制到以下目录



这样solrhome和solrcore就安装成功了。

#### Solrcore配置

在solrcore的conf目录下，有一个solrconfig.xml的配置文件，该配置文件，配置来solrcor的运行信息



在该文件中，主要配置三个标签：lib标签、datadir标签、requestHandler标签

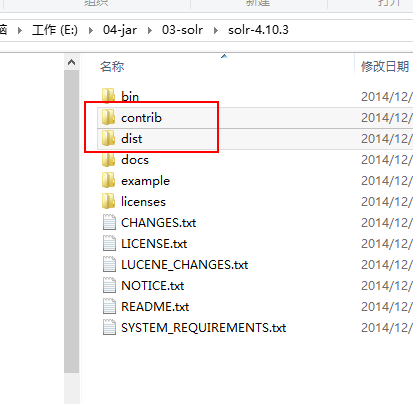
如果对该文件不进行配置也可以，即使用默认的配置项。

##### Lib 标签

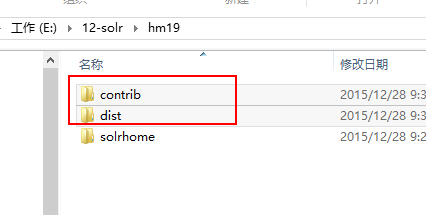
Solrcore需要添加一个扩展依赖包，通过lib标签来指定依赖包的地址

solr.install.dir：表示solrcore的安装目录

将以下目录的文件进行拷贝



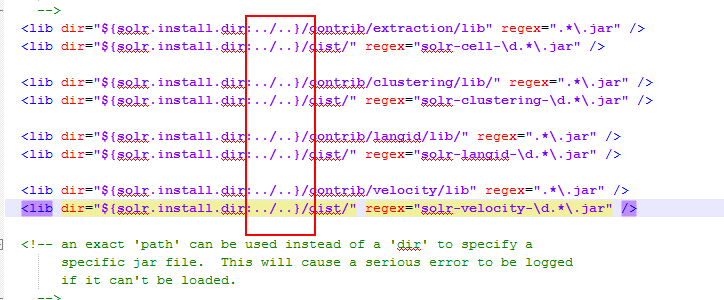
复制到以下目录



修改lib标签

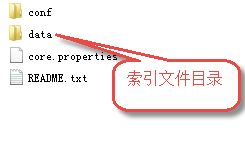
如果之前在web.xml 中配置了solrhome的位置。这里就需要修改

${solr.install.dir:../..} 为 solrhome相对于后面的/contrib/extraction/lib这个目录需要前进或者后退几级才能找到后面目录的表达式



##### 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-->

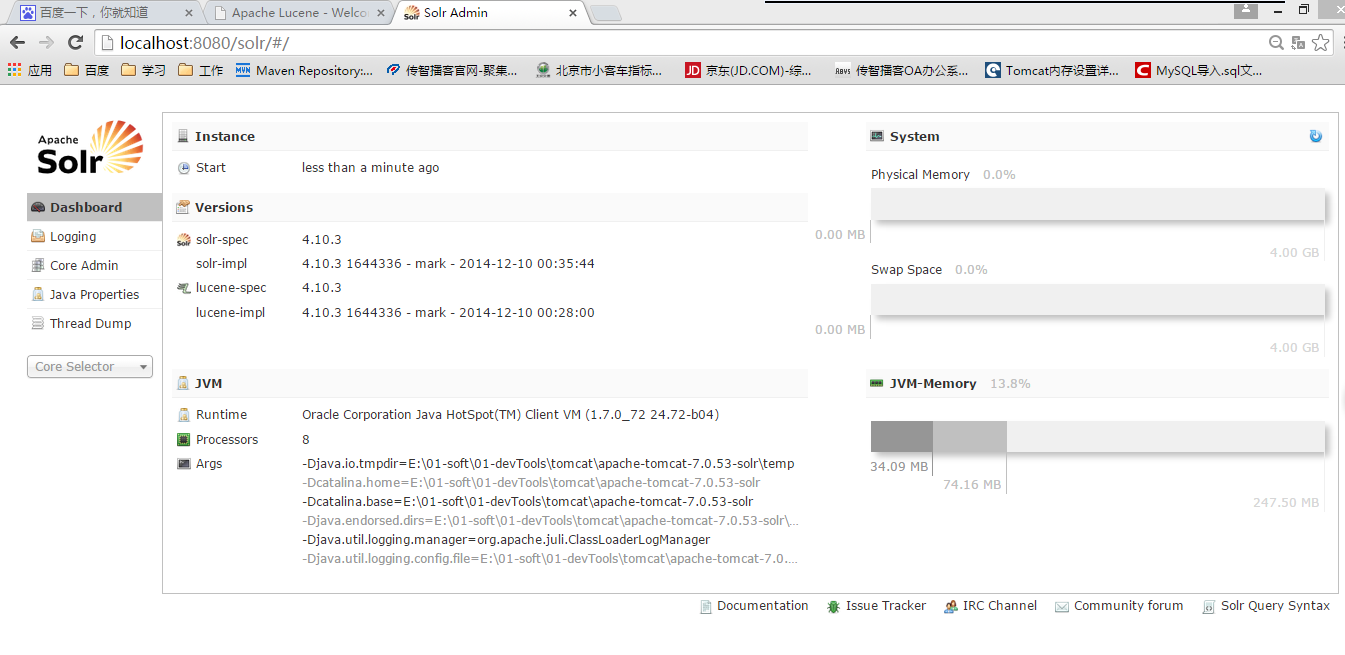
</lst>

</requestHandler>

## solr界面介绍

启动solr服务

<http://localhost:8080/solr>



### 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

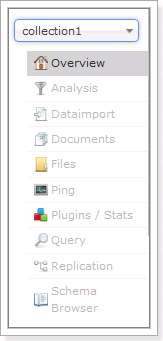
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

### 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

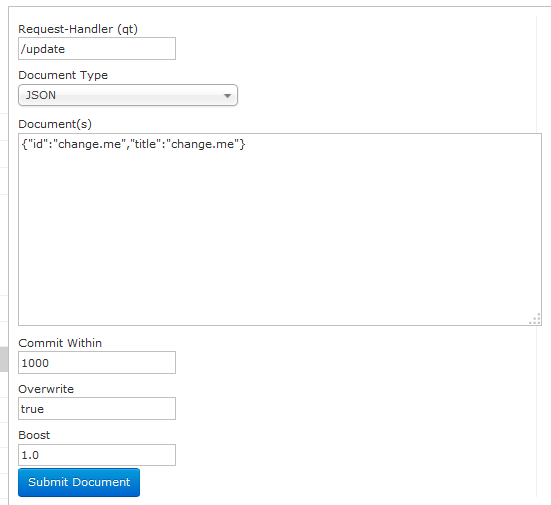
可以定义数据导入处理器，从关系数据库将数据导入到Solr索引库中。

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。



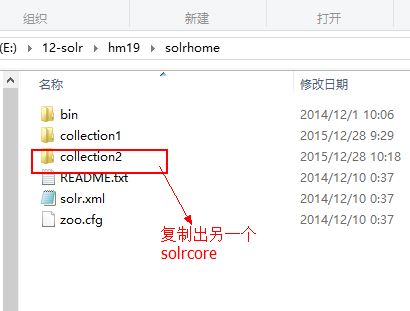
## 多solrcore的配置

配置多solrcore的好处：

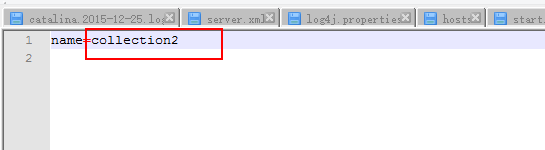
1. 在进行solrcloud的时候，必须配置多solrcore
2. 每个solrcore之间是独立的，都可以单独对外提供服务。不同的业务模块可以使用不同的solrcore来提供搜索和索引服务。

添加

第一步：复制solrhome下的collection1目录到本目录下，修改名称为collection2



第二步：修改solrcore目录下的core.properties

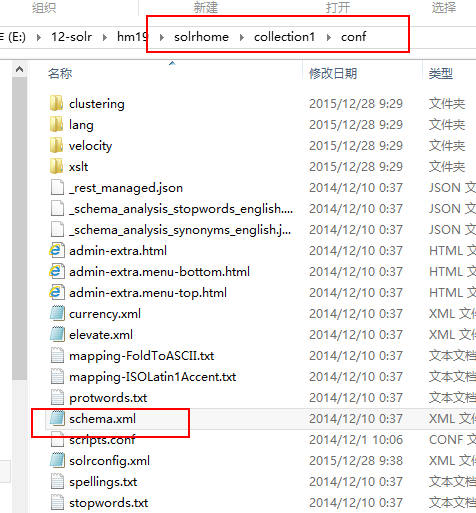


这样多solrcore就配置完成了。

# Solr的基本使用

## Schema.xml

在schema.xml文件中，主要配置了solrcore的一些数据信息，包括Field和FieldType的定义等信息，**在solr中，Field和FieldType都需要先定义后使用**。



### Filed

定义Field域

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

Name：指定域的名称

Type：指定域的类型

Indexed：是否索引

Stored：是否存储

Required：是否必须

multiValued：是否多值，比如商品信息中，一个商品有多张图片，一个Field像存储多个值的话，必须将multiValued设置为true。

### dynamicField

动态域

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

Name：指定动态域的命名规则

### uniqueKey

指定唯一键

<uniqueKey>**id**</uniqueKey>

其中的id是在Field标签中已经定义好的域名，而且该域要设置为required为true。

一个schema.xml文件中必须有且仅有一个唯一键

### copyField

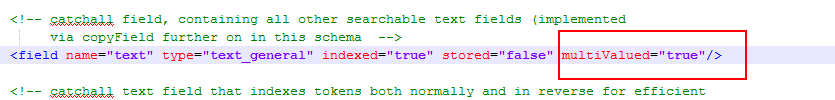
复制域

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

Source：要复制的源域的域名

Dest：目标域的域名

由dest指的的目标域，必须设置multiValued为true。



### FieldType

定义域的类型

|  |
| --- |
| <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：指定域类型的名称

Class：指定该域类型对应的solr的类型

Analyzer：指定分析器

Type：index、query，分别指定搜索和索引时的分析器

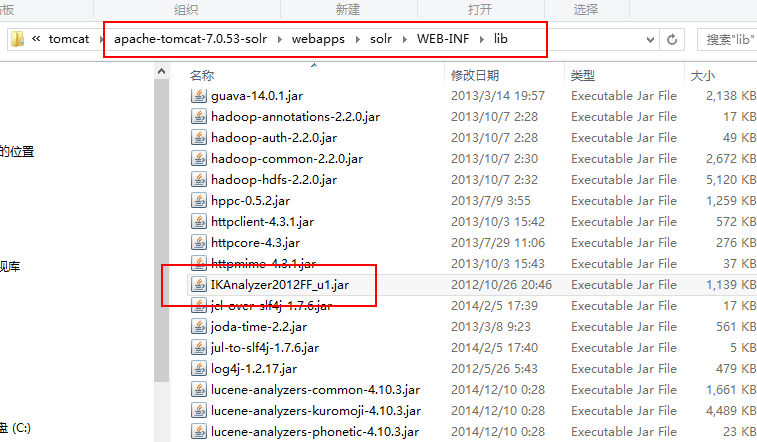
Tokenizer：指定分词器

Filter：指定过滤器

## 中文分词器

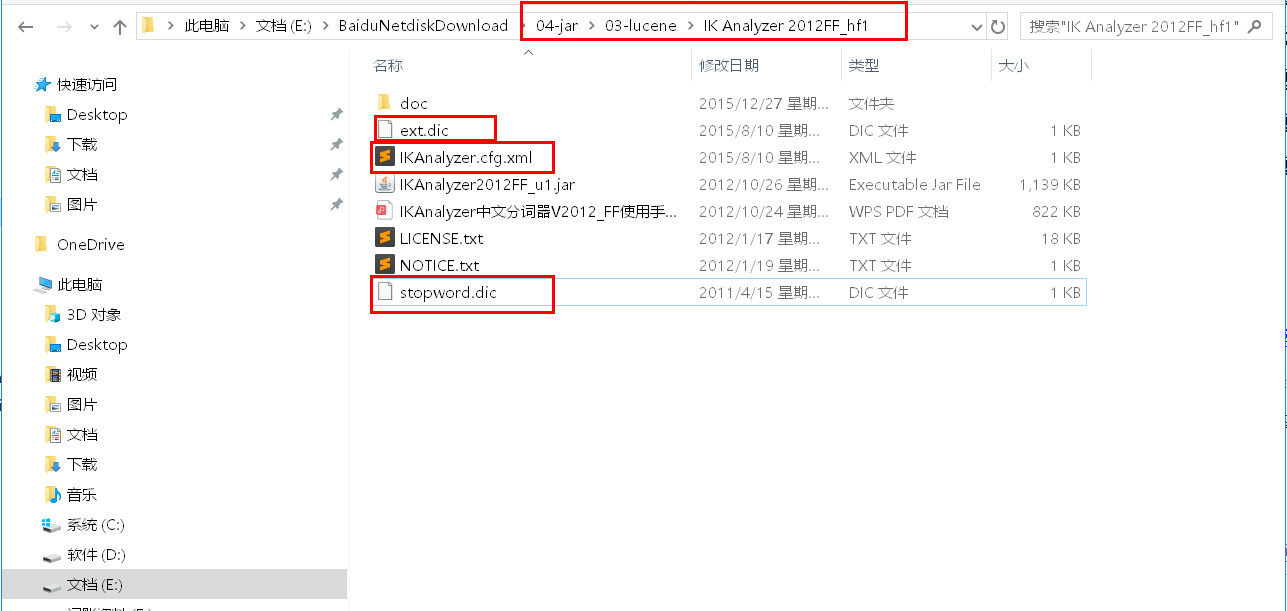
使用ikanalyzer进行中文分词

第一步：将ikanalyzer的jar包拷贝到以下目录

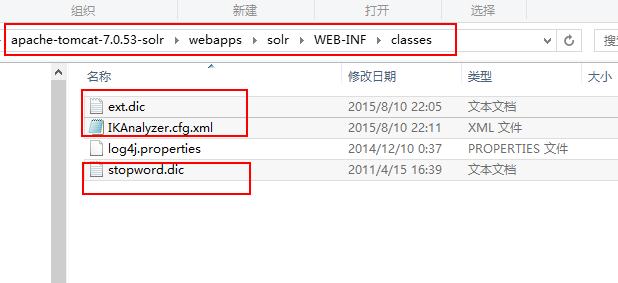


第二步：将ikanalyzer的扩展词库的配置文件拷贝到 目录

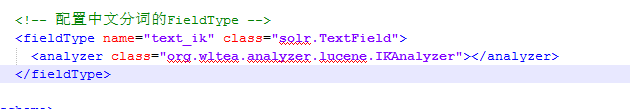
配置文件所在位置



复制到目标 位置



第三步：配置FieldType



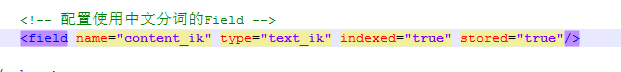
<!-- IKAnalyzer-->

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

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

</fieldType>

第四步：配置使用中文分词的Field



<!--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



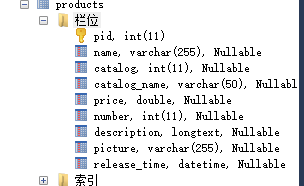
## 配置业务Field

### 需求

对京东案例中的products表的数据进行索引，所以需要先定义对应的Field域。

### 分析配置

Products的表结构



需要往索引库添加的字段有：

pid、name、catalog、catalog\_name、price、description、picture

**FieldType：**

经分析，由于中文分词器已经配置完FieldType 【上面中文分词器的第三步配置的text\_ik】，所以目前FieldType已经满足需要，无需配置。

**Field：**

Pid：

由于pid在products表中是唯一键，而且在solr的shema.xml中已有一个id的唯一键配置，所以不需要再重新定义pid域。

<!--配置京东案例的field -->

<!--商品名称 -->

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

<!-- 商品分类ID -->

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

<!-- 商品分类名称 -->

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

<!-- 商品价格 -->

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

<!-- 商品描述 -->

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

<!-- 商品图片 -->

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

<!-- 目标域 -->

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

<!-- 将商品名称添加到目标域 -->

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

<!-- 将商品描述添加到目标域 -->

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



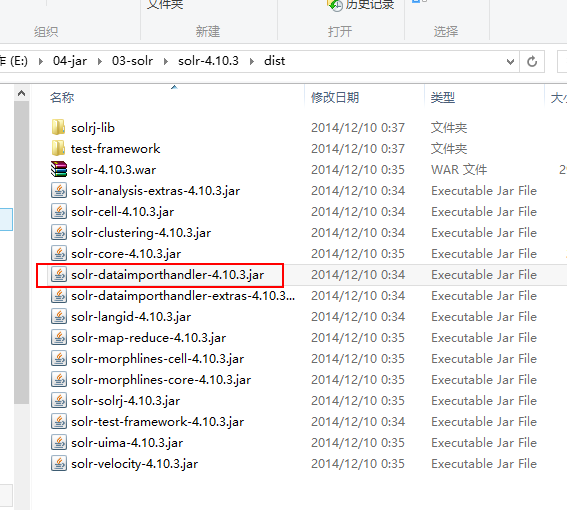
## Dataimport

该插件可以将数据库中指定的sql语句的结果导入到solr索引库中。

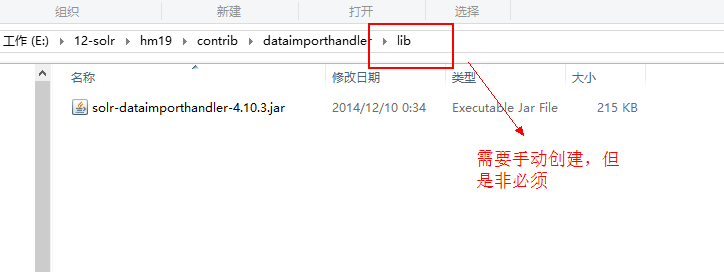
### 第一步：添加jar包

* Dataimport的jar包

复制以下目录的jar包（此jar包版本应与solr版本保持一致）



添加到以下目录

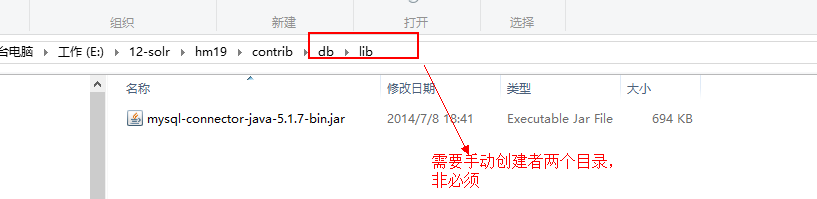


修改solrconfig.xml文件，添加lib标签

<lib dir="${solr.install.dir:../..}/contrib/dataimporthandler/lib" regex=".\*\.jar" />

* MySQL数据库驱动包

将mysql的驱动包，复制到以下目录



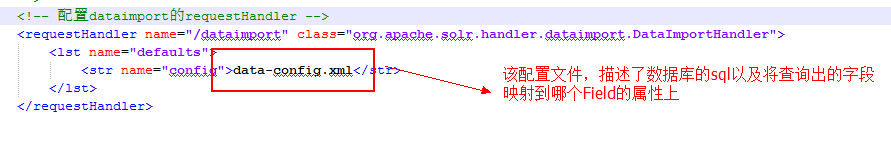
修改solrconfig.xml文件，添加lib标签

<lib dir="${solr.install.dir:../..}/contrib/db/lib" regex=".\*\.jar" />

### 第二步：配置requestHandler

在solrconfig.xml中，添加一个dataimport的requestHandler

data-config.xml 的位置与solrconfig.xml 平级



<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data-config.xml

在solrconfig.xml同级目录下，创建data-config.xml



<?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"/>

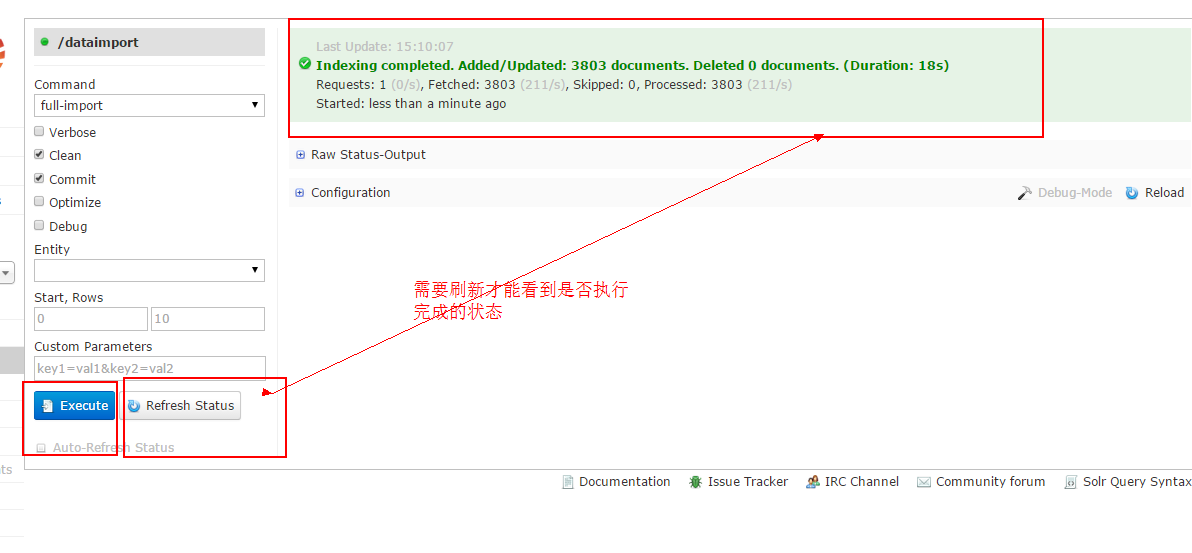
<field column="picture" name="product\_picture"/>

</entity>

</document>

</dataConfig>

### 重启tomcat



# Solrj的使用

## 什么是solrj

Solrj就是solr服务器的java客户端。

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

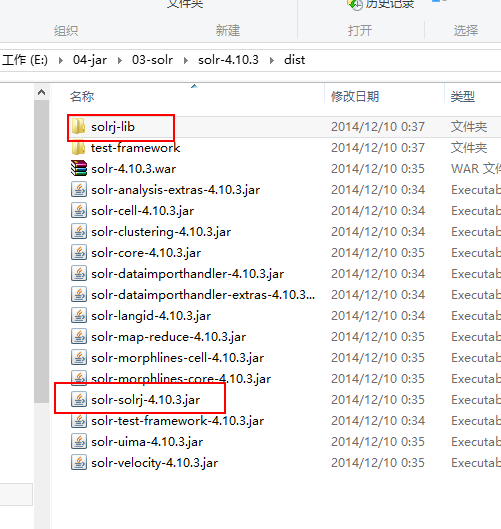
最终Solr完在索引和搜索

## 环境准备

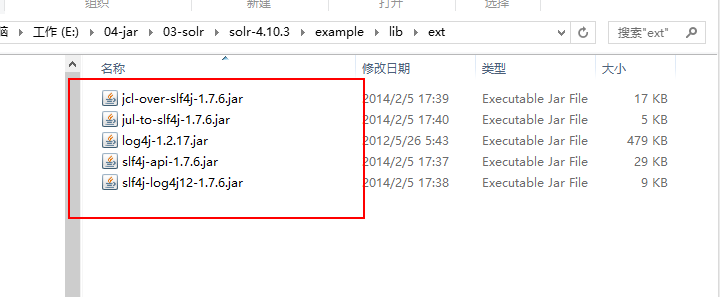
* Jdk
* Ide
* Tomcat
* Solrj

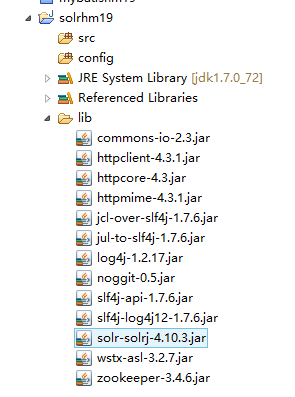
## 搭建工程

* Solrj的依赖包和核心包



* Solr的扩展服务包

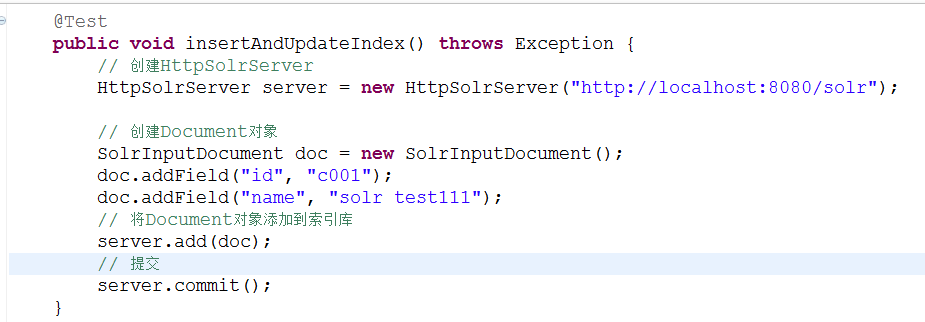




## 使用solrj完成索引维护

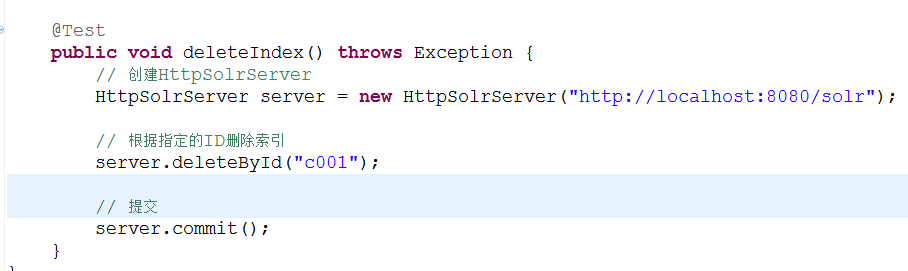
### 添加/修改索引

在solr中，索引库中都会存在一个唯一键，如果一个Document的id存在，则执行修改操作，如果不存在，则执行添加操作。



### 删除索引

#### 根据指定ID来删除

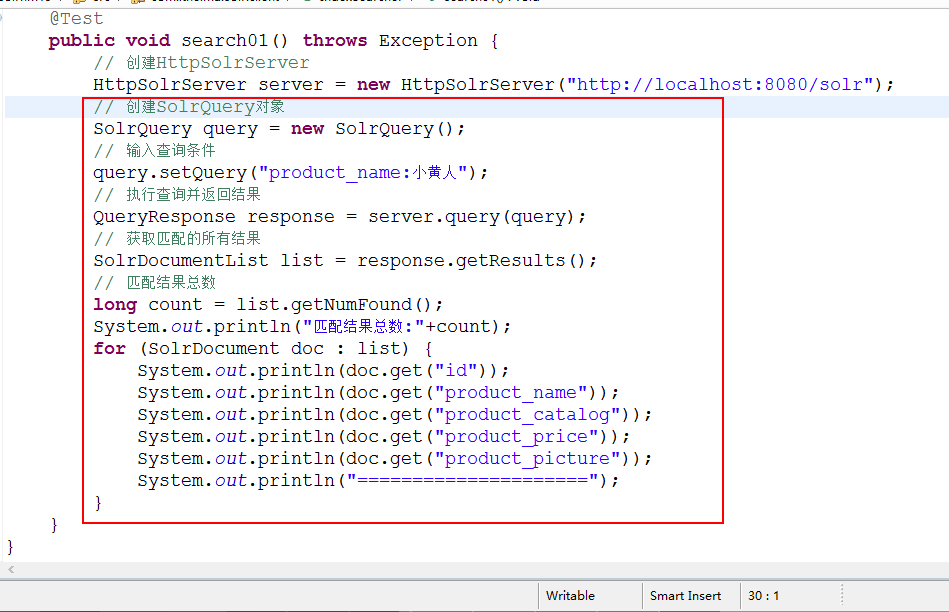


#### 根据条件删除



### 查询索引

#### 简单查询

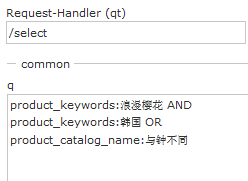


#### 复杂查询

##### 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



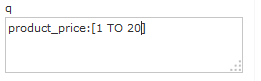
1. fq - （filter query）过虑查询，作用：在q查询符合结果中同时是fq查询符合的，例如：：

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。

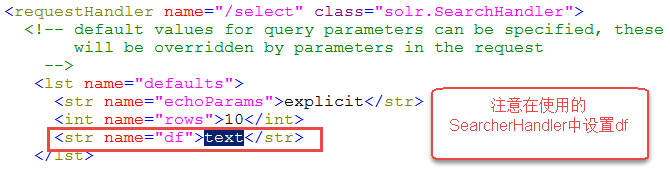


显示商品图片、商品名称、商品价格

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。



##### 代码

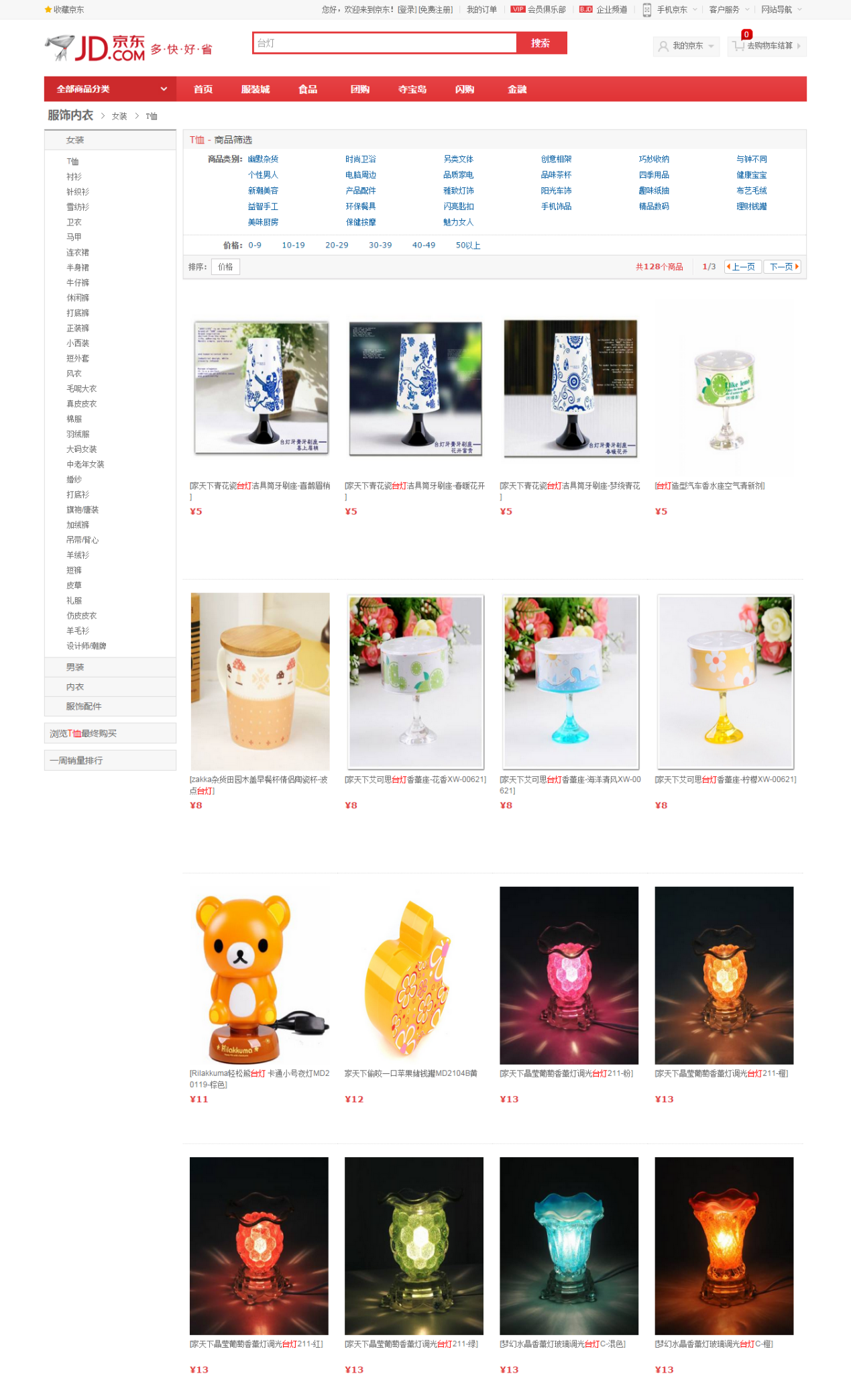
|  |
| --- |
| @Test  **public** **void** search02() **throws** Exception {  // 创建HttpSolrServer  HttpSolrServer server = **new** HttpSolrServer("http://localhost:8080/solr");  // 创建SolrQuery对象  SolrQuery query = **new** SolrQuery();  // 输入查询条件  query.setQuery("product\_name:小黄人");  // query.set("q", "product\_name:小黄人");  // 设置过滤条件  // 如果设置多个过滤条件的话，需要使用query.addFilterQuery(fq)  query.setFilterQueries("product\_price:[1 TO 10]");  // 设置排序  query.setSort("product\_price", ORDER.*asc*);  // 设置分页信息（使用默认的）  query.setStart(0);  query.setRows(10);  // 设置显示的Field的域集合  query.setFields("id,product\_name,product\_catalog,product\_price,product\_picture");  // 设置默认域  query.set("df", "product\_keywords");  // 设置高亮信息  query.setHighlight(**true**);  query.addHighlightField("product\_name");  query.setHighlightSimplePre("<em>");  query.setHighlightSimplePost("</em>");  // 执行查询并返回结果  QueryResponse response = server.query(query);  // 获取匹配的所有结果  SolrDocumentList list = response.getResults();  // 匹配结果总数  **long** count = list.getNumFound();  System.*out*.println("匹配结果总数:" + count);  // 获取高亮显示信息  Map<String, Map<String, List<String>>> highlighting = response  .getHighlighting();  **for** (SolrDocument doc : list) {  System.*out*.println(doc.get("id"));  List<String> list2 = highlighting.get(doc.get("id")).get(  "product\_name");  **if** (list2 != **null**)  System.*out*.println("高亮显示的商品名称：" + list2.get(0));  **else** {  System.*out*.println(doc.get("product\_name"));  }  System.*out*.println(doc.get("product\_catalog"));  System.*out*.println(doc.get("product\_price"));  System.*out*.println(doc.get("product\_picture"));  System.*out*.println("=====================");  }  } |

# 京东案例

## 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，同时还可以分页。

界面如下：



## 分析

### UI分析



### 架构分析

应用服务器服务端：

表现层：使用springmvc接收前台搜索页面的查询条件等信息

业务层：调用dao层完成数据库持久化

如果数据库数据发生变化，调用solrj的客户端同步索引库。

Dao层：使用mybatis完成数据库持久化

Solrj服务器：

提供搜索和索引服务

数据库服务器：

提供数据库服务

spring容器

表现层

springmvc

Service层

Dao层

Solr索引库

mysql数据库

商品信息

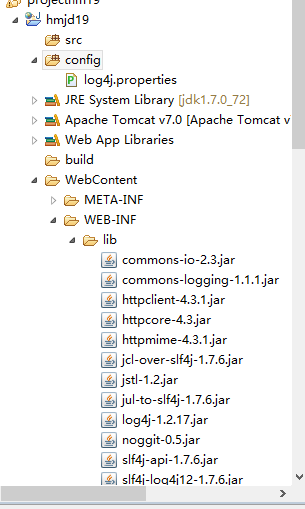
Tomcat

Solr服务

索引、搜索请求

## 工程搭建

* Solrj的jar包
* Solr的扩展包
* Springmvc的包



## 代码实现

### Pojo

Product.java

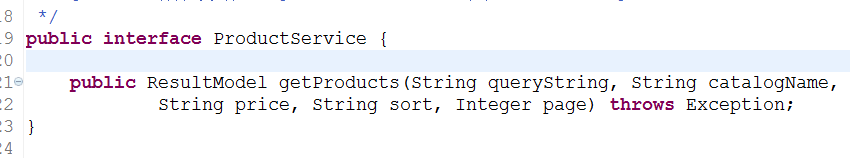
|  |
| --- |
| **public** **class** Product {  // 商品编号  **private** String pid;  // 商品名称  **private** String name;  // 商品分类名称  **private** String catalog\_name;  // 价格  **private** **float** price;  // 商品描述  **private** String description;  // 图片名称  **private** String picture;  } |

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;  } |

### Service

Service接口



Service实现类

|  |
| --- |
| @Service  **public** **class** ProductServiceImpl **implements** ProductService {  // 依赖注入HttpSolrServer  @Autowired  **private** HttpSolrServer server;  @Override  **public** ResultModel getProducts(String queryString, String catalogName,  String price, String sort, Integer page) **throws** Exception {  // 创建SolrQuery对象  SolrQuery query = **new** SolrQuery();  // 输入关键字  **if** (StringUtils.*isNotEmpty*(queryString)) {  query.setQuery(queryString);  } **else** {  query.setQuery("\*:\*");  }  // 输入商品分类过滤条件  **if** (StringUtils.*isNotEmpty*(catalogName)) {  query.addFilterQuery("product\_catalog\_name:" + catalogName);  }  // 输入价格区间过滤条件  // price的值：0-9 10-19  **if** (StringUtils.*isNotEmpty*(price)) {  String[] ss = price.split("-");  **if** (ss.length == 2) {  query.addFilterQuery("product\_price:[" + ss[0] + " TO " + ss[1]  + "]");  }  }  // 设置排序  **if** ("1".equals(sort)) {  query.setSort("product\_price", ORDER.*desc*);  } **else** {  query.setSort("product\_price", ORDER.*asc*);  }  // 设置分页信息  **if** (page == **null**)  page = 1;  query.setStart((page - 1) \* 20);  query.setRows(20);  // 设置默认域  query.set("df", "product\_keywords");  // 设置高亮信息  query.setHighlight(**true**);  query.addHighlightField("product\_name");  query.setHighlightSimplePre("<font style=\"color:red\" >");  query.setHighlightSimplePost("</font>");  QueryResponse response = server.query(query);  // 查询出的结果  SolrDocumentList results = response.getResults();  // 记录总数  **long** count = results.getNumFound();  List<Products> products = **new** ArrayList<>();  Products prod;  // 获取高亮信息  Map<String, Map<String, List<String>>> highlighting = response  .getHighlighting();  **for** (SolrDocument doc : results) {  prod = **new** Products();  // 商品ID  prod.setPid(doc.get("id").toString());  List<String> list = highlighting.get(doc.get("id")).get(  "product\_name");  // 商品名称  **if** (list != **null**)  prod.setName(list.get(0));  **else** {  prod.setName(doc.get("product\_name").toString());  }  // 商品价格  prod.setPrice(Float.*parseFloat*(doc.get("product\_price").toString()));  // 商品图片地址  prod.setPicture(doc.get("product\_picture").toString());  products.add(prod);  }  // 封装ResultModel对象  ResultModel rm = **new** ResultModel();  rm.setProductList(products);  rm.setCurPage(page);  rm.setRecordCount(count);  **int** pageCount = (**int**) (count / 20);  **if** (count % 20 > 0)  pageCount++;  // 设置总页数  rm.setPageCount(pageCount);  **return** rm;  }  } |

### 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);

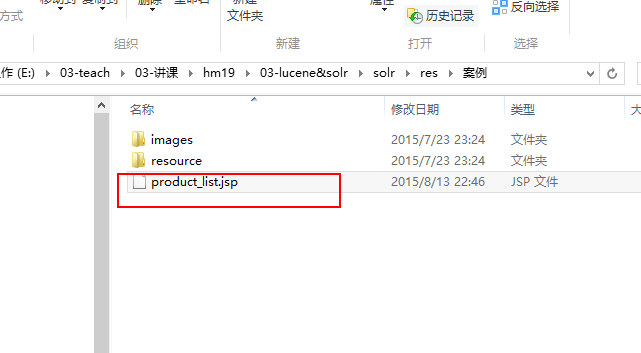
**return** "product\_list";

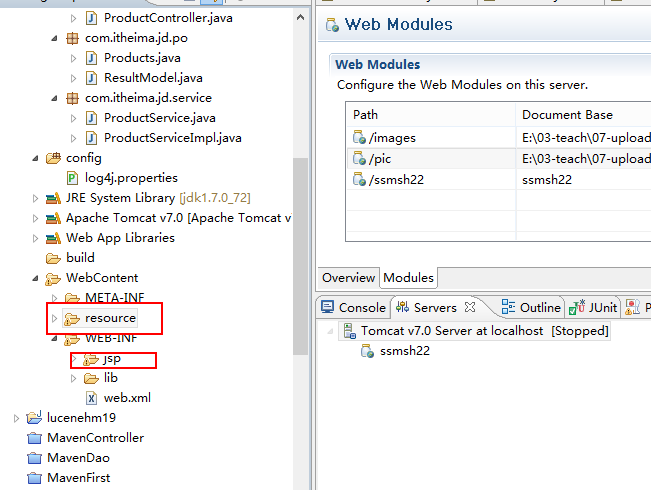
}

}

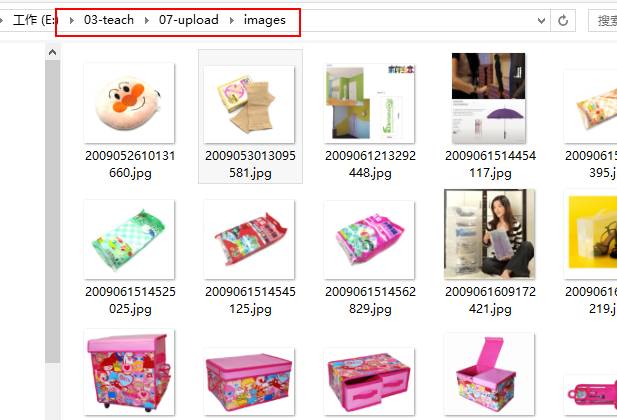
#### Jsp和静态资源

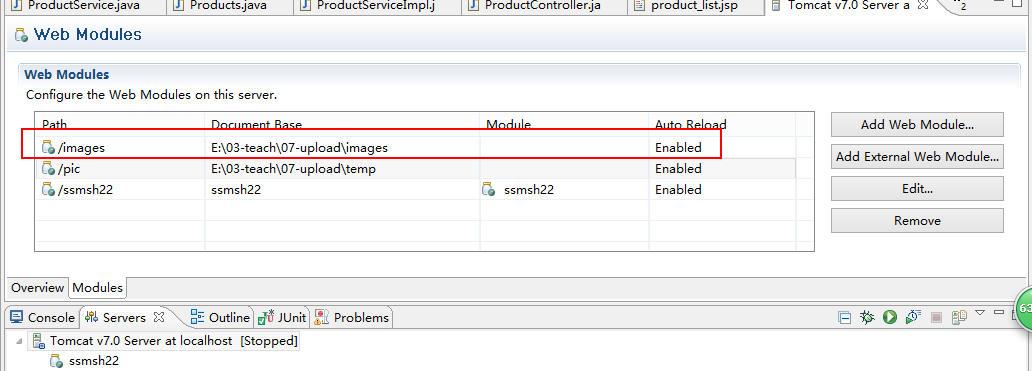
从资料中拷贝





图片信息放到以下目录





#### 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>/</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> |

#### 配置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: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>