**品优购电商系统开发**

**第10章**

**全文检索solr**

传智播客.黑马程序员

# 课程目标

目标1：理解什么是solr

目标2：完成安装配置solr

目标3：完成solr的基本使用

* 1. schema.xml文件
  2. 配置中文分词器
  3. 配置业务域

目标4：会使用solrJ

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

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。Solr不提供构建页面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。

Solr是一个可以独立运行的搜索服务器，使用solr进行全文检索服务的话，只需要通过http请求访问该服务器即可。

## 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。



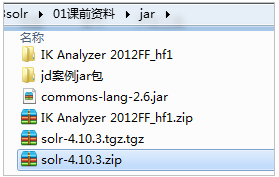
# Solr安装配置

## 下载solr

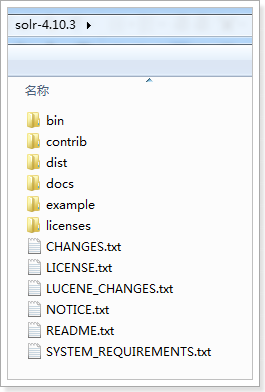
Solr和lucene的版本是同步更新的，本课程使用的版本：4.10.3

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

Linux下需要solr-4.10.3.tgz，windows下需要solr-4.10.3.zip。



解压solr-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

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），

使用jetty启动：

使用cmd命令行，进入example文件夹启动

启动命令java -jar start.jar



启动后访问地址：http://127.0.0.1:8983/solr

但是企业中一般使用Tomcat作为服务器，本课程也是一样，

相关环境如下：

* Solr：4.10.3
* Jdk环境：1.7（solr4.10 不能使用jdk1.7以下）
* 服务器：Tomcat 7

## SolrCore配置

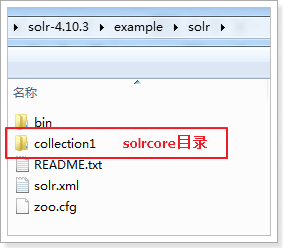
### SolrHome和SolrCore

SolrHome是Solr服务运行的主目录，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。

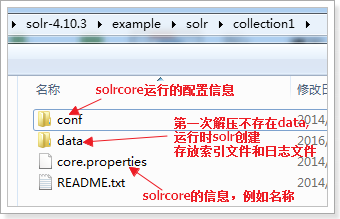
每个SolrCore提供单独的搜索和索引服务。

#### 目录结构

SolrHome目录：



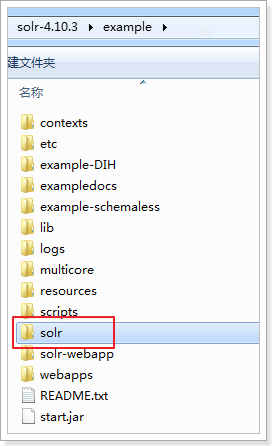
SolrCore目录：



### 创建SolrCore

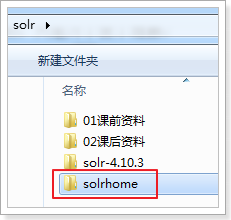
创建SolrCore先要创建SolrHome。在solr解压包下solr-4.10.3\example\solr文件夹就是一个标准的SolrHome，只需要将它复制到指定的目录下即可。

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

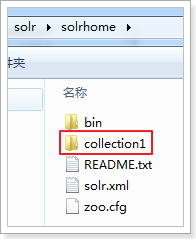


复制该文件夹到本地的一个目录，把文件名称改为solrhome。

改名不是必须的，只是为了便于理解

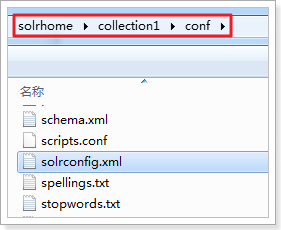


打开solrhome目录确认solrcore



### 配置SolrCore

其实就是配置SolrCore目录下的conf/solrconfig.xml。

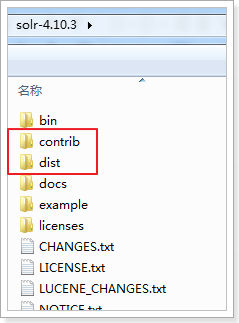


这个文件是来配置SolrCore实例的相关信息。如果使用默认配置可以不用做任何修改。它里面包含了不少标签，但是我们经常使用的标签为：lib标签、datadir标签、requestHandler标签。

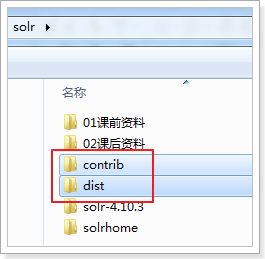
#### lib 标签

在solrconfig.xml中可以加扩展载一些的jar，如果需要使用，则首先要把这些jar复制到指定的目录，我们复制到SolrHome同级目录

复制之前解压的文件夹中的contrib和dist文件夹



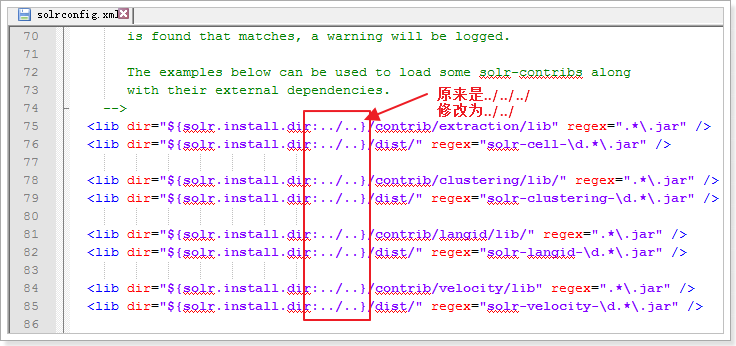
粘贴到SolrHome同级目录下。



修改solrconfig.xml配置文件加载扩展的jar。

configsolr.install.dir表示${SolrCore}的目录位置，需要如下修改：

./ 表示当前目录 ../表示上一级目录

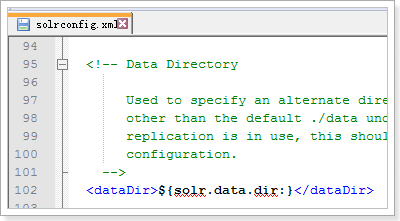


#### datadir标签

配置SolrCore的data目录。

data目录用来存放SolrCore的索引文件和tlog日志文件

solr.data.dir表示${SolrCore}/data的目录位置



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

例如：

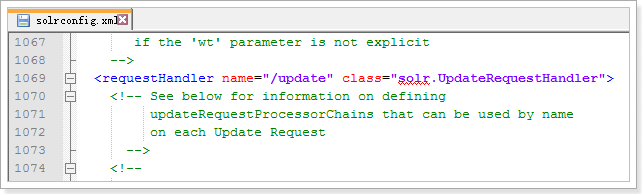


(建议不修改，否则配置多个SolrCore会报错）

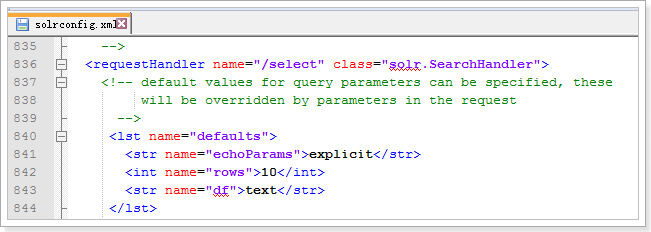
#### requestHandler标签

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。



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>

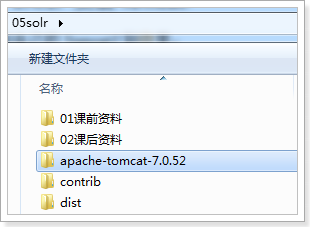
</requestHandler>

## Solr工程部署

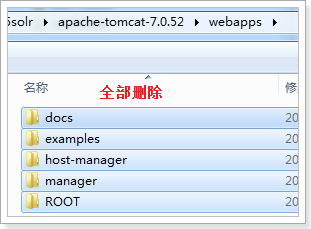
由于在项目中用到的web服务器大多数是用的Tomcat，所以就进行solr和Tomcat的整合。

### 安装Tomcat

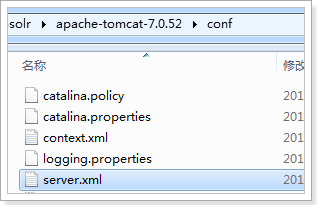
复制自己的Tomcat7到这里



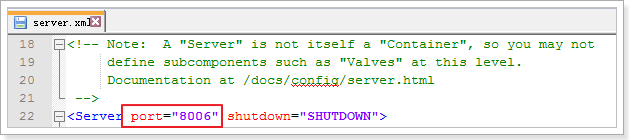
删除不用的应用（可以不删）

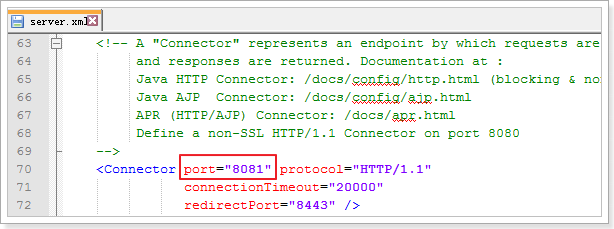


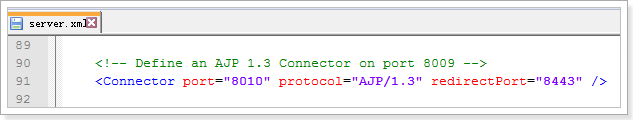
修改server.xml配置文件里面的端口号（否则后面eclipse使用Tomcat会冲突）



修改以下三个端口号



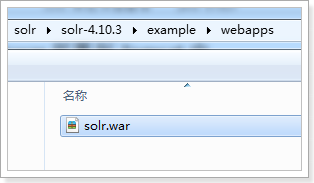




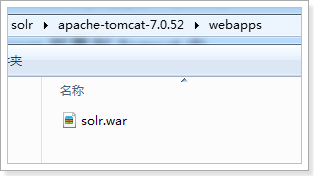
### 部署solr.war到Tomcat中

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

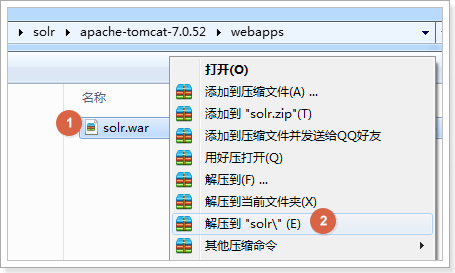
复制solr.war



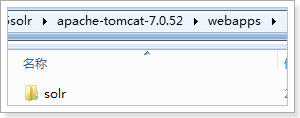
粘贴到自己Tomcat的webapps里

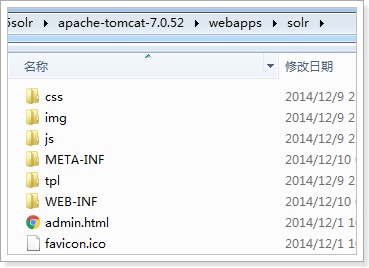


在Tomcat的webapps里，把war解压到当前路径，并删除solr.war



效果：

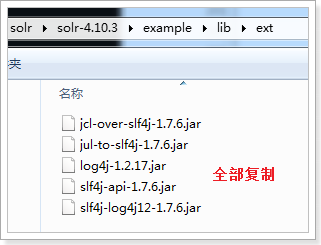




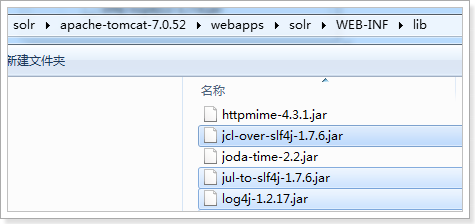
### 添加solr服务的扩展jar包（日志包）

把solr解压包下solr-4.10.3\example\lib\ext目录下的所有jar包拷贝到Tomcat部署的solr的WEB-INF/lib文件夹

复制扩展jar包

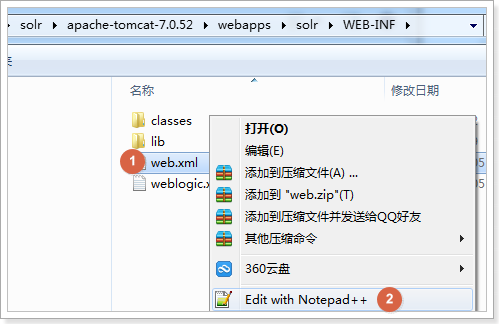


粘贴到Tomcat的webapps的solr工程的WEB-INF\lib目录



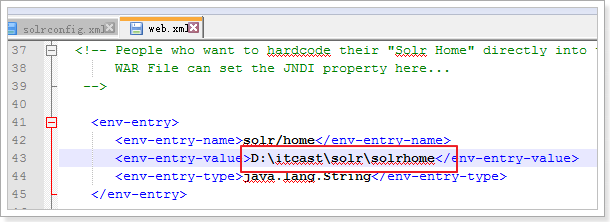
### 配置solr应用的web.xml

需要修改web.xml，让Tomcat使用JNDI的方式告诉solr服务器SolrHome在哪。



修改内容：

第42行的Solr/home名称必须是固定的，修改第43行，如下图

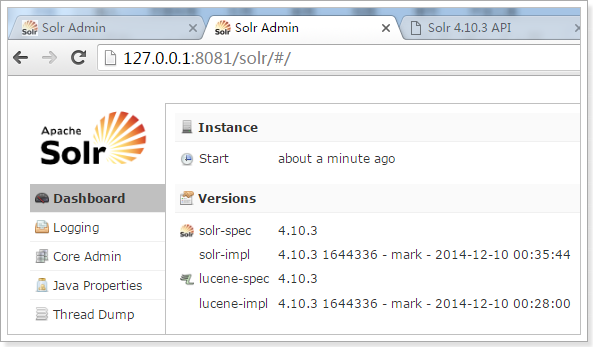


### 启动Tomcat进行访问

访问

http://localhost:8081/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，即电商项目会演示。

### Core Admin

Solr Core的管理界面。在这里可以添加SolrCore实例（有bug，不推荐使用浏览器界面添加SolrCore）。

### java properties

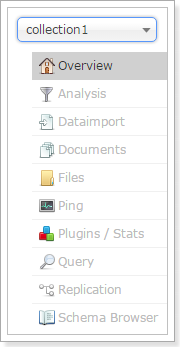
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

### 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

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



#### dataimport

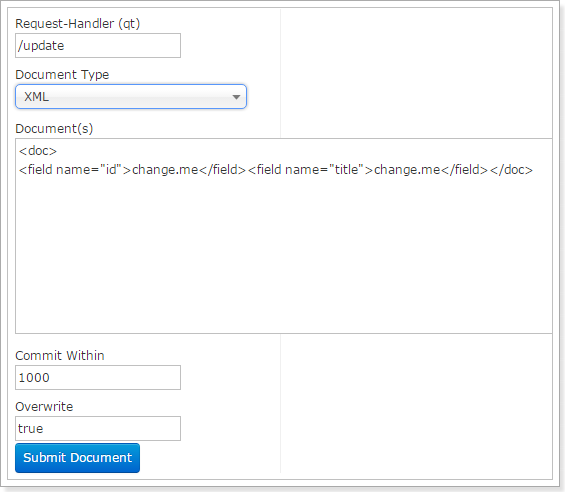
可以定义数据导入处理器，从关系数据库将数据导入到Solr索引库中。

默认没有配置，需要手工配置。

#### Document

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。

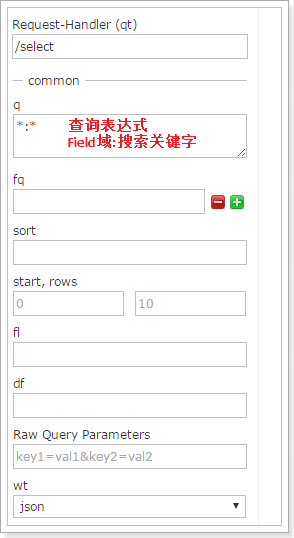
通过此菜单可以创建索引、更新索引、删除索引等操作，界面如下：



* 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，每隔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

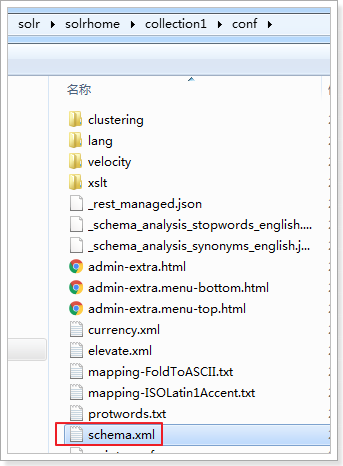
通过/select执行搜索索引，必须指定“q”查询条件方可搜索。



# solr基本使用

## schema.xml

schema.xml文件在SolrCore的conf目录下，在此配置文件中定义了域以及域的类型等一些配置。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）

### dynamicField（动态域）

<dynamicField name="\*\_s" type="string" 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。

### 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，可以提高查询效率。

定义目标域：

<field name="text" type="text\_general" indexed="true" stored="false" multiValued="true"/>

目标域必须要使用：multiValued="true"

### 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" />

<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"/>

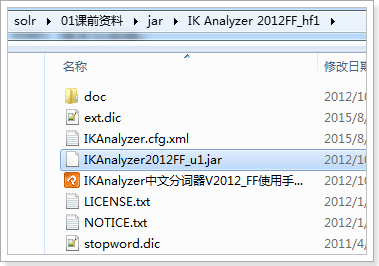
</analyzer>

</fieldType>

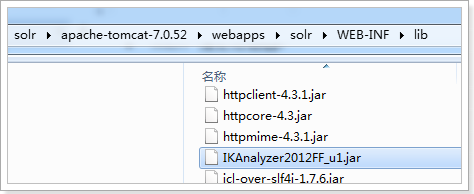
* name：域类型的名称
* class：指定域类型的solr类型。
* analyzer：指定分词器。在FieldType定义的时候最重要的就是定义这个类型的数据在建立索引和进行查询的时候要使用的分析器analyzer，包括分词和过滤。
* type：index和query。Index 是创建索引，query是查询索引。
* tokenizer：指定分词器
* filter：指定过滤器

## 配置中文分析器

使用IKAnalyzer中文分析器

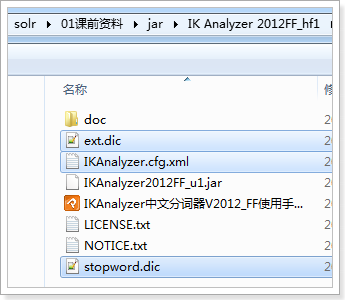


第一步：把IKAnalyzer2012FF\_u1.jar添加到solr/WEB-INF/lib目录下。

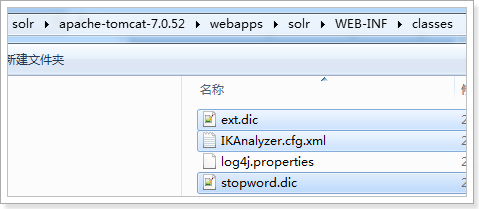


第二步：复制IKAnalyzer的配置文件和自定义词典和停用词词典到solr的solr/WEB-INF/classes目录下。

复制IK分词器配置文件、自定义词典、停用词词典



粘贴到Tomcat的solr的/WEB-INF/classes目录下



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>

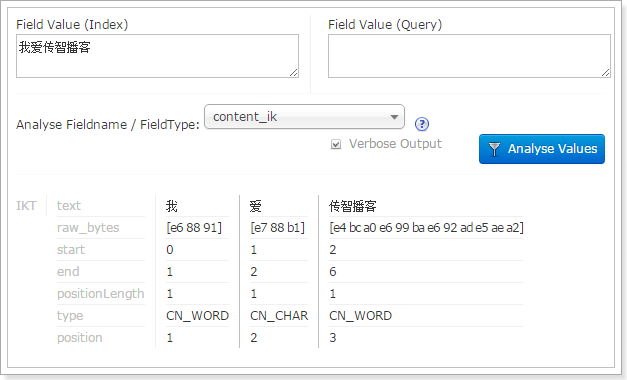
第四步：在schema.xml中添加field，指定field的type属性为text\_ik

<!--IKAnalyzer Field-->

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

第五步：重启tomcat

效果：



## 配置业务Field

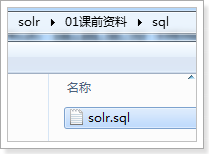
### 需求

要使用solr实现网站中商品搜索，需要 将mysql数据库中数据在solr中创建索引。

1. 需要在solr的schema.xml文件定义要存储的商品Field。
2. 需要把MySQL的数据导入到solr索引库中
3. 开发搜索功能

### 数据库添加数据

在数据库中运行solr.sql脚本

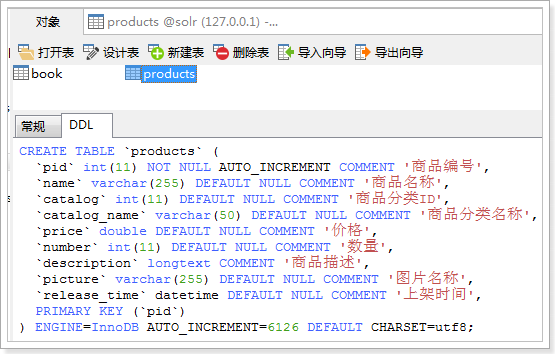


### 定义Field

先确定定义的商品document的Field域有哪些？

可以根据mysql数据库中商品表的字段来确定：

products商品表：



Schema.xml中配置业务域

<!--product-->

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

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

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

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

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

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

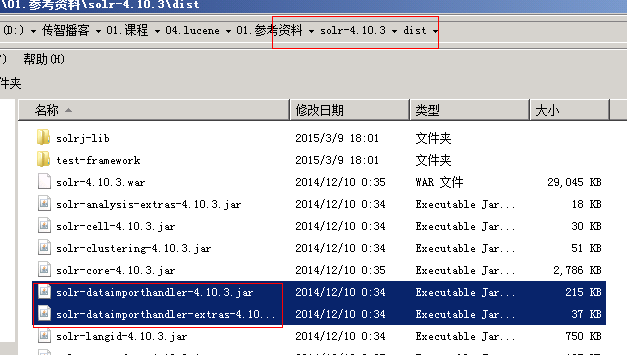
<copyField source="product\_name" dest="product\_keywords"/>

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

## dataimportHandler插件

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solrcore（collection1\lib）中, 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/lucene"  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> |

第四步：重启tomcat

# 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，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

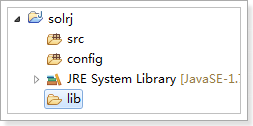
使用solrj调用solr服务实现对索引库的增删改查操作。

## 环境准备

* Solr：4.10.3
* Jdk环境：1.7
* IDE环境：Eclipse Mars2

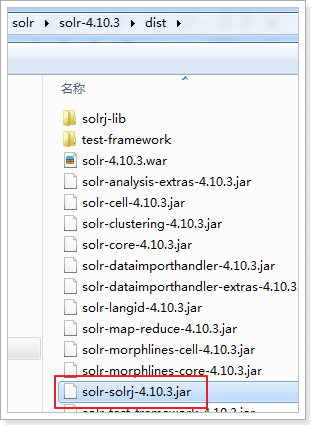
## 工程搭建

### 创建java工程



### 添加jar

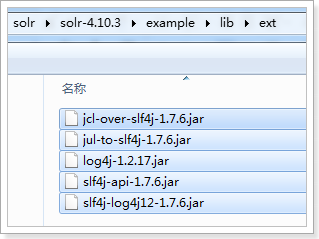
Solrj的包，\solr-4.10.3\dist\目录下



solrj依赖包，\solr-4.10.3\dist\solrj-lib



Solr服务的依赖包，\solr\example\lib\ext



## 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** testCreateAndUpdateIndex() **throws** Exception {

// 1. 创建HttpSolrServer对象

// 设置solr服务接口,浏览器客户端地址http://127.0.0.1:8081/solr/#/

String baseURL = "http://127.0.0.1:8081/solr";

HttpSolrServer httpSolrServer = **new** HttpSolrServer(baseURL);

// 2. 创建SolrInputDocument对象

SolrInputDocument document = **new** SolrInputDocument();

document.addField("id", "c1001");

document.addField("content ", "Hello world!");

// 3. 把SolrInputDocument对象添加到索引库中

httpSolrServer.add(document);

// 4. 提交

httpSolrServer.commit();

}

### 删除索引

#### 代码

抽取HttpSolrServer 的创建代码

**private** HttpSolrServer httpSolrServer;

// 提取HttpSolrServer创建

@Before

**public** **void** init() {

// 1. 创建HttpSolrServer对象

// 设置solr服务接口,浏览器客户端地址http://127.0.0.1:8081/solr/#/

String baseURL = "http://127.0.0.1:8081/solr/";

**this**.httpSolrServer = **new** HttpSolrServer(baseURL);

}

删除索引逻辑，两种：

根据id删除

根据条件删除，根据条件删除

可以使用\*:\*作为条件，就是删除所有数据（慎用）

@Test

**public** **void** testDeleteIndex() **throws** Exception {

// 根据id删除索引数据

// this.httpSolrServer.deleteById("c1001");

// 根据条件删除（如果是\*:\*就表示全部删除，慎用）

**this**.httpSolrServer.deleteByQuery("\*:\*");

// 提交

**this**.httpSolrServer.commit();

}

### 查询索引

#### 简单查询

/\*\*

\* 简单搜索

\*

\* @throws Exception

\*/

@Test

public void testSearchIndex1() throws Exception {

// 创建搜索对象

SolrQuery query = new SolrQuery();

// 设置搜索条件

query.setQuery("\*:\*");

// 发起搜索请求

QueryResponse response = this.httpSolrServer.query(query);

// 处理搜索结果

SolrDocumentList results = response.getResults();

System.*out*.println("搜索到的结果总数：" + results.getNumFound());

// 遍历搜索结果

for (SolrDocument solrDocument : results) {

System.*out*.println("----------------------------------------------------");

System.*out*.println("id：" + solrDocument.get("id"));

System.*out*.println("content" + solrDocument.get("content"));

}

}