Solr

全文检索技术

1. Solr介绍 15
2. Solr安装配置-SolrCore配置 15
3. Solr安装配置-Solr工程部署 10
4. Solr安装配置-Solr工程测试 15
5. Solr安装配置-多SorlCore配置 10
6. Solr入门--schema 20
7. Solr入门-安装中文分词器 15
8. Solr入门-自定义业务系统Field 20
9. Solr入门-SolrJ介绍10
10. Solr入门-SolrJ索引维护 20
11. Solr入门-数据导入处理器20
12. Solr入门-搜索语法 10
13. Solr入门-SolrJ-搜索 20
14. Solr案例-需求分析 10
15. Solr案例-架构设计 15
16. Solr案例-Service 20
17. Solr案例-Controller 20
18. Solr案例-主体调试 20
19. Solr案例-功能完善

【Solr介绍】

Solr是什么

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。



Solr与Lucene的区别

Lucene是一个开放源代码的全文检索引擎工具包，它不是一个完整的全文检索引擎，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。

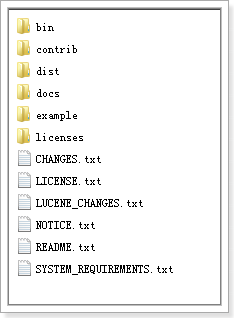
Solr的目标是打造一款企业级的搜索引擎系统，**它是一个搜索引擎服务，可以独立运行，通过Solr可以非常快速的构建企业的搜索引擎，通过Solr也可以高效的完成站内搜索功能。**

## Solr下载

从Solr官方网站（http://lucene.apache.org/solr/ ）下载Solr4.10.3，根据Solr的运行环境，Linux下需要下载lucene-4.10.3.tgz，windows下需要下载lucene-4.10.3.zip。

Solr使用指南可参考：https://wiki.apache.org/solr/FrontPage。

下载solr-4.10.3.zip并解压：



bin：solr的运行脚本

contrib：solr的一些贡献软件/插件，用于增强solr的功能。

dist：该目录包含build过程中产生的war和jar文件，以及相关的依赖文件。

docs：solr的API文档

example：solr工程的例子目录：

* example/solr：

该目录是一个包含了默认配置信息的Solr的Core目录。

* 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，环境如下：

Solr：Solr4.10.3

Jdk：jdk1.7.0\_72

Tomcat：apache-tomcat-7.0.53

# Solr安装配置

## Solr与Tomcat整合

### 【SolrCore配置】

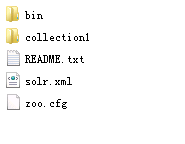
SolrHome与SolrCore

创建一个Solr home目录，SolrHome是Solr运行的主目录，目录中包括了运行Solr实例所有的配置文件和数据文件。

**Solr实例即SolrCore，一个SolrHome中可以包括多个SolrCore目录，每个SolrCore对外提供单独的搜索和索引服务。**

目录结构

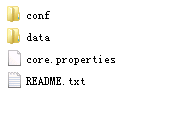
example\solr是一个solr home目录结构，如下：



collection1：叫做一个Solr运行实例SolrCore也称为一个collection，SolrCore的名称不固定，命名方式类似collection1/collection2/collection3等，每个collection对外单独提供索引和搜索接口。

solrHome中可以创建多个solrCore即collection。

SolrCore目录内容如下所示：



说明：

（1）一个solrCore中包括一个索引目录，该目录存储了该 SolrCore的索引文件。data是SolrCore的索引文件目录 ，如果没有solr会自动创建。

（2）conf是SolrCore的配置文件目录 ：里面包括两个重要的配置文件solrconfig.xml与schema.xml配置文件

创建SolrCore

创建目录 F:\develop\solr

将example\solr目录 拷贝至 F:\develop\solr目录下并改名为solrHome。

将contrib和dist两个目录拷贝到F:\develop\solr下。



SolrHome下：



#### SolrCore配置

每个SolrCore的conf目录下有solrconfig.xml文件，它是SolrCore运行的配置文件。

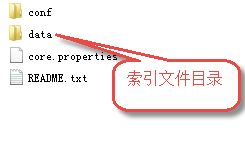
##### 加载jar包

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，需要根据扩展Jar的实际存放位置坐对应的修改：



##### dataDir索引目录

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：



##### 请求处理器

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-->

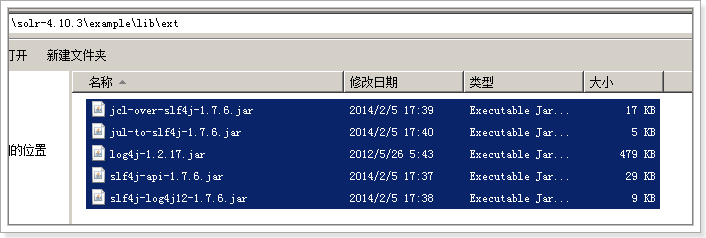
<str name="df">text</str><!--默认搜索字段-->

</lst>

</requestHandler>

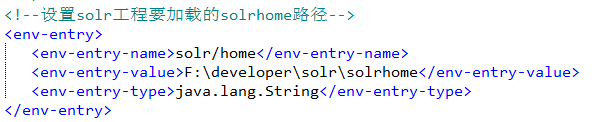
【Solr工程部署】

1. 将dist\solr-4.10.3.war拷贝到Tomcat的webapp目录下改名为solr.war
2. 启动tomcat后，solr.war自动解压，将原来的solr.war删除。
3. 拷贝example\lib\ext 目录下所有jar包拷贝到Tomcat的webapp\solr\WEB-INF\lib目录下



1. 修改Tomcat目录 下webapp\solr\WEB-INF\web.xml文件，如下所示：

设置Solrhome



1. 拷贝log4j.properties文件

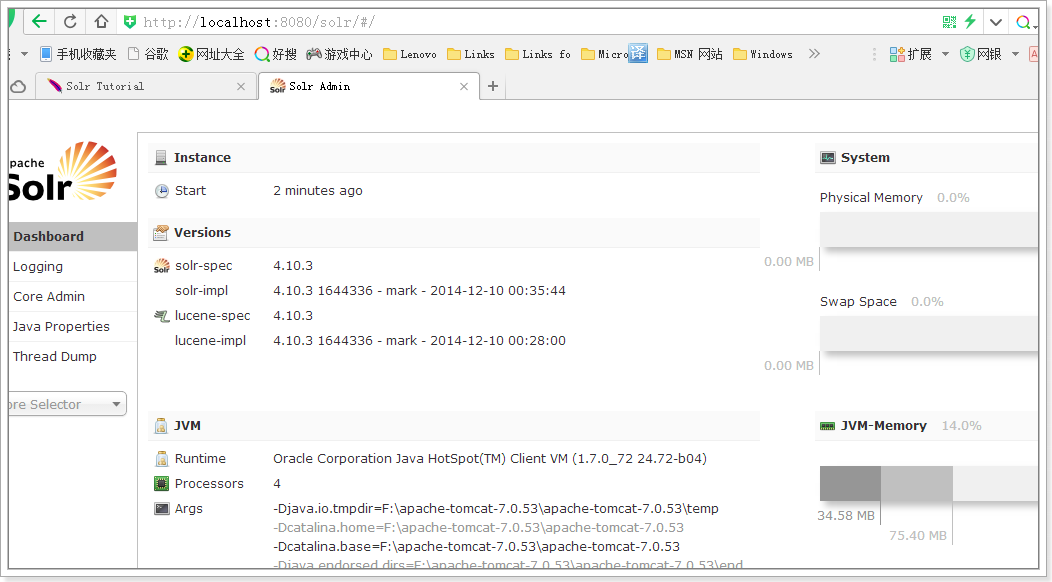
在 Tomcat下webapps\solr\WEB-INF目录中创建文件 classes文件夹，

复制Solr目录下example\resources\log4j.properties至Tomcat下webapps\solr\WEB-INF\classes目录

### 启动Tomcat

访问http://localhost:8080/solr

## 【管理界面】



### Dashboard：

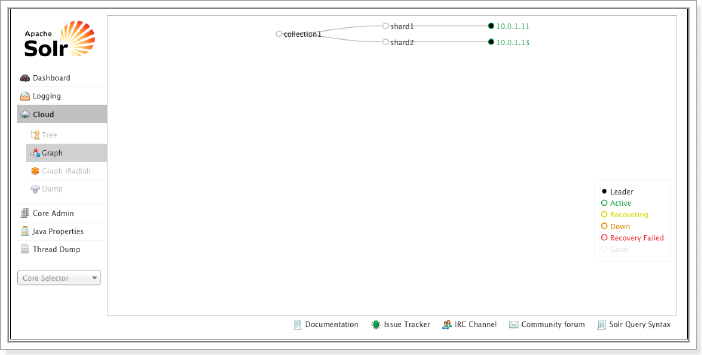
仪表盘，显示了该Solr实例开始启动运行的时间、版本、系统资源、jvm等信息。

### 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，如下图是Solr Cloud的管理界面：



### Core Admin：

Solr Core的管理界面。Solr Core 是Solr的一个独立运行实例单位，它可以对外提供索引和搜索服务，一个Solr工程可以运行多个SolrCore（Solr实例），一个Core对应一个索引目录。

### java properties

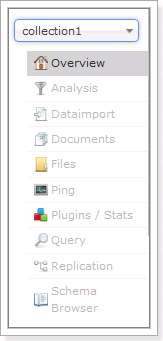
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

### 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（重点）



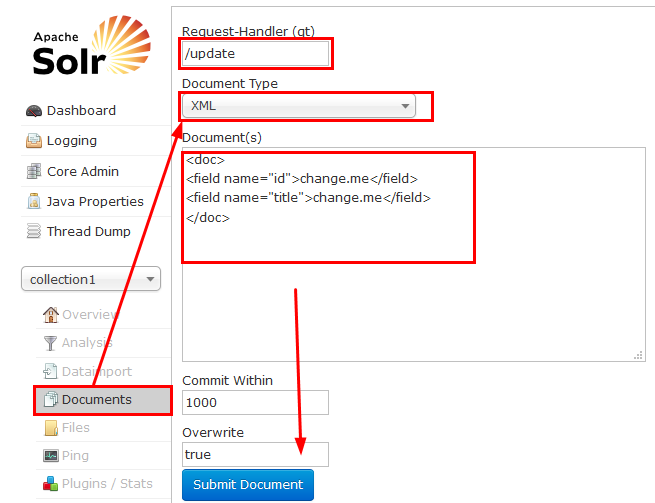
通过此界面可以测试索引分析器和搜索分析器的执行情况。

### 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。

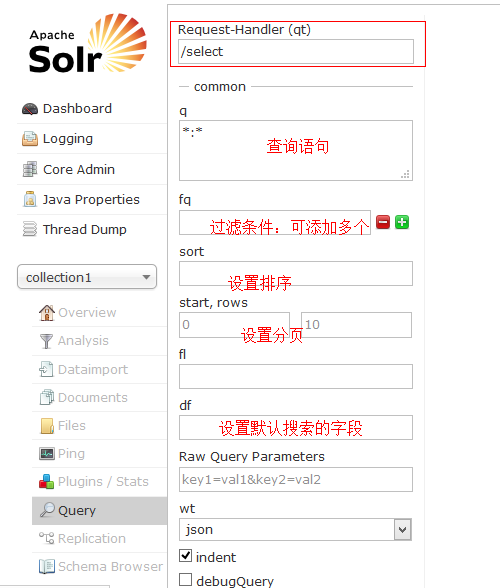
Document（重点）

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：



/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。

### query（重点）



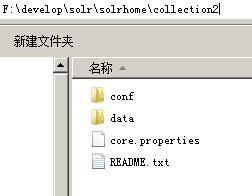
**通过/select执行搜索索引，必须指定“q”查询条件方可搜索。**

## 【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：

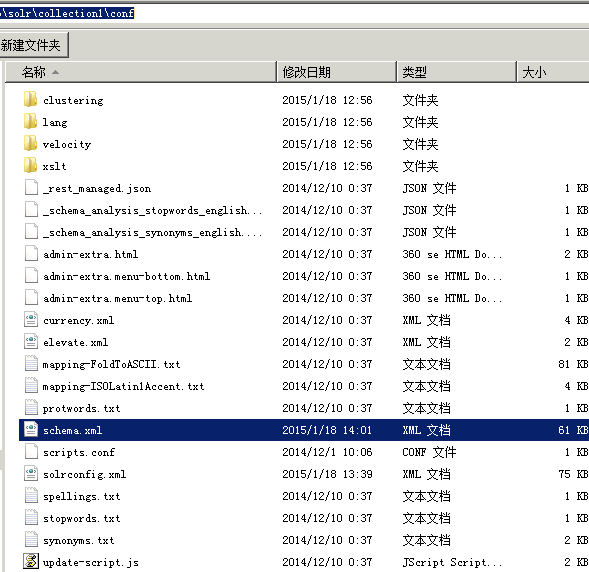


演示多core的使用，在collection1和collection2中分别创建索引、搜索索引。

# Solr基本使用

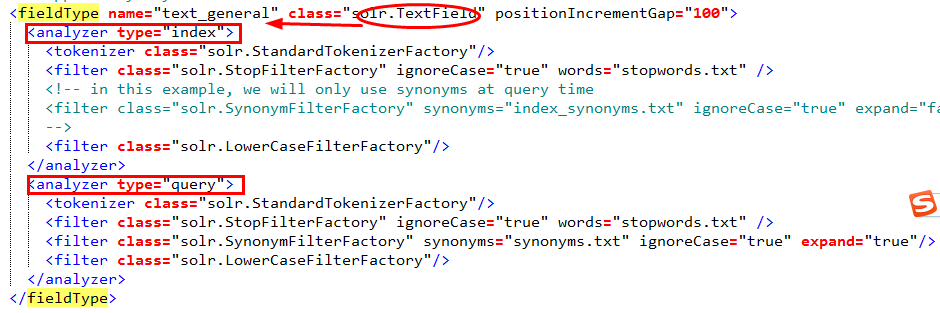
## 【scheam.xml】

schema.xml，在SolrCore的conf目录下，它是Solr数据表配置文件，它定义了加入索引的数据的数据类型的。主要包括FieldTypes、Fields和其他的一些缺省设置。



FieldType域类型定义

下边“text\_general”是Solr默认提供的FieldType，通过它说明FieldType定义的内容：



FieldType子结点包括：name,class,positionIncrementGap等一些参数：

name：是这个FieldType的名称

class：是Solr提供的solr.TextField，solr.TextField 允许用户通过分析器来定制索引和查询，分析器包括一个分词器（tokenizer）和多个过滤器（filter）

positionIncrementGap：可选属性，定义在同一个文档中此类型数据的空白间隔，避免短语匹配错误，此值相当于Lucene的短语查询设置slop值，根据经验设置为100。

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

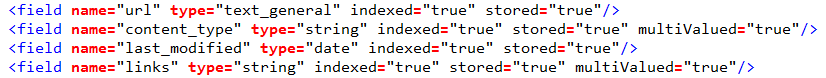
索引分析器中：使用solr.StandardTokenizerFactory标准分词器，solr.StopFilterFactory停用词过滤器，solr.LowerCaseFilterFactory小写过滤器。

搜索分析器中：使用solr.StandardTokenizerFactory标准分词器，solr.StopFilterFactory停用词过滤器，这里还用到了solr.SynonymFilterFactory同义词过滤器。

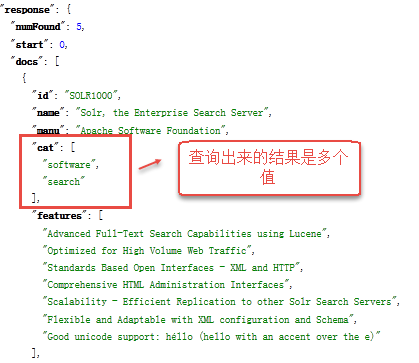
Field定义

在fields结点内定义具体的Field，filed定义包括name,type（为之前定义过的各种FieldType）,indexed（是否被索引）,stored（是否被储存），multiValued（是否存储多个值）等属性。

如下：



multiValued：该Field如果要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），通过使用solr查询要看出返回给客户端是数组：



uniqueKey

Solr中默认定义唯一主键key为id域，如下：



Solr在删除、更新索引时使用id域进行判断，也可以自定义唯一主键。

**注意在创建索引时必须指定唯一约束。**

copyField复制域

copyField复制域，可以将多个Field复制到一个Field中，以便进行统一的检索：

比如，输入关键字要搜索title标题和作者author以及描述description三个域时，要用到复制域。定义title、authro、descripton、text的域：



Text域允许多个值。



根据关键字只搜索text域的内容就相当于搜索title、author、description，将title、author、description复制到text中，如下：



dynamicField（动态字段）

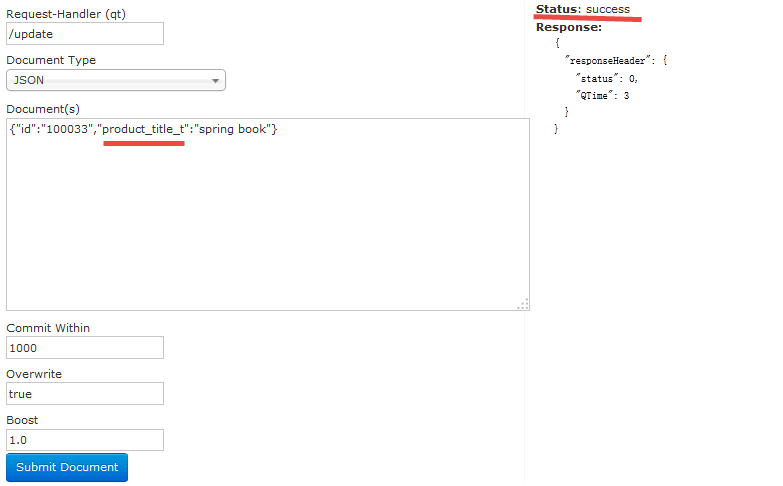
动态字段就是不用指定具体的名称，只要定义字段名称的规则，例如定义一个 dynamicField，name 为\*\_i，定义它的type为text，那么在使用这个字段的时候，任何以\_i结尾的字段都被认为是符合这个定义的，例如：name\_i，gender\_i，school\_i等。

自定义Field名为：product\_title\_t，“product\_title\_t”和scheam.xml中的dynamicField规则匹配成功，如下：

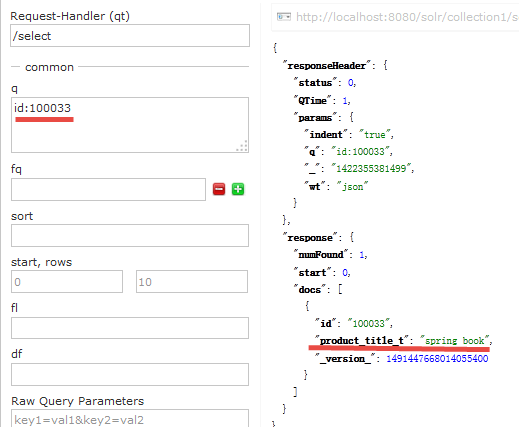


“product\_title\_t”是以“\_t”结尾。

创建索引：



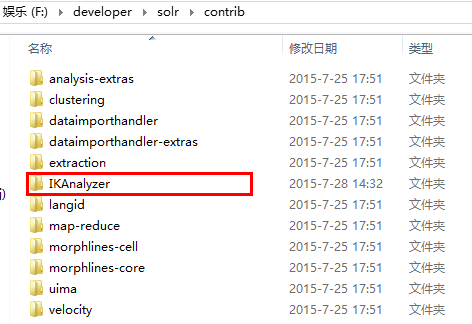
搜索索引：



## 【安装中文分词器】

IKAnalyzer部署

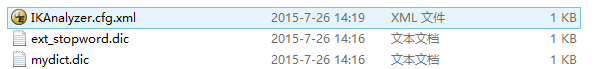
（1）将IKAnalyzer2012FF\_u1.jar拷贝到 solr\contrib\IKAnalyzer\lib下，在solrconfig.xml中添加：



（2）在Tomcat的webapps/solr/WEB-INF/下创建classes目录

将IKAnalyzer.cfg.xml、ext\_stopword.dic mydict.dic copy到

Tomcat/webapps/solr/WEB-INF/classes下



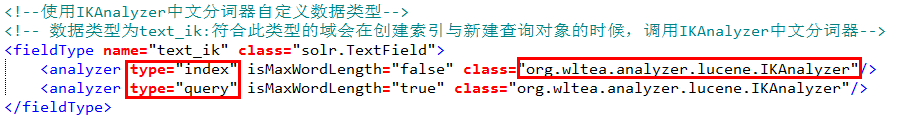
**注意：ext\_stopword.dic 和mydict.dic必须保存成无BOM的utf-8类型。**

修改schema.xml文件

1. FieldType

首先需要在types结点内定义一个FieldType子结点，包括name,class,等参数，name就是这个FieldType的名称，class指向org.apache.solr.analysis包里面对应的class名称，用来定义这个类型的行为。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

修改Solr的schema.xml文件，添加FieldType：



在index或query分别配置isMaxWordLength，查询采用IK自己的最大分词法设置为true，索引则采用它的细粒度分词法设置为false。

1. Field：

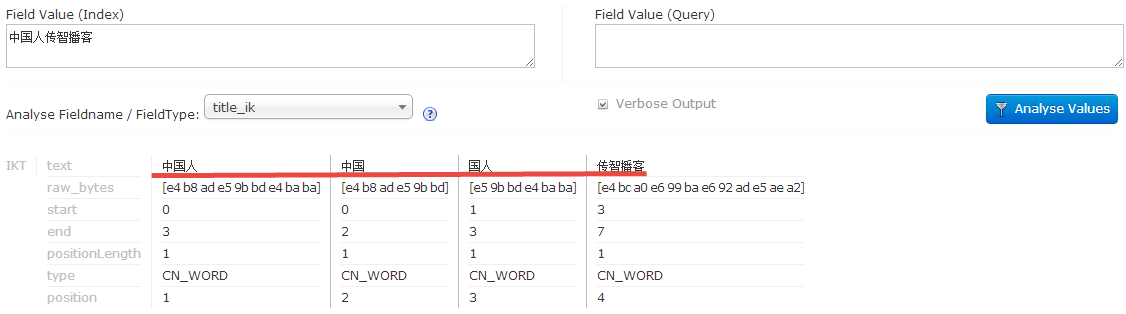
FieldType定义好后就可以在fields结点内定义具体的field，filed定义包括name,type（即FieldType）,indexed（是否被索引）,stored（是否被储存），multiValued（是否有多个值）等

<!--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"/>

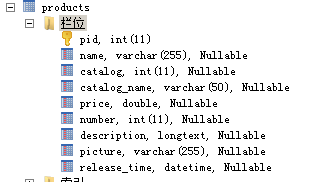
测试



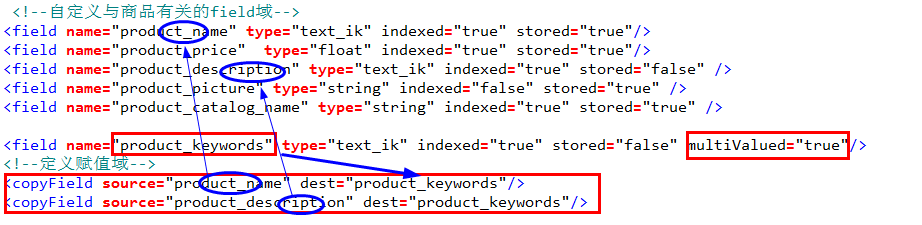
## 【自定义业务系统Field】

如果不使用Solr提供的Field可以针对具体的业务需要自定义一套Field，如下是商品信息Field：

对Mysql数据库中的product表数据定义field：



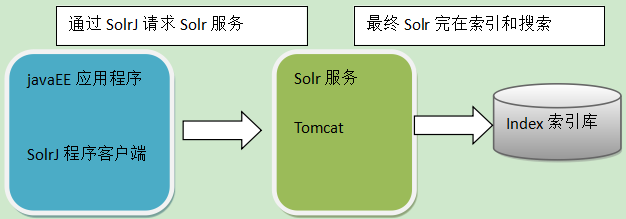
在schema.xml中定义如下：



## 【SolrJ索引维护】

### SolrJ介绍

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通常嵌入在业务系统中，通过SolrJ的API接口操作Solr服务，如下图：



### SolrJ安装

Solrj就是一堆jar包，将solrj的jar包加入工程即可使用。

jar包明细：

solr-4.10.3\dist\solrj-lib\\*

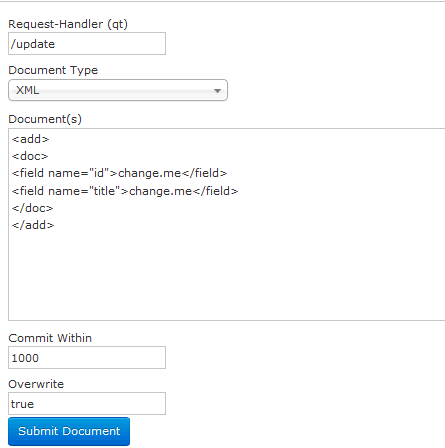
solr-4.10.3\dist\ solr-solrj-4.10.3.jar

solr-4.10.3\example\lib\ext\\*

junit包

【添加/更新索引】

使用/update进行索引维护，进入Solr管理界面SolrCore下的Document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，在</add>后边添加“<commit/>”，如下：

<add>

<doc>

<field name="id">change.me</field>

<field name="title">change.me</field>

</doc>

</add>

<commit/>

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。

请求xml格式如下：

<add>

<doc>

<field name="id">change.me</field>

<field name="？？" >？？</field>

。。。

</doc>

</add>

说明：唯一标识 Field必须有，这里使用Solr默认的id。

使用SolrJ创建索引，通过调用SolrJ提供的API请求Solr服务，Document通过SolrInputDocument进行构建。

// 创建索引

**public** **void** **createIndex()** **throws** SolrServerException**,** IOException**{**

**//当没有指定solrCore时，会默认使用先创建的solrCore**

**String** baseURL**=**"http://127.0.0.1:8080/solr"**;**

**SolrServer** solrServer**=new** *HttpSolrServer***(**baseURL**);**

**//创建文档**

**SolrInputDocument** doc**=new** *SolrInputDocument***();**

**//向文档中添加域**

doc**.***addField***(**"id"**,** "7"**);**

doc**.***addField***(**"product\_name"**,** "魔幻星座音乐水晶球内雕音乐盒七彩渐变音乐球"**);**

doc**.***addField***(**"product\_price"**,** **48.0f);**

**UpdateResponse** updateResponse **=** solrServer**.***add***(**doc**);**

solrServer**.***commit***();**

**System.*out*.***println***(**"通过solrJ添加索引成功"**);**

**}**

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。

删除索引

删除索引格式如下：

1） 删除制定ID的索引

<delete>

<id>8</id>

。。。

</delete>

<commit/>

2） 删除查询到的索引数据

<delete>

<query>product\_catalog\_name:幽默杂货</query>

</delete>

<commit/>

3） 删除所有索引数据

<delete>

<query>\*:\*</query>

</delete>

<commit/>

SorJ代码如下：

// 删除索引

@Test

**public** **void** testDeleteIndex() **throws** SolrServerException, IOException {

SolrServer solrServer = **new** HttpSolrServer(*urlString*);

//根据id删除

UpdateResponse response = solrServer.deleteById("c0001");

//根据多个id删除

// solrServer.deleteById(ids);

//自动查询条件删除

// solrServer.deleteByQuery("product\_keywords:教程");

// 提交

solrServer.commit();

}

说明：deleteById(String id)根据id删除索引，此方法为重载方法，也可以传个多个id批量删除，也可以调用deleteByQuery() 根据查询条件删除

## 【数据导入处理器】

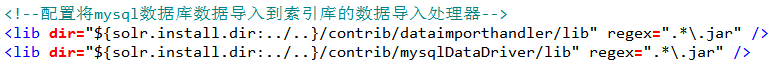
安装dataimport-Handler从关系数据库将数据导入到索引库。

**第一步：加载dataimport的jar包**

将solr\dist\solr-dataimporthandler-4.10.3.jar拷贝至solr\contrib\dataimporthandler下，

将mysql数据库驱动的jar拷贝至solr\contrib\databaseDriver下

在solrconfig.xml加载以上jar包：



**第二步：编辑data-config.xml文件**，存放在SolrCore的conf目录

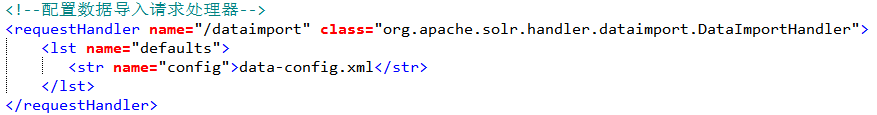


说明：

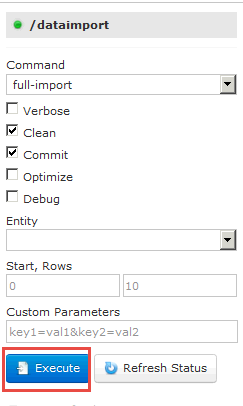
<field column="pid" name="id"/>必须有一个id域，这里使用Solr默认的id域，域值是从关系数据库查询的pid列值。

下边以“product\_”开头的Field都是在schema.xml中自定义的商品信息Field。

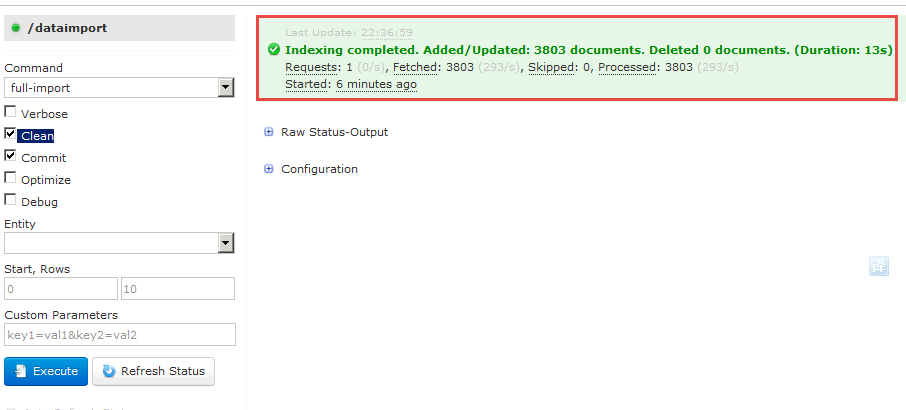
**第三步：配置请求处理器。**修改solrconfig.xml，添加requestHandler：



**第四步：重启Tomcat****。**进入管理界面-->SolrCore-->dataimport下执行导入



**第五步：查看导入结果。**进入管理界面-->SolrCore-->dataimport下

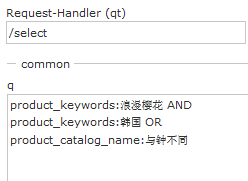


## SolrJ搜索

查询语法

通过/select搜索索引，Solr制定一些参数完成不同需求的搜索：

1. q - 查询字符串，必须的，如果查询所有使用\*:\*。

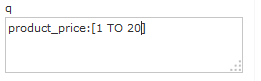


1. fq - （filter query）过虑查询，作用：在q查询符合结果中同时是fq查询符合的，例如：：



过滤查询价格从1到20的记录。

也可以在“q”查询条件中使用product\_price:[1 TO 20]，如下：



也可以使用“\*”表示无限，例如：

20以上：product\_price:[20 TO \*]

20以下：product\_price:[\* TO 20]

1. sort - 排序，格式：sort=<field name>+<desc|asc>[,<field name>+<desc|asc>]… 。示例：

按价格降序



1. start - 分页显示使用，开始记录下标，从0开始
2. rows - 指定返回结果最多有多少条记录，配合start来实现分页。



显示前10条。

1. fl - 指定返回那些字段内容，用逗号或空格分隔多个。

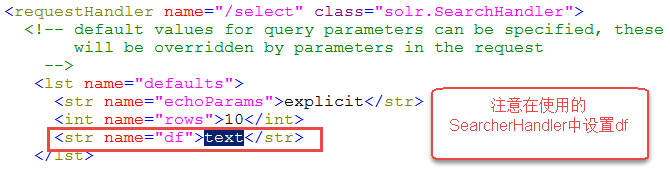


显示商品图片、商品名称、商品价格

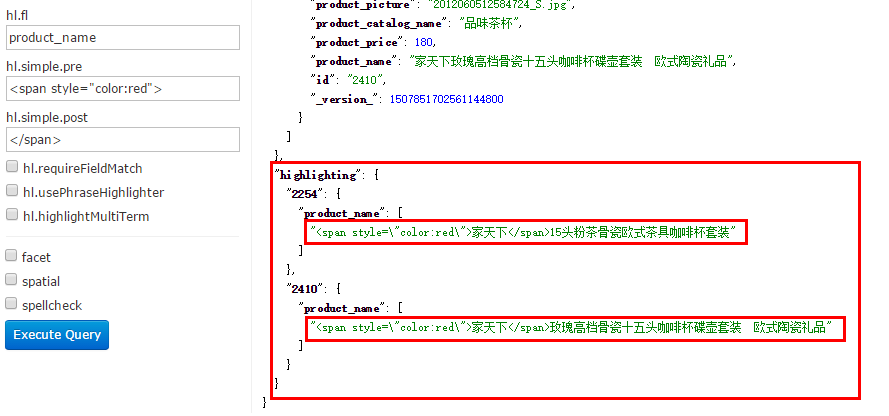
1. df-指定一个搜索Field



也可以在SolrCore目录 中conf/solrconfig.xml文件中指定默认搜索Field，指定后就可以直接在“q”查询条件中输入关键字。



1. wt - (writer type)指定输出格式，可以有 xml, json, php, phps, 后面 solr 1.3增加的，要用通知我们，因为默认没有打开。
2. hl 是否高亮 ,设置高亮Field，设置格式前缀和后缀。



SolrJ代码

1. 最基本的查询：

@Test

**public** **void** **indexSearch()** **throws** SolrServerException**,** IOException**{**

**//创建solr服务器**

**SolrServer** solrServer**=new** *HttpSolrServer***(**baseURL**);**

**//创建solr查询对象**

**SolrQuery** solrQuery**=new** *SolrQuery***();**

**//设置查询条件**

***solrQuery.set("q", "product\_keywords:家天下");***

**//使用solr服务器执行查询**

**QueryResponse** queryResponse **=** solrServer**.***query***(**solrQuery**);**

**//获取查询结果**

**SolrDocumentList** solrDocumentList **=** queryResponse**.***getResults***();**

**for** **(SolrDocument** solrDocument **:** solrDocumentList**)** **{**

**System.*out*.***println***(**"商品ID："**+**solrDocument**.***get***(**"id"**)+**"\t"**+**

"商品名称："**+**solrDocument**.***get***(**"product\_name"**)**

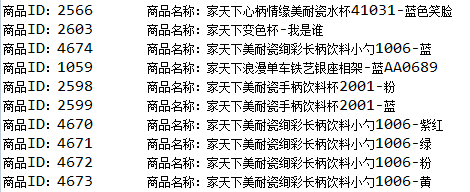
**);**

**}**

solrServer**.***commit***();**

**}**

部分查询结果如下：



1. 添加筛选条件：

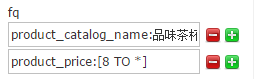
**//设置查询条件**

solrQuery**.***set***(**"q"**,** "product\_keywords:家天下"**);**

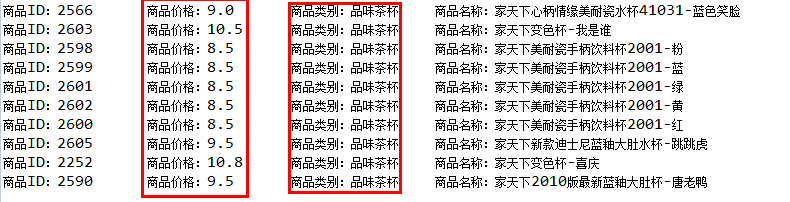
**//设置从查询结果中筛选的筛选条件**

solrQuery**.***addFilterQuery***(**"product\_catalog\_name:品味茶杯"**);**

solrQuery**.***addFilterQuery***(**"product\_price:[8 TO \*]"**);**



查询结果：

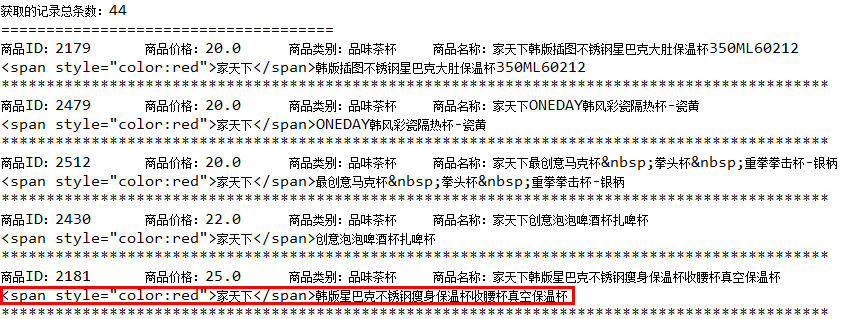


1. 排序

**//设置排序**

solrQuery**.***setSort***(**"product\_price"**,** ORDER**.*asc*);**

1. 高亮



1. 分页

**//设置分页**

**Integer** rows**=5;**

**Integer** curpage**=1;**

**Integer** start**=(**curpage**-1)\***rows**;**

solrQuery**.***setRows***(**rows**);**

solrQuery**.***setStart***(**start**);**

完整的solrJ代码：

@Test

**public** **void** **advancedIndexSearch()** **throws** SolrServerException**,** IOException**{**

**//创建solr服务器**

**SolrServer** solrServer**=new** *HttpSolrServer***(**baseURL**);**

**//创建solr查询对象**

**SolrQuery** solrQuery**=new** *SolrQuery***();**

**//设置查询条件**

solrQuery**.***set***(**"q"**,** "product\_keywords:家天下"**);**

**//设置从查询结果中筛选的筛选条件**

solrQuery**.***addFilterQuery***(**"product\_catalog\_name:品味茶杯"**);**

solrQuery**.***addFilterQuery***(**"product\_price:[20 TO \*]"**);**

**//设置排序**

solrQuery**.***setSort***(**"product\_price"**,** ORDER**.*asc*);**

**//设置分页**

**Integer** rows**=5;**

**Integer** curpage**=1;**

**Integer** start**=(**curpage**-1)\***rows**;**

solrQuery**.***setRows***(**rows**);**

solrQuery**.***setStart***(**start**);**

**//开启高亮**

solrQuery**.***setHighlight***(true);**

**//添加要高亮显示的域（可以添加多个）**

solrQuery**.***addHighlightField***(**"product\_name"**);**

**//指定高亮的前缀与后缀**

solrQuery**.***setHighlightSimplePre***(**"<span style=\"color:red\">"**);**

solrQuery**.***setHighlightSimplePost***(**"</span>"**);**

**//使用solr服务器执行查询**

**QueryResponse** queryResponse **=** solrServer**.***query***(**solrQuery**);**

**//从查询结果中获取高亮信息：因为查询信息与高亮信息时分开的，需要单独去取**

Map**<String,** Map**<String,** List**<String>>>** highlighting **=** queryResponse**.***getHighlighting***();**

**//获取查询结果**

**SolrDocumentList** solrDocumentList **=** queryResponse**.***getResults***();**

**System.*out*.***println***(**"获取的记录总条数："**+**solrDocumentList**.***getNumFound***());**

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

**for** **(SolrDocument** solrDocument **:** solrDocumentList**)** **{**

**System.*out*.***println***(**"商品ID："**+**solrDocument**.***get***(**"id"**)+**"\t"**+**

"商品价格："**+**solrDocument**.***get***(**"product\_price"**)+**"\t"**+**

"商品类别："**+**solrDocument**.***get***(**"product\_catalog\_name"**)+**"\t"**+**

"商品名称："**+**solrDocument**.***get***(**"product\_name"**)**

**);**

**//根据商品主键取出当前商品的高亮信息**

Map**<String,** List**<String>>** hlInfo **=** highlighting**.***get***(**solrDocument**.***get***(**"id"**).***toString***());**

**String** highLighInfo**=**hlInfo**.***get***(**"product\_name"**).***get***(0);//一个商品可能多个域上都加了高亮信息，所以此处返回的是List**

**System.*out*.***println***(**highLighInfo**);**

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

**}**

solrServer**.***commit***();**

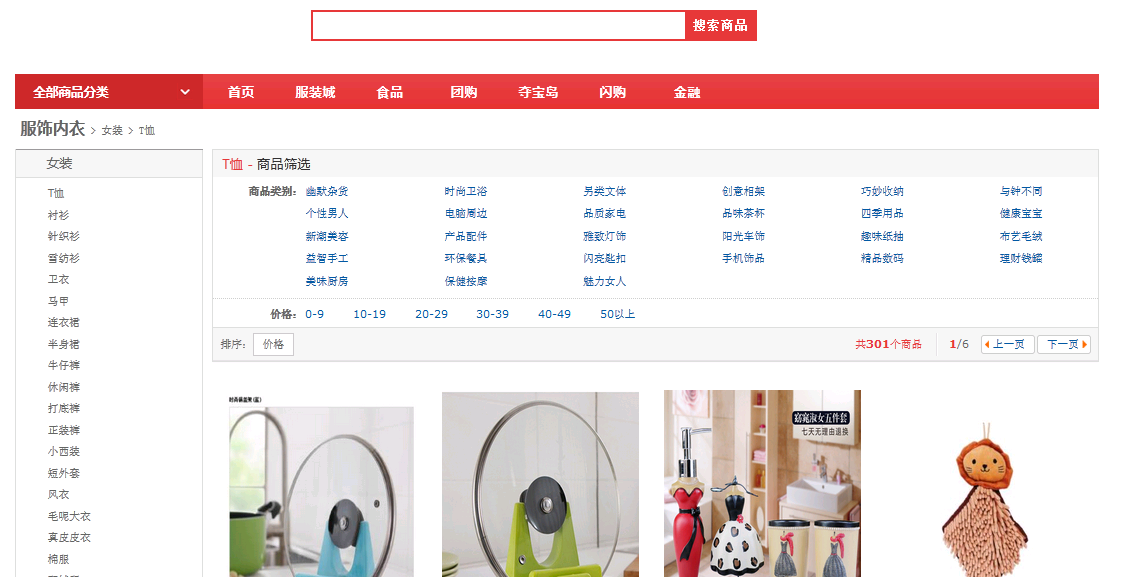
**}**

案例

【需求分析】

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：



1、通过关键字搜索商品名称和商品内容。

2、用户点击分类搜索商品。

3、根据价格范围搜索商品。

4、根据价格进行排序（升序、降序）

点击一下升序，再点一下是降序。

1. 默认搜索相关度排序。
2. 可以分页查找 。

7、用户查询出来要展示商品图片、商品名称、商品价格。

【系统架构】

spring容器

表现层

springmvc

Service层

SolrJ

Dao层

Solr索引库

mysql数据库

商品信息

Tomcat

Solr服务

索引、搜索请求

架构包括三块：

业务系统：针对具体需求开发的业务系统，本例子是电商系统

mysql数据库：存储了电商系统的表用于持久化，其中包括商品信息。

Solr服务：部署的solr服务，对外提供全文检索服务。

索引流程如下：

1、管理员在业务系统中维护商品信息

2、将商品信息写入关系数据库

3、将商品信息写入Solr的索引库

（本例子不实现商品信息维护功能所以直接使用dataImport-handler处理器完成索引创建）

搜索流程如下：

springmvc：

1、用户在页面输入关键字，点击“商品搜索”

2、Controller方法接收用户提交的关键字

3、action调用Service查询商品信息

Service:

4、service使用Solrj请求Solr服务查询商品信息

## 运行环境

jdk1.7

eclipse indigo

jar包：

spring3.2.0(包括 springmvc)

jstl

junit

log4j

solrJ相关jar包

## 搜索实现

【Service】

#### PO

/\*\*

\* 商品信息对象模型

\*/

**public** **class** Products {

//商品编号

**private** String pid;

//商品名称

**private** String name;

//商品分类

**private** **int** catalog;

//商品分类名称

**private** String catalog\_name;

//价格

**private** Float price;

//数量

**private** **int** number;

//商品描述

**private** String description;

//图片名称

**private** String picture;

//上架时间

**private** Date Release\_time;

查询模型：

**public** **class** ProductsQuery **extends** Products {

//关键字

**private** String keywords;

//商品价格起止

**private** Float price\_start;

**private** Float price\_end;

//排序字段

**private** String sortField;

//排序方式 asc/desc

**private** String sortType;

//当前页码

**private** Integer curPage;

//每页显示个数

**private** Integer rows;

//总页数，用于返回

**private** Integer pageCount;

//总记录数，用于返回在页面显示

**private** Long recordCount;

Service接口

**public** **interface** ProductSearchService {

//搜索商品

**public** List<Products> searchProducts(ProductsQuery productsQuery)**throws** Exception;

}

service接口实现类使用@service注解标识。

Service接口实现类：

@Service

**public** **class** **ProductSearchServiceImpl** **implements** **ProductSearchService** **{**

**// solr服务地址，当没有指定core时默认一个core提交，实际开发时将此地址配置在配置文件中**

**private** **String** solrUrl **=** "http://localhost:8088/solr"**;**

@Override

**public** List**<Products>** **searchProducts(ProductsQuery** productsQuery**)**

**throws** Exception **{**

**// 进行关键信息的校验**

**// 非空校验**

**if** **(**productsQuery **==** **null)** **{**

productsQuery **=** **new** **ProductsQuery();**

**}**

**// 如果当前页码没有传**

**if** **(**productsQuery**.**getCurPage**()** **==** **null)** **{**

**// 默认第一页**

productsQuery**.**setCurPage**(1);**

**}**

**// 如果每页显示个数没有传默认16个**

**if** **(**productsQuery**.**getRows**()** **==** **null)** **{**

productsQuery**.**setRows**(16);**

**}**

**// 定义solrServer对象，通过此对象发起solr的请求**

**SolrServer** solrServer **=** **new** *HttpSolrServer***(**solrUrl**);**

**// 定义solr的查询对象**

**SolrQuery** solrQuery **=** **new** *SolrQuery***();**

**// 设置查询条件**

**// 指定查询语句 q，必须要指定查询q**

**// 如果用户不输入关键要查询所有**

**if** **(**productsQuery**.**getKeywords**()** **==** **null**

**||** productsQuery**.**getKeywords**().**equals**(**""**))** **{**

solrQuery**.***set***(**"q"**,** "\*:\*"**);**

**}** **else** **{**

solrQuery**.***set***(**"q"**,**

"product\_keywords:" **+** productsQuery**.**getKeywords**());**

**}**

**//使用过虑，过虑商品分类**

**if(**productsQuery**.**getCatalog\_name**()!=null** **&&** **!**productsQuery**.**getCatalog\_name**().**equals**(**""**)){**

solrQuery**.***addFilterQuery***(**"product\_catalog\_name:"**+**productsQuery**.**getCatalog\_name**());**

**}**

**//过虑 ，价格范围**

**if(**productsQuery**.**getPrice\_start**()!=null){**

**if(**productsQuery**.**getPrice\_end**()!=null){**

**//拼接起止价格**

solrQuery**.***addFilterQuery***(**"product\_price:["**+**productsQuery**.**getPrice\_start**()** **+** " TO "**+** productsQuery**.**getPrice\_end**()** **+**"]"**);**

**}else{**

**//拼接起始价格和\***

solrQuery**.***addFilterQuery***(**"product\_price:["**+**productsQuery**.**getPrice\_start**()** **+** " TO \*]"**);**

**}**

**}**

**//进行价格排序**

**if(**productsQuery**.**getSortField**()!=null** **&&** **!**productsQuery**.**getSortField**().**equals**(**""**)){**

**if(**productsQuery**.**getSortType**()!=null** **&&** productsQuery**.**getSortType**().**equals**(**"desc"**)){**

solrQuery**.***setSort***(**"product\_price"**,** ORDER**.*desc*);**

**}else{**

solrQuery**.***setSort***(**"product\_price"**,** ORDER**.*asc*);**

**}**

**}**

**// 分页**

**// 指定每页显示个数**

**int** rows **=** productsQuery**.**getRows**();**

**// 指定开始下标，根据每页显示个数和当前页码计算**

**int** curPage **=** productsQuery**.**getCurPage**();**

**int** start **=** rows **\*** **(**curPage **-** **1);**

**//设置获取记录数**

solrQuery**.***setRows***(**rows**);**

**//设置开始下标**

solrQuery**.***setStart***(**start**);**

**//开启高亮**

solrQuery**.***setHighlight***(true);**

**//指定高亮 的field**

solrQuery**.***addHighlightField***(**"product\_name"**);**

**//指定高亮 前缀**

solrQuery**.***setHighlightSimplePre***(**"<span style=\"color:red\">"**);**

**//指定高亮 后缀**

solrQuery**.***setHighlightSimplePost***(**"</span>"**);**

**//请求查询，得到响应**

**QueryResponse** response **=** solrServer**.***query***(**solrQuery**);**

**//从响应中取出数据**

**SolrDocumentList** docs **=** response**.***getResults***();**

**//搜索到总记录数**

**long** numFound **=** docs**.***getNumFound***();**

**//设置商品总数，回传给conroller，再由controller传到页面**

productsQuery**.**setRecordCount**(**numFound**);**

**//计算总页数**

**int** pageCount **=** **(int)** **Math.*ceil*(**numFound **\*** **1.0/**rows**);**

productsQuery**.**setPageCount**(**pageCount**);**

Map**<String,** Map**<String,** List**<String>>>** highlighting **=** response**.***getHighlighting***();**

**//返回结果集对象**

List**<Products>** list **=** **new** *ArrayList***<Products>();**

**for(SolrDocument** doc**:**docs**){**

**//创建一个新商品对象**

**Products** products **=** **new** **Products();**

**//商品id**

**if(**doc**.***get***(**"id"**)!=null){**

**String** id **=** **(String)** doc**.***get***(**"id"**);**

products**.**setPid**(**id**);**

**}**

**//商品名称**

**if(**doc**.***get***(**"product\_name"**)!=null){**

**String** product\_name **=** **(String)** doc**.***get***(**"product\_name"**);**

products**.**setName**(**product\_name**);**

**}**

**//商品图片**

**if(**doc**.***get***(**"product\_picture"**)!=null){**

**String** product\_picture **=** **(String)** doc**.***get***(**"product\_picture"**);**

products**.**setPicture**(**product\_picture**);**

**}**

**//商品价格**

**if(**doc**.***get***(**"product\_price"**)!=null){**

**Float** product\_price **=** **Float.*parseFloat*(**doc**.***get***(**"product\_price"**).***toString***());**

products**.**setPrice**(**product\_price**);**

**}**

**//商品分类**

**if(**doc**.***get***(**"product\_catalog\_name"**)!=null){**

**String** product\_catalog\_name **=** **(String)**doc**.***get***(**"product\_catalog\_name"**);**

products**.**setCatalog\_name**(**product\_catalog\_name**);**

**}**

**//取出高亮信息，根据主键取出高亮信息**

Map**<String,** List**<String>>** map **=** highlighting**.***get***(**doc**.***get***(**"id"**).***toString***());**

**// 根据高亮field域名字取出高亮信息内容**

List**<String>** list\_l **=** map**.***get***(**"product\_name"**);**

**if(**list\_l**!=null** **&&** list\_l**.***get***(0)!=null){**

products**.**setName**(**list\_l**.***get***(0));**

**}**

list**.**add**(**products**);**

**}**

**return** list**;**

**}**

**}**

【Controller】

方法

@Controller

**public** **class** ProductSearchController {

@Autowired

**private** ProductSearchService productSearchService;

@RequestMapping("/search")

//商品搜索

**public** String search(Model model,@ModelAttribute ProductsQuery productsQuery) **throws** Exception{

//调用 service搜索商品

List<Products> list = productSearchService.searchProducts(productsQuery);

model.addAttribute("list", list);

**return** "search";

}

}

页面

参考“案例相关/search.jsp”

springmvc.xml

<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-3.2.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.2.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop-3.2.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx-3.2.xsd "*>

<!-- 组件扫描-->

<context:component-scan base-package=*"cn.itcast.solr"* />

<!-- mvc注解驱动 -->

<mvc:annotation-driven/>

<!-- 视频解析器 -->

<bean

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

<property name=*"prefix"* value=*"/WEB-INF/jsp/"* />

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

</bean>

</beans>

web.xml

<!-- springmvc前端控制器 -->

<servlet>

<servlet-name>springmvc</servlet-name> <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-clas>

<init-param>

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

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

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

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

</servlet-mapping>

<!-- post乱码 -->

<filter>

<filter-name>CharacterEncodingFilter</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>CharacterEncodingFilter</filter-name>

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

</filter-mapping>

## 【调试】

略