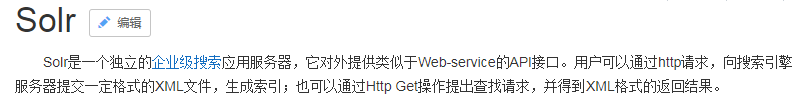
Solr课程内容教案

# Solr概述

## 什么是Solr



Solr是一个高性能，采用Java5开发，基于Lucene的全文搜索服务器。同时对其进行了扩展，提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文搜索引擎。

## Solr的下载

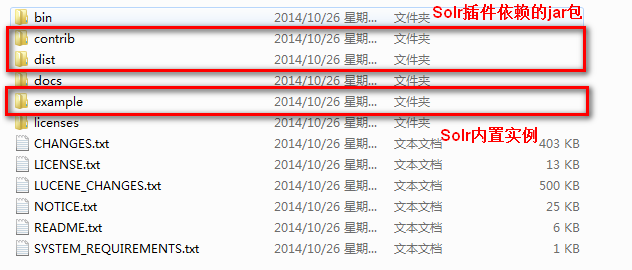
Solr是Apache下的一个开源项目，由于是基于Lucene的一个实现产品，因此将Solr放在Lucene项目当中，官方网站为：<http://lucene.apache.org/solr/>。最新版本为7.1.0，我们这里使用于Lucene相匹配的4.10.2版本进行我们的学习。

Solr的历史下载地址为：http://archive.apache.org/dist/lucene/solr/，我们可以从中下载4.10.2版本，我们这里已经下载完成了。

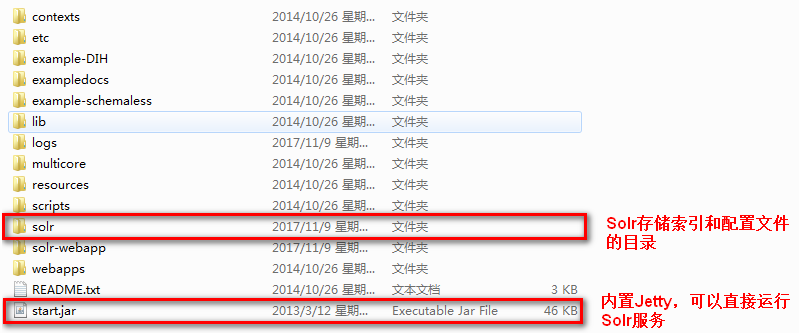


## Solr的下载

将下载的Solr文件解压缩即可获取Solr的内容，我们看到目录结构如下。



在example中包含Solr的完整案例，我们需要了解。



这里我们需要注意的是：

Solr目录用来存放Solr的Core，主要包含索引和配置文件。

solr-webapp目录中管理者Jetty运行的Solr服务去程序。

Webapps目录中的solr.war为solr服务的文本应用程序。

# 运行Solr服务

Solr实际上是一个web服务器应用程序，因此要启动Solr需要文本服务器，而在Solr实例代码中已经直接集成了Jetty服务器，我们可以直接使用，也可以单独配置和使用Tomcat服务器来启动Solr服务。

## 启动Solr服务的方式

### 使用Solr内置的Jetty服务器启动Solr

使用内置的Jetty来启动Solr服务器只需要在example目录下，执行start.jar程序即可，我们可以直接执行命令：java –jar start.jar。

当服务启动后，默认发布在8983端口，所以可以访问该端口来访问Solr服务。

### 将Solr部署到Tomcat中

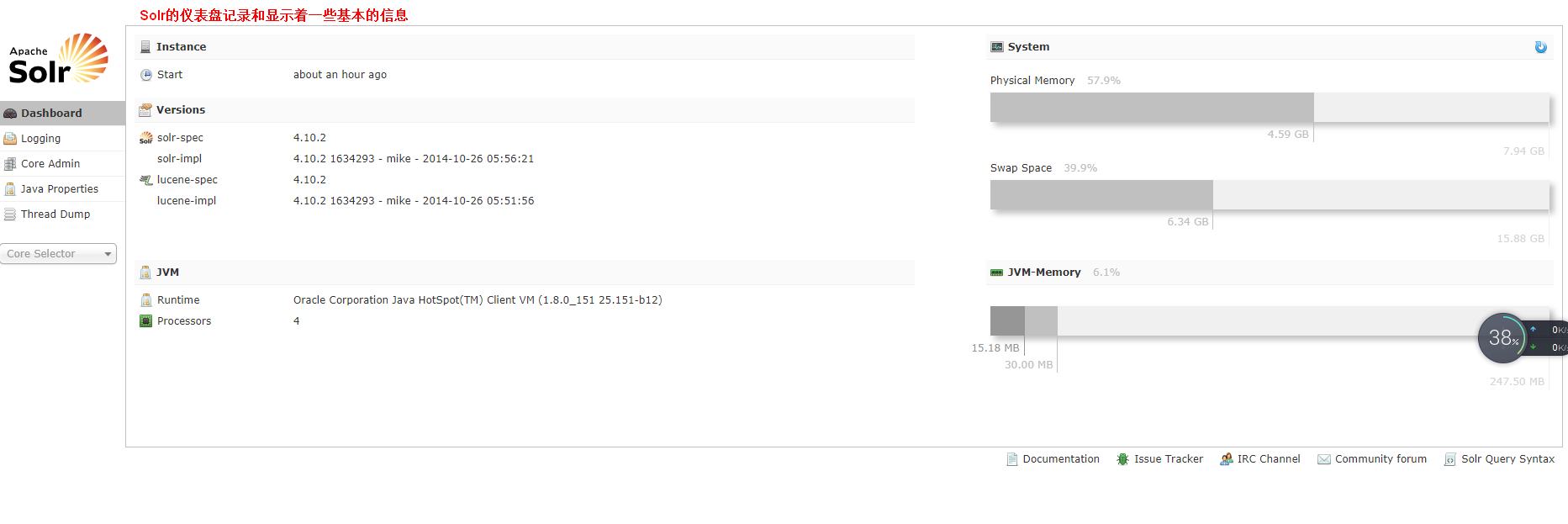
在Tomcat中部署Web服务，将solr-4.10.2/example/webapps/solr.war复制到自己的tomcat/webapps目录中，进行并解压。

将Solr用到的jar包拷贝到项目的lib目录下，在启动前还需要设置solr.solr.home启动参数，在catalina.bat文件添加配置：set "JAVA\_OPTS=-Dsolr.solr.home=d:/test/solr"。

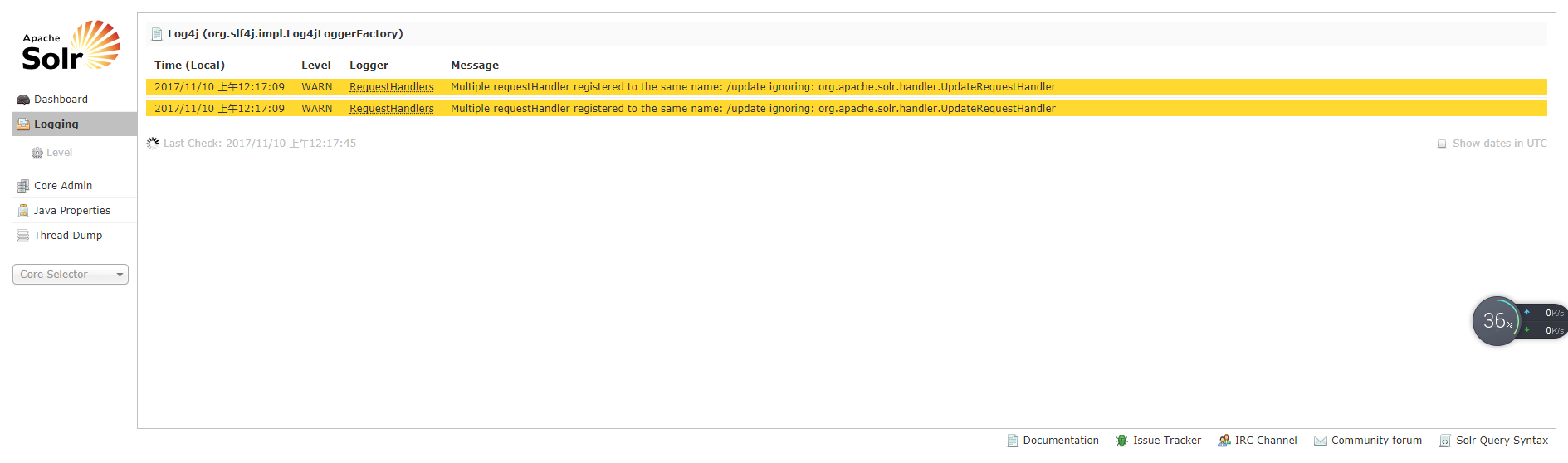
启动Tomcat则Solr的服务器发布就完成了。

## Solr管理页面

### DashBoard



### Logging

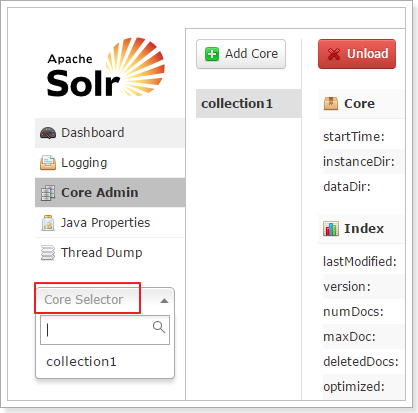


### Core Admin

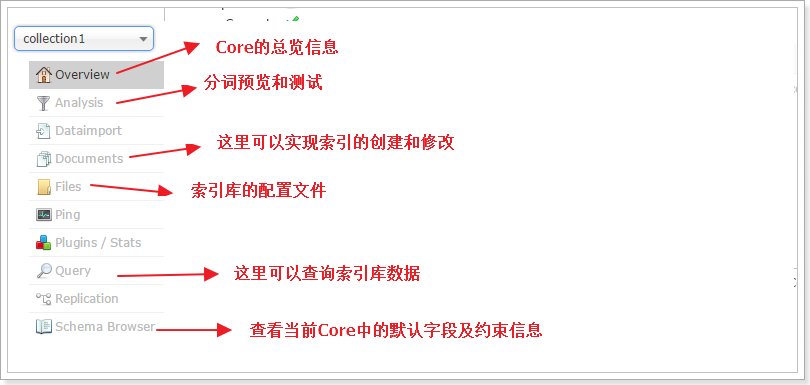
在Solr中，每一个Core，代表一个索引库，里面包含索引数据及其信息。Solr中可以拥有多个Core，也就同时管理多个索引库！就像在MySQL中可以有多个database一样。



### CoreSelector



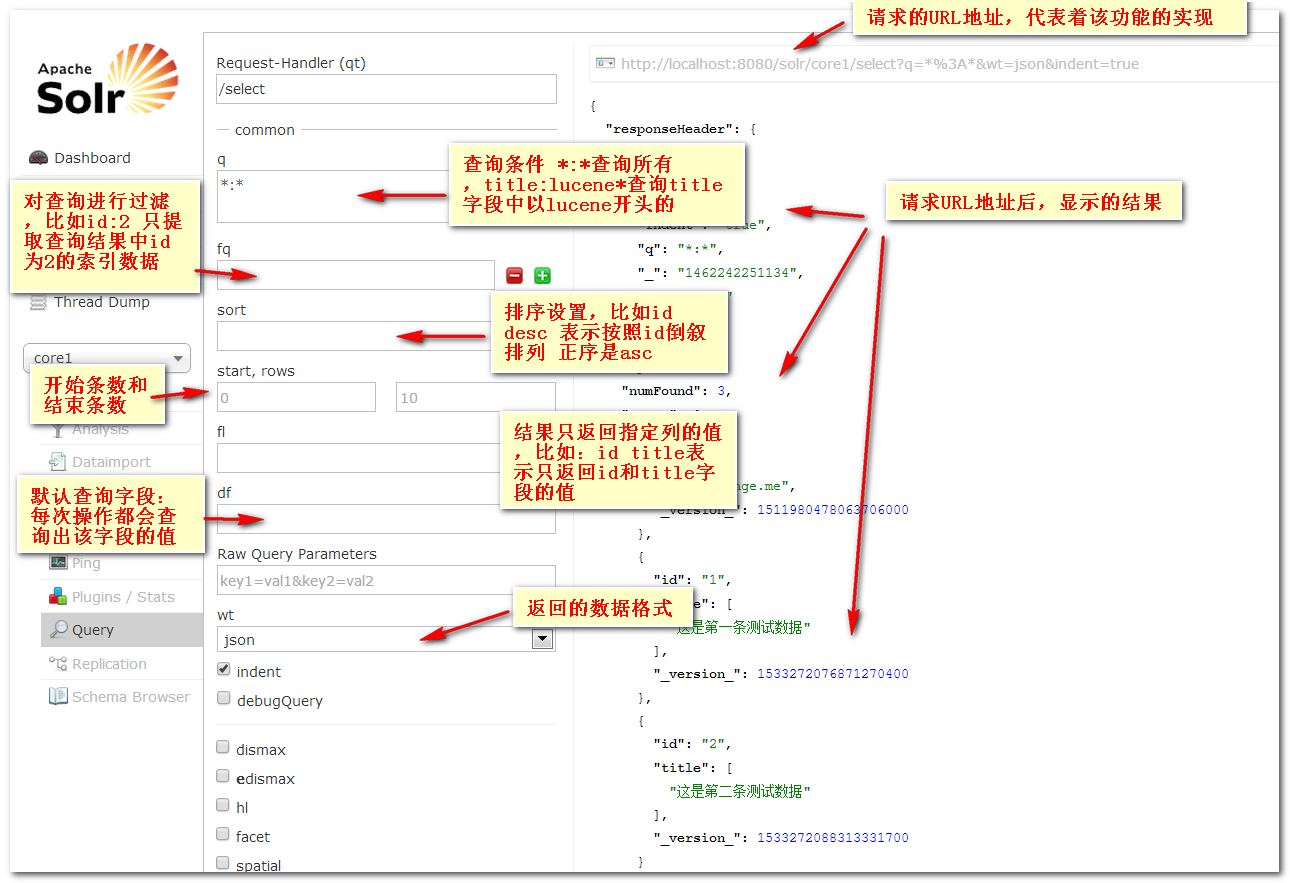
选择一个Core，可以进行更详细的操作：



#### 通过Solr管理界面添加索引数据



#### 通过Solr管理界面查询索引数据



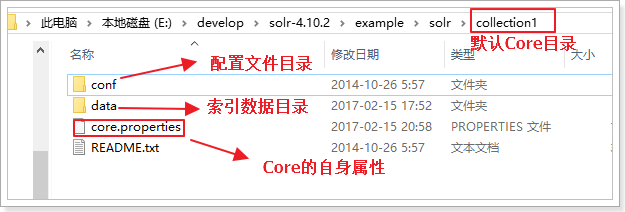
# Solr中的Core详解

## Core的概念

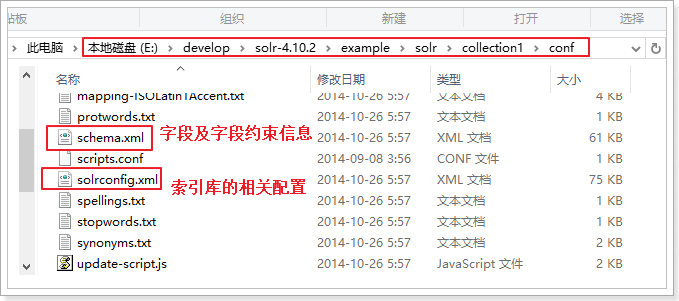
在Solr中一个core就是一个索引库，里面包含索引信息和配置文件。

## 目录结构

Core中有两个重要目录：conf和data



conf目录中有两个非常重要的配置文件：schema.xml和solrconfig.xml

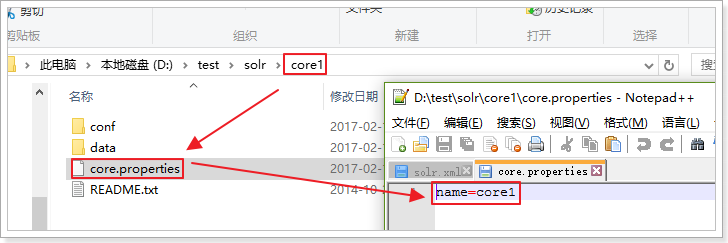


## Core的配置文件详解

### core.properties

#### 修改core名称

Core的属性文件，记录当前core的名称、索引位置、配置文件名称等信息。一般要求Core名称跟Core的文件夹名称一致！这里都是Collection1，我们可以手动修改这个属性。



此时重启Tomcat，可以看到core的名字已经改变！

#### 添加多个core

在solr目录下创建新的文件夹core2,作为新的core目录，创建conf目录和data目录，并且创建文件core.properties，从core1/conf目录下复制配置文件core2/conf/下。

### schema.xml

Solr中会提前对文档中的字段进行定义，并且在schema.xml中对这些字段的属性进行约束，例如：字段数据类型、字段是否索引、是否存储、是否分词等等。

#### 通过Field字段定义字段的属性信息

属性及含义：

|  |
| --- |
| name：字段名称  type：字段类型，指向的是本文件中的<fieldType>标签  indexed：是否创建索引  stored：是否被存储  multiValued：是否可以有多个值，如果字段可以有多个值，设置为true |

注意：在本文件中，有两个字段是Solr自带的字段，绝对不要删除：\_version节点和\_root节点。

#### 通过FieldType指定数据类型

属性及含义：

|  |
| --- |
| name：字段类型的名称，可以自定义，<field>标签的type属性可以引用该字段，来指定数据类型  class：字段类型在Solr中的类。StrField可索引不可分词。TextField字段可索引，可以分词，所以需要指定分词器  <analyzer>：这个子标签用来指定分词器 |

#### 唯一主键

Lucene中本来是没有主键的。删除和修改都需要根据词条进行匹配。而Solr却可以设置一个字段为唯一主键，这样增删改操作都可以根据主键来进行。

#### 动态字段

动态字段可以动态匹配，一般使用比较少，使用dynamicField来定义。

### solrconfig.xml

这个配置文件主要配置跟索引库和请求处理相关的配置。

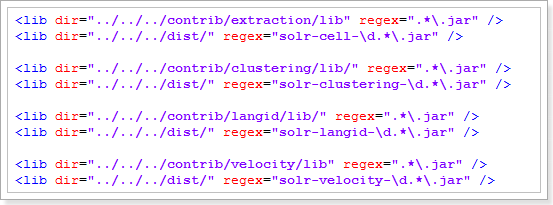
#### <lib/>标签

用途：配置插件依赖的jar包

注意事项：

o 如果引入多个jar包，要注意包和包的依赖关系，被依赖的包配置在前面

o 这里的jar包目录如果是相对路径，那么是相对于core所在目录



#### <requestHandler/>标签

用途：配置Solr处理各种请求（搜索/select、更新索引/update、等）的各种参数

主要参数：

o name：请求类型，例如：select、query、get、update

o class：处理请求的类

o initParams：可选。引用<initParams>标签中的配置

o <lst name="defaults">：定义各种缺省的配置，比如缺省的parser、缺省返回条数

# SolrJ的使用

## 概述

SolrJ是Apache官方提供的一套Java开发的，访问Solr服务的API，通过这套API可以让我们的程序与Solr服务产生交互，让我们的程序可以实现对Solr索引库的增删改查。

添加依赖：

|  |
| --- |
| <!-- Junit单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>  <dependency>  <groupId>org.apache.solr</groupId>  <artifactId>solr-solrj</artifactId>  <version>4.10.2</version>  </dependency>  <!-- Solr底层会使用到slf4j日志系统 -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>1.7.22</version>  </dependency>  <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>1.2</version>  </dependency> |

## 使用SolrJ添加或修改索引库数据

### 以Document形式添加或修改数据

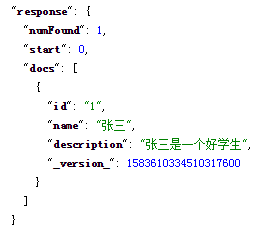
|  |
| --- |
| @Test  **public** **void** testInsertIndexByDocument() **throws** Exception {  //创建服务器对象  HttpSolrServer server = **new** HttpSolrServer("http://localhost:8983/solr/collection2");    //创建文档对象  SolrInputDocument document = **new** SolrInputDocument();    //添加索引数据  document.addField("id", "1");  document.addField("title", "这是第一条测试数据");    //添加文档  server.add(document);    //提交请求，如果ID对应的数据存在则进行更新操作，如果不存在则创建索引  server.commit();  } |



### 使用注解和JavaBean添加或修改数据

创建JavaBean，并且用注解标明要添加到索引库的字段，直接通过SolrServer添加JavaBean。

|  |
| --- |
| @Test  **public** **void** testInsertIndexByBean() **throws** Exception {  //创建服务器对象  HttpSolrServer server = **new** HttpSolrServer("http://localhost:8983/solr/collection2");    //添加数据  server.addBean(**new** Person("1","张三","张三是一个好学生"));    //提交请求，如果ID对应的数据存在则进行更新操作，如果不存在则创建索引  server.commit();  } |



## 使用SolrJ删除索引库数据

删除索引可以根据ID删除，也可以写一个查询条件，匹配到条件的都会被删除

|  |
| --- |
| @Test  **public** **void** testDleteIndex() **throws** Exception{  //创建服务器对象  HttpSolrServer server = **new** HttpSolrServer("http://localhost:8983/solr/collection2");    //根据条件进行删除  server.deleteByQuery("name:张三");    //提交请求  server.commit();  } |

## 使用SolrJ查询索引库数据

### 以Document形式返回查询结果

|  |
| --- |
| @Test  **public** **void** testQueryIndex() **throws** Exception{  //创建服务器对象  HttpSolrServer server = **new** HttpSolrServer("http://localhost:8983/solr/collection2");  //定义查询条件  SolrQuery query = **new** SolrQuery("name:李四");  //进行查询处理  QueryResponse queryResponse = server.query(query);  SolrDocumentList results = queryResponse.getResults();    System.***out***.println("找到了"+results.size()+"条记录");    //便利数据  **for** (SolrDocument solrDocument : results) {  System.***out***.println(solrDocument.get("id"));  System.***out***.println(solrDocument.get("name"));  System.***out***.println(solrDocument.get("description"));  }  } |

### 以JavaBean形式返回查询结果

|  |
| --- |
| @Test  **public** **void** testQueryIndexBean() **throws** Exception{  //创建服务器对象  HttpSolrServer server = **new** HttpSolrServer("http://localhost:8983/solr/collection2");  //定义查询条件  SolrQuery query = **new** SolrQuery("name:李四");  //进行查询处理  QueryResponse queryResponse = server.query(query);  List<Person> beans = queryResponse.getBeans(Person.**class**);    **for** (Person person : beans) {  System.***out***.println("id:"+person.getId());  System.***out***.println("name:"+person.getName());  System.***out***.println("description:"+person.getDescription());  }  } |

### SolrQuery对象的高级查询

在创建SolrQuery时，我们填写的Query语句，可以有以下高级写法：

查询语句中如果有特殊字符，需要转义，可以使用：””

1、匹配所有文档：\*:\* （通配符？和\*：“\*”表示匹配任意字符；“？”表示匹配出现的位置）

2、布尔操作：AND、OR和NOT布尔操作（推荐使用大写，区分普通字段）

3、子表达式查询（子查询）：可以使用“()”构造子查询。 比如：(query1 AND query2) OR (query3 AND query4)

4、相似度查询：

（1）默认相似度查询：title:appla~ ，此时编辑举例是2

（2）指定编辑举例的相似度查询：对模糊查询可以设置编辑举例，可选0~2的整数。

5、范围查询（Range Query）：Lucene支持对数字、日期甚至文本的范围查询。结束的范围可以使用“\*”通配符。

（1）日期范围（ISO-8601 时间GMT）：a\_begin\_date:[1990-01-01T00:00:00.000Z TO 1999-12-31T24:59:99.999Z]

（2）数字：salary:[2000 TO \*]

（3）文本：entryNm:[a TO a]

6、日期匹配：YEAR, MONTH, DAY, DATE (synonymous with DAY) HOUR, MINUTE, SECOND, MILLISECOND, and MILLI (synonymous with MILLISECOND)可以被标志成日期。

（1）r\_event\_date:[\* TO NOW-2YEAR]：2年前的现在这个时间

（2）r\_event\_date:[\* TO NOW/DAY-2YEAR]：2年前前一天的这个时间

### SolrQuery实现排序

|  |
| --- |
| 1. @Test 2. **public** **void** testQueryIndexSort() **throws** Exception{ 3. //创建服务器对象 4. HttpSolrServer server = **new** HttpSolrServer("http://localhost:8983/solr/collection2"); 5. //定义查询条件 6. SolrQuery query = **new** SolrQuery("\*:\*"); 7. //设置按ID排序 8. query.setSort("id", ORDER.***desc***); 9. //进行查询处理 10. QueryResponse queryResponse = server.query(query); 11. SolrDocumentList results = queryResponse.getResults(); 13. System.***out***.println("找到了"+results.size()+"条记录"); 15. //便利数据 16. **for** (SolrDocument solrDocument : results) { 17. System.***out***.println(solrDocument.get("id")); 18. System.***out***.println(solrDocument.get("title")); 19. } 20. } |

### SolrQuery实现分页

|  |
| --- |
| 1. @Test 2. **public** **void** testQueryIndexPages() **throws** Exception{ 3. //要查询的页数 4. **int** pageNum = 2; 5. //每页显示条数 6. **int** pageSize = 2; 7. //当前页的起始条数 8. **int** start = (pageNum - 1) \* pageSize; 10. //创建服务器对象 11. HttpSolrServer server = **new** HttpSolrServer("http://localhost:8983/solr/collection2"); 12. //定义查询条件 13. SolrQuery query = **new** SolrQuery("\*:\*"); 14. //设置按ID排序 15. query.setSort("id", ORDER.***desc***); 17. //设置起始条数 18. query.setStart(start); 19. //设置每页条数 20. query.setRows(pageSize); 21. //进行查询处理 22. QueryResponse queryResponse = server.query(query); 23. SolrDocumentList results = queryResponse.getResults(); 25. System.***out***.println("当前第" + pageNum + "页，本页共" + results.size() + "条数据。"); 27. //便利数据 28. **for** (SolrDocument solrDocument : results) { 29. System.***out***.println(solrDocument.get("id")); 30. System.***out***.println(solrDocument.get("title")); 31. } 32. } |

### SolrQuery实现高亮显示

|  |
| --- |
| 1. @Test 2. **public** **void** testQueryIndexHighlighting() **throws** Exception{ 3. //创建服务器对象 4. HttpSolrServer server = **new** HttpSolrServer("http://localhost:8983/solr/collection2"); 5. //定义查询条件 6. SolrQuery query = **new** SolrQuery("title:测试"); 7. //设置高亮的标签 8. query.setHighlightSimplePre("<em>"); 9. query.setHighlightSimplePost("</em>"); 10. //高亮字段 11. query.addHighlightField("title"); 13. //进行查询处理 14. QueryResponse queryResponse = server.query(query); 15. SolrDocumentList results = queryResponse.getResults(); 17. //获取高亮字段 18. Map<String, Map<String, List<String>>> highlighting = queryResponse.getHighlighting(); 20. System.***out***.println("找到了"+results.size()+"条记录"); 22. //便利数据 23. **for** (SolrDocument solrDocument : results) { 24. String id = solrDocument.get("id").toString(); 25. System.***out***.println("id:"+id); 26. System.***out***.println(highlighting.get(id).get("title").get(0)); 27. } 28. } |