# solrCloud基本概念和搭建

## 一、背景

由于公司pms系统，需要做solr全文搜索技术支持，做成duboo框架的RPC远程服务调用形式。为其它项目提供服务，再将我所了解的solr相关知识和搭建步骤记录，以便日后查看。

## 二、solrCloud简介

1. solr采用java5开发
2. 基于Lucene（一个用java编写的全文检索引擎工具包），进行一些扩展和封装，提供了比Lucene更丰富的查询语句，还实现了可配置，可扩展，查询语句优化。
3. 提供了一个完善的功能管理界面。
4. SolrCloud是Solr4.0版本以后基于Solr和Zookeeper的分布式搜索方案。SolrCloud是Solr的基于Zookeeper一种部署方式。Solr可以以多种方式部署，例如单机方式，多机Master-Slaver方式。

## 三、特色功能

1. 集中式的配置信息

使用ZK进行集中配置。启动时可以指定把Solr的相关配置文件上传Zookeeper，多机器共用。这些ZK中的配置不会再拿到本地缓存，Solr直接读取ZK中的配置信息。配置文件的变动，所有机器都可以感知到。另外，Solr的一些任务也是通过ZK作为媒介发布的。目的是为了容错。接收到任务，但在执行任务时崩溃的机器，在重启后，或者集群选出候选者时，可以再次执行这个未完成的任务。

1. 自动容错

SolrCloud对索引分片，并对每个分片创建多个Replication。每个Replication都可以对外提供服务。一个Replication挂掉不会影响索引服务。更强大的是，它还能自动的在其它机器上帮你把失败机器上的索引Replication重建并投入使用。

1. 近实时搜索

立即推送式的replication（也支持慢推送）。可以在秒内检索到新加入索引。

4. 查询时自动负载均衡

SolrCloud索引的多个Replication可以分布在多台机器上，均衡查询压力。如果查询压力大，可以通过扩展机器，增加Replication来减缓。

5.自动分发的索引和索引分片

发送文档到任何节点，它都会转发到正确节点。

6.事务日志

事务日志确保更新无丢失，即使文档没有索引到磁盘。

7. 索引存储在HDFS

索引的大小通常在G和几十G，上百G的很少，这样的功能或许很难实用。但是，如果你有上亿数据来建索引的话，也是可以考虑一下的。我觉得这个功能最大的好处或许就是和下面这个“通过MR批量创建索引”联合实用。

8. 强大的RESTful API

通常你能想到的管理功能，都可以通过此API方式调用。这样写一些维护和管理脚本就方便多了。

## 四、概念

**1.Collection**：在SolrCloud集群中逻辑意义上的完整的索引。它常常被划分为一个或多个Shard，它们使用相同的Config Set。如果Shard数超过一个，它就是分布式索引，SolrCloud让你通过Collection名称引用它，而不需要关心分布式检索时需要使用的和Shard相关参数。

**2.Config Set**: Solr Core提供服务必须的一组配置文件。每个config set有一个名字。最小需要包括solrconfig.xml (SolrConfigXml)和schema.xml (SchemaXml)，除此之外，依据这两个文件的配置内容，可能还需要包含其它文件。它存储在Zookeeper中。Config sets可以重新上传或者使用upconfig命令更新，使用Solr的启动参数bootstrap\_confdir指定可以初始化或更新它。

**3.Core**: 也就是Solr Core，一个Solr中包含一个或者多个Solr Core，每个Solr Core可以独立提供索引和查询功能，每个Solr Core对应一个索引或者Collection的Shard，Solr Core的提出是为了增加管理灵活性和共用资源。在SolrCloud中有个不同点是它使用的配置是在Zookeeper中的，传统的Solr core的配置文件是在磁盘上的配置目录中。

**4.Leader**: 赢得选举的Shard replicas。每个Shard有多个Replicas，这几个Replicas需要选举来确定一个Leader。选举可以发生在任何时间，但是通常他们仅在某个Solr实例发生故障时才会触发。当索引documents时，SolrCloud会传递它们到此Shard对应的leader，leader再分发它们到全部Shard的replicas。

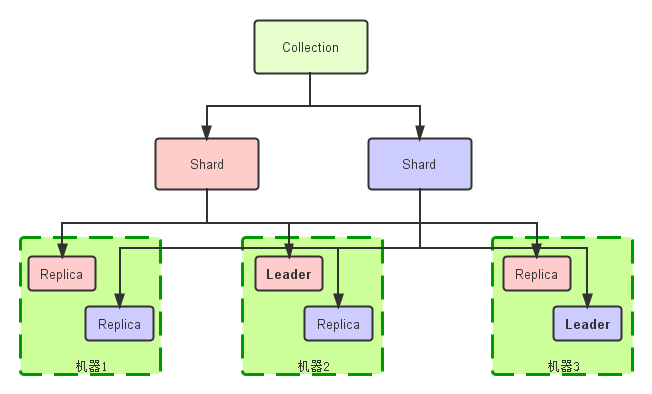
**5.Replica**: Shard的一个拷贝。每个Replica存在于Solr的一个Core中。一个命名为“test”的collection以numShards=1创建，并且指定replicationFactor设置为2，这会产生2个replicas，也就是对应会有2个Core，每个在不同的机器或者Solr实例。一个会被命名为test\_shard1\_replica1，另一个命名为test\_shard1\_replica2。它们中的一个会被选举为Leader。

**6.Shard**: Collection的逻辑分片。每个Shard被化成一个或者多个replicas，通过选举确定哪个是Leader。

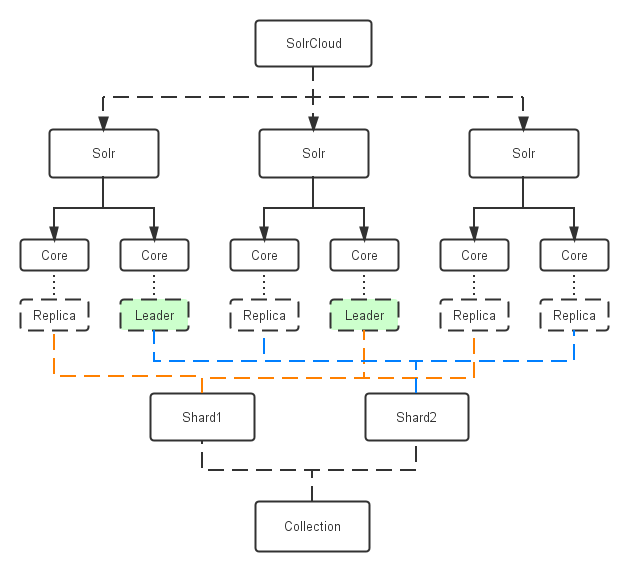
**7.Zookeeper**: Zookeeper提供分布式锁功能，对SolrCloud是必须的。它处理Leader选举。Solr可以以内嵌的Zookeeper运行，但是建议用独立的，并且最好有3个以上的主机。

## 五、架构图

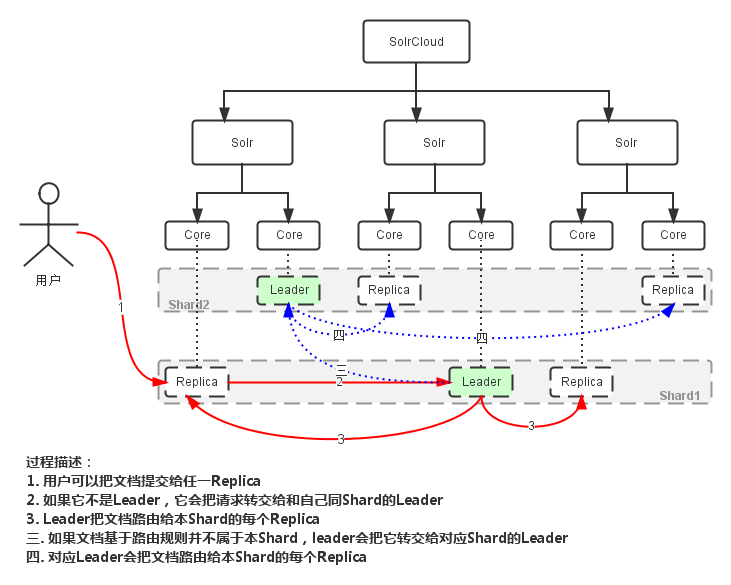
1.索引（collection）的逻辑图



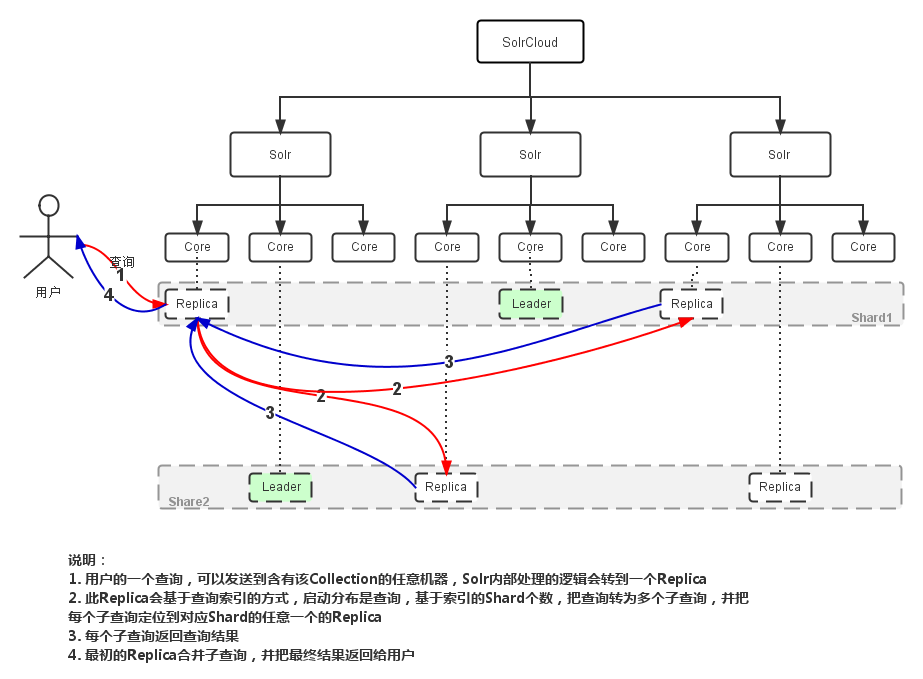
2．Solr和索引对照图



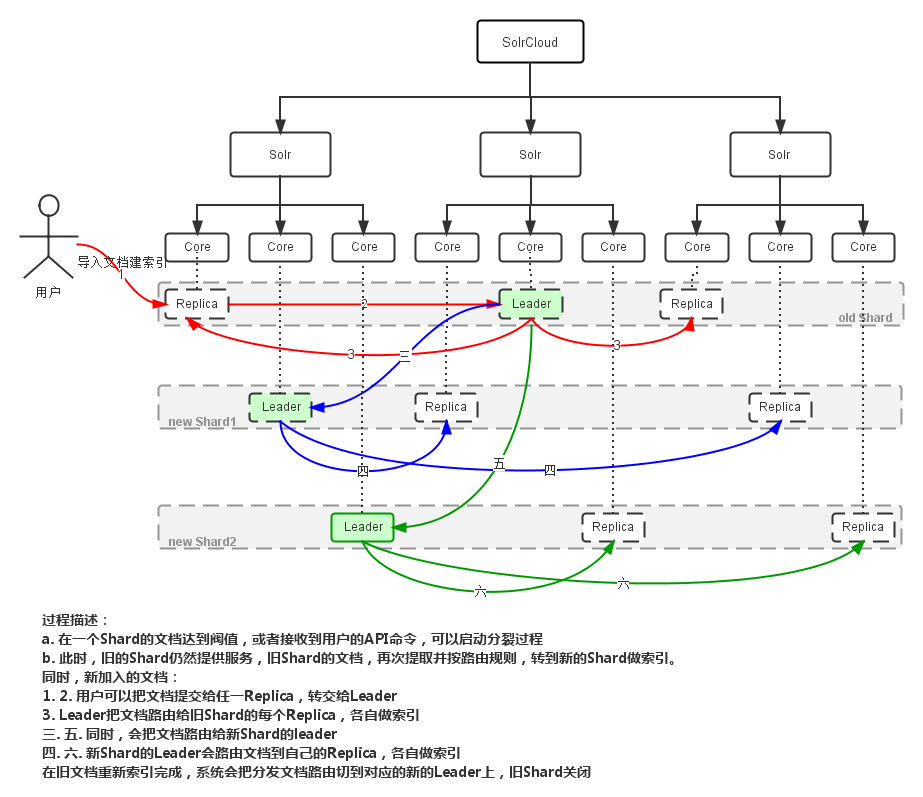
1. 创建索引过程



5. 分布式查询



1. Shard Splitting 分片分裂

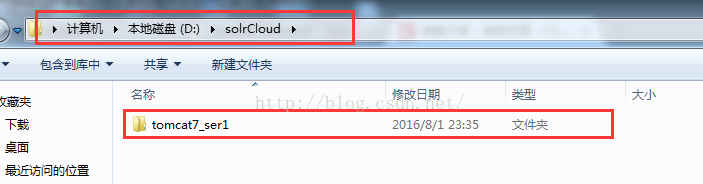


## 六、部署过程

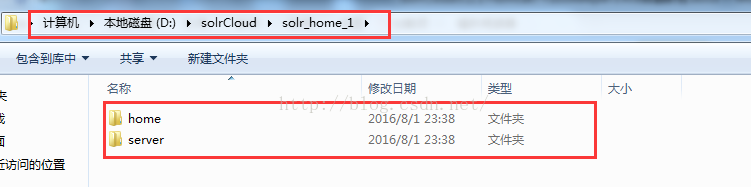
#### 1、部署单机版Tomcat7 +Solr5.2.1

第一步：在D盘根目录下建立solrCloud目录。并把apache-tomcat-7.0.56.zip解压到solrCloud目录下，复制apache-tomcat-7.0.56并重命名为tomcat7\_ser1

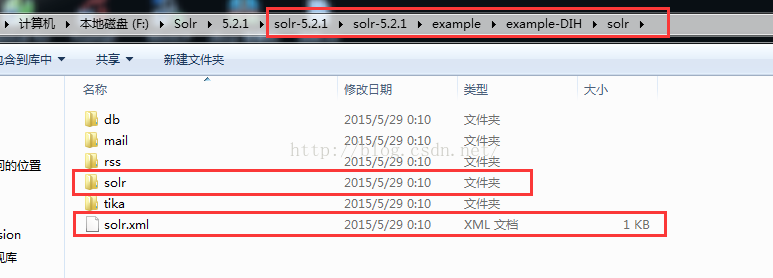
如图所示：

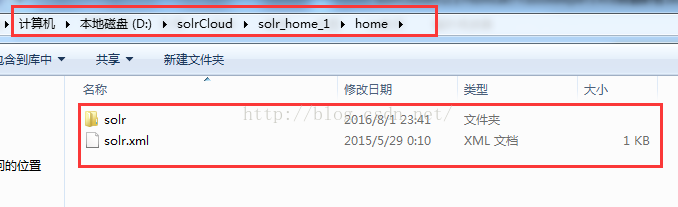


第二步，在solrCloud目录下新建solr\_home\_1 文件夹如下图，并在目录下，创建 home 和 server 文件夹

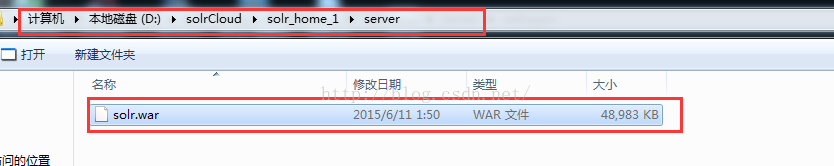


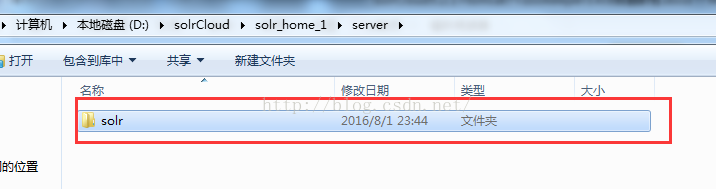
第三步：把solr-5.2.1\example\example-DIH\solr指定文件复制到solr\_home\_1\home目录下,如下图





第四步：把solr-5.2.1\server\webapps\solr.war复制到solr\_home\_1\server目录下。将solr.war解压,解压后将solr.war文件删除,如下图





第五步：修改D:/solrCloud/solr\_home\_1/server/solr/WEB-INF目录下的web.xml文件，里面修改solr/home的value值

<env-entry>

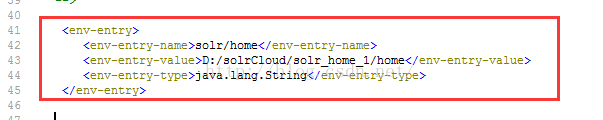
   <env-entry-name>solr/home</env-entry-name>

   <env-entry-value>D:/solrCloud/solr\_home\_1/home</env-entry-value>

   <env-entry-type>java.lang.String</env-entry-type>

</env-entry>

如下图：



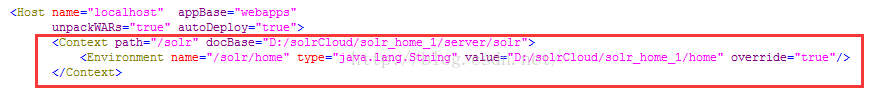
第六步：因为我们未将solr服务放到tomcat webapps下面，所以需要修改D:/solrCloud/tomcat7\_ser1/conf/server.xml指向solr服务路径。

在Host节点下加入如下代码：

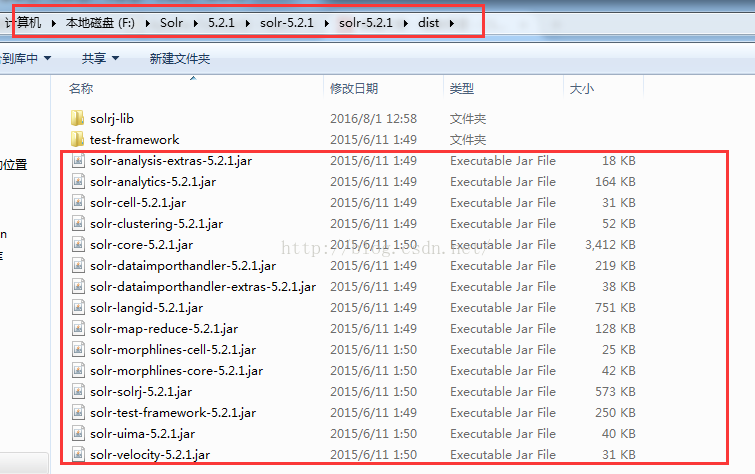
<context docbase="D:/solrCloud/solr\_home\_1/server/solr" path="/solr">

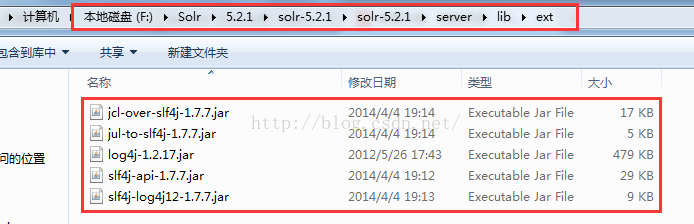
    <environment name="/solr/home" override="true" type="java.lang.String" value="D:/solrCloud/solr\_home\_1/home">

</environment></context>

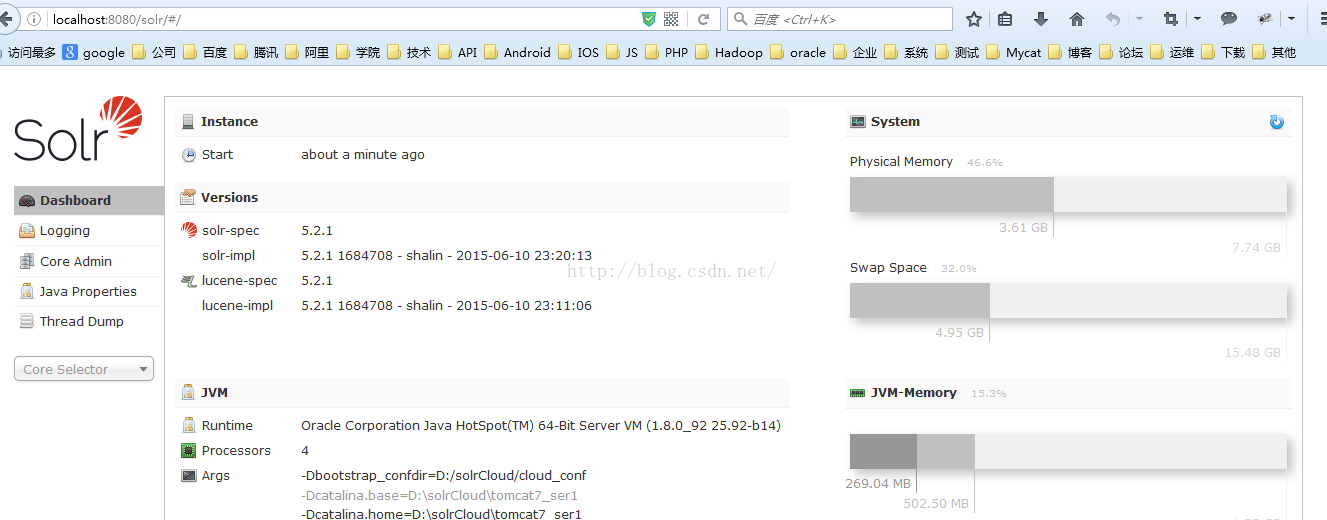


第七步：将solr-5.2.1\dist和solr-5.2.1\server\lib\ext目录下的jar包复制到solr\_home\_1\server\solr\WEB-INF\lib下



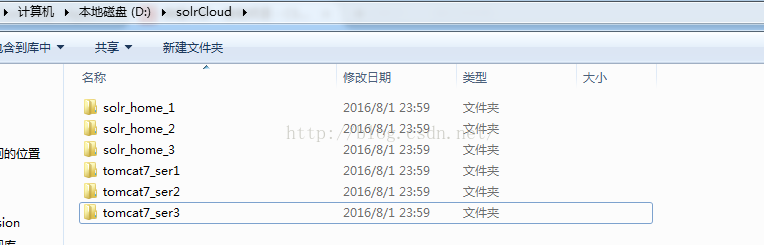


第八步：现在单机版本的就已经配置OK了。启动tomcat,访问<http://localhost:8080/solr>



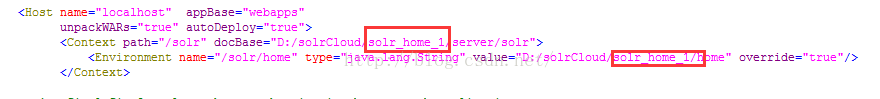
#### 2、配置多Tomcat+solr同时运行

第一步：  
把tomcat7\_ser1复制2份，分别命名tomcat7\_ser2，tomcat7\_ser3；  
把solr\_home\_1复制2份，分别命名solr\_home\_2，solr\_home\_3。  
目录结构如下

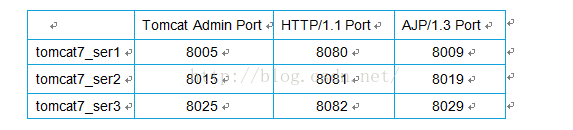


第二步：修改D:\solrCloud\tomcat7\_ser\*\conf\server.xml配置

1. 修改 tomcat7\_ser\* 分别对应 各自solrhome\* 和 solr 服务 。  
   如下图：tomcat7\_ser1对应solr\_home\_1、2对应2 、3对应3



1. 为了三个tomcat能够在一台机器上同时启动，需要在server.xml修改tomcat的端口信息。修改方案如下：



第三步：配置各个tomcat和solr之间的关系,修改D:\solrCloud\solr\_home\_\*\server\solr\WEB-INF目录下的web.xml文件(对应关系)

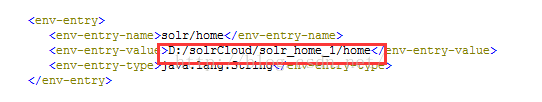
<env-entry>

   <env-entry-name>solr/home</env-entry-name>

   <env-entry-value>D:/solrCloud/solr\_home\_1/home</env-entry-value>

   <env-entry-type>java.lang.String</env-entry-type>

</env-entry>



第四步：验证修改是否成功，依次启动三个Tomcat。并在[浏览器](http://www.2cto.com/os/liulanqi/)输入如下的URL

http://localhost:8080/solr/

http://localhost:8081/solr/

<http://localhost:8082/solr/>

如果都能正常访问到solr的admin页面，那么说明配置是成功的。否则就需要检查哪里错了或者遗漏了。

#### 3、配置ZooKeeper集群



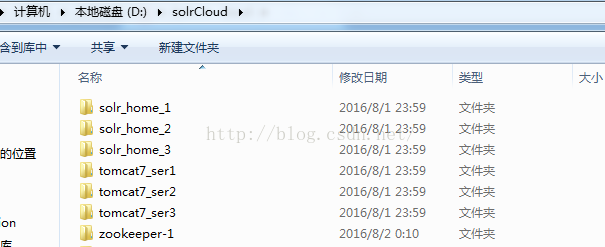




这部分的内容与前面tomcat+solr是没有关联的

第一步：解压zookeeper-3.4.6.tar.gz到D:\solrCloud目录，重命名为zookeeper-1。

如图所示：



第二步：把D:\solrCloud\zookeeper-1\conf\目录下的zoo\_sample.cfg修改为zoo.cfg。并写入如下的配置参数：

tickTime=2000

initLimit=10

syncLimit=5

dataDir=D:/solrCloud/zookeeper-1/data

dataLogDir=D:/solrCloud/zookeeper-1/datalog

clientPort=2181

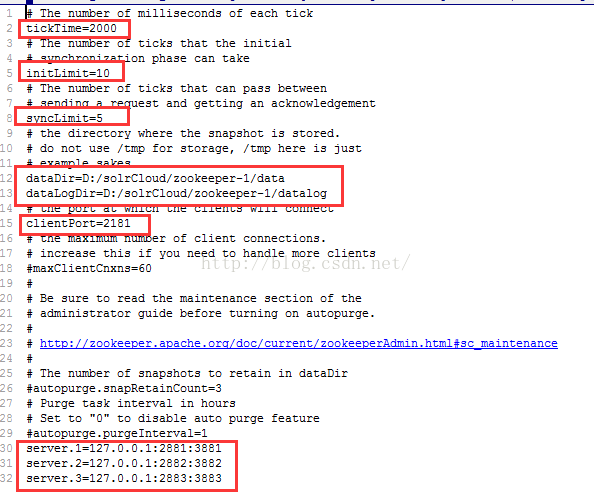
#autopurge.purgeInterval=1

server.1=127.0.0.1:2881:3881

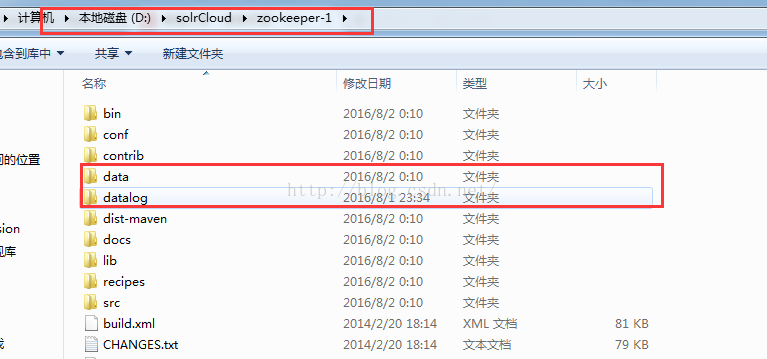
server.2=127.0.0.1:2882:3882

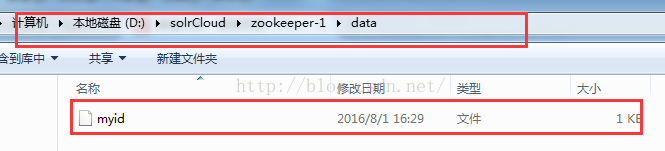
server.3=127.0.0.1:2883:3883

如下图所示：

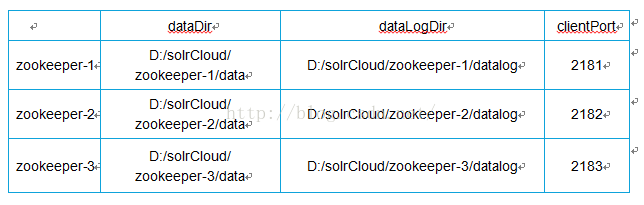


并且按照配置创建相应的data和datalog目录。如果不不创建目录是无法正常启动的



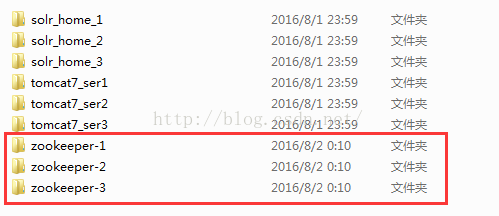


第三步：把zookeeper-1复制2份，分别命名为zookeeper-2、zookeeper-3。然后修改各个zookeeper-\*的conf目录下zoo.cfg的dataDir和dataLogDir和clientPort。修改方案如下：

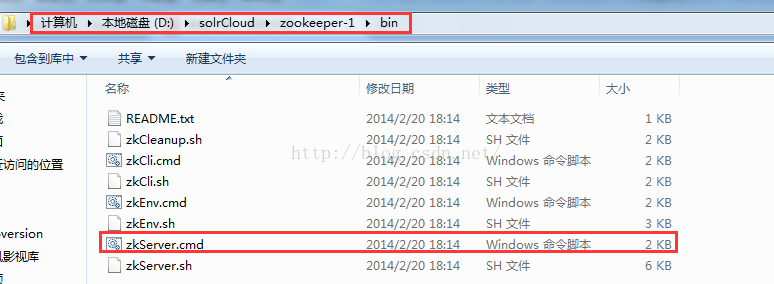


并且修改每个data目录下的myid文件中的内容。zookeeper-1是1，zookeeper-2是2，zookeeper-3是3

如下图所示：



第四步：启动三个zookeeper。并验证是否配置成功。目录结构以及启动目录如下图:



注意：连接第一台时有异常信息，不用管，等都连接起来就没有异常了。

进入cmd 命令，输入:netstat-ano|findstr 2181 查看端口是否启用

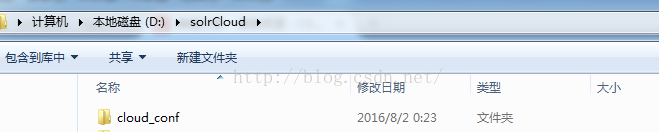


#### 搭建Tomcat7+ solr-5.2.1+zookeeper3.4.6集群

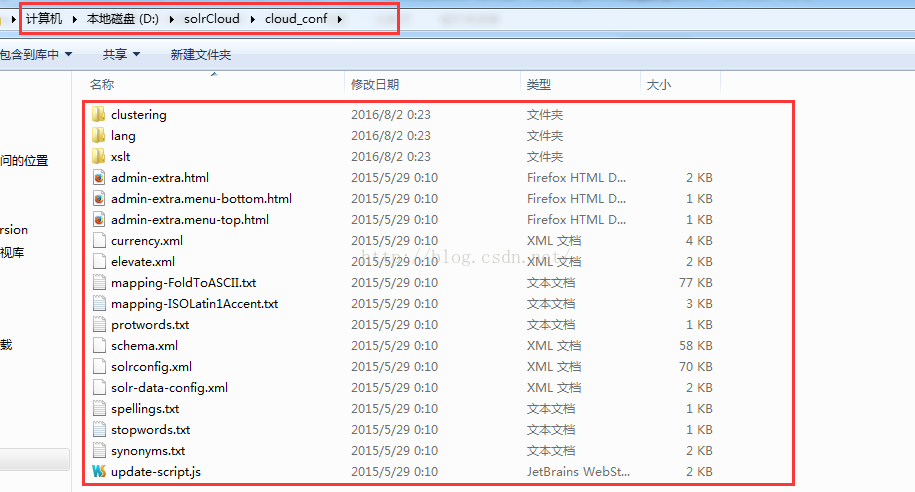
前面tomcat+solr能够启动和访问了，而且zookeeper也能启动成功了。接下来就需要把他们关联起来。大家看那么多贴子应该知道（当然不知道也要记住这个知识点）：**solrcloud的所有配置需要zookeeper统一管理**。

第一步:将需要的配置库集中放到一个目录中。

在D:\solrCloud下创建cloud\_conf文件夹作为配置库，如下图：



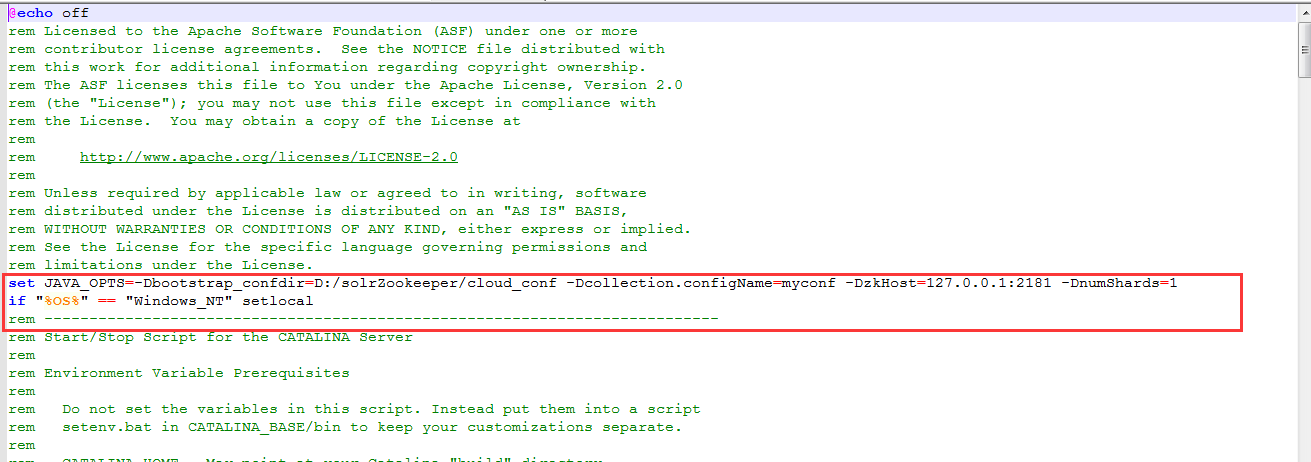
将solr-5.2.1\server\solr\configsets\sample\_techproducts\_configs\conf下配置文件复制到cloud\_conf文件夹中。(也可以使用solr-5.2.1\example\example-DIH\solr\solr\conf中的配置文件，我这里复制的就是solr-5.2.1\example\example-DIH\solr\solr\conf中的配置文件)



第二步:配置zookeeper路径

在D:\solrCloud\tomcat7\_ser1\bin\catalina.bat文件中,我是在 setlocal上一行添加：

set JAVA\_OPTS=-Dbootstrap\_confdir=D:/solrCloud/cloud\_conf -Dcollection.configName=myconf -DzkHost=127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183 -DnumShards=3



**配置参数解释:**

-DzkHost是用来指定zookeeper服务器的ip和端口。

-Dnumshareds=3用来指定当前集群中分片数为3

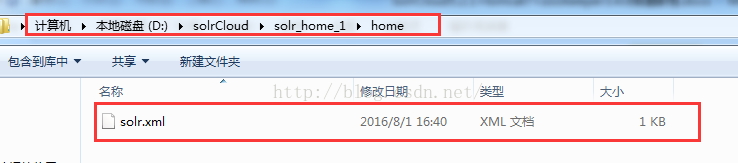
-Dcollection.configName是在指定你的配置文件上传到zookeeper后的名字，省略这个参数将导致配置名字为默认的“configuration1”

-Dbootstrap\_confdir ：zooKeeper需要准备一份集群配置的副本，所以这个参数是告诉SolrCloud这些配置是放在哪里。同时作为整个集群共用的配置文件。**可以看作是第一份solr配置。因为后续我们可以通过上传，来实现多collection。**

其余tomcat7\_ser\* catalina.bat设置

set JAVA\_OPTS= -DzkHost=127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183

第三步: 因为配置由zookeeper统一管理了，所以home下面的solr配置就不起作用了。删除D:\solrCloud\solr\_home\_\*\home\solr文件夹, 修改配置solr.xml,如下图



同时，solr.xml改成如下所示：

<!--?xml version="1.0" encoding="UTF-8" standalone="yes"?-->

<solr>

<solrcloud>

<str name="host">${host:}</str>

<int name="hostPort">${jetty.port:8080}</int>

<str name="hostContext">${hostContext:solr}</str>

<bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>

<int name="zkClientTimeout">${zkClientTimeout:30000}</int>

<int name="distribUpdateSoTimeout">${distribUpdateSoTimeout:600000}</int>

<int name="distribUpdateConnTimeout">${distribUpdateConnTimeout:60000}</int>

</solrcloud>

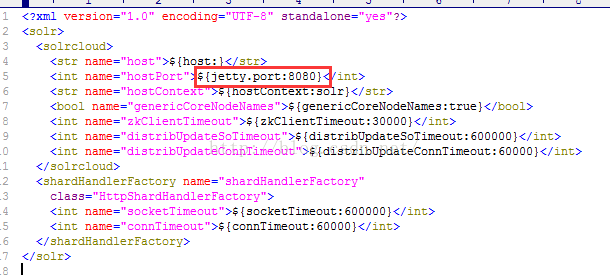
<shardhandlerfactory class="HttpShardHandlerFactory" name="shardHandlerFactory">

<int name="socketTimeout">${socketTimeout:600000}</int>

<int name="connTimeout">${connTimeout:60000}</int>

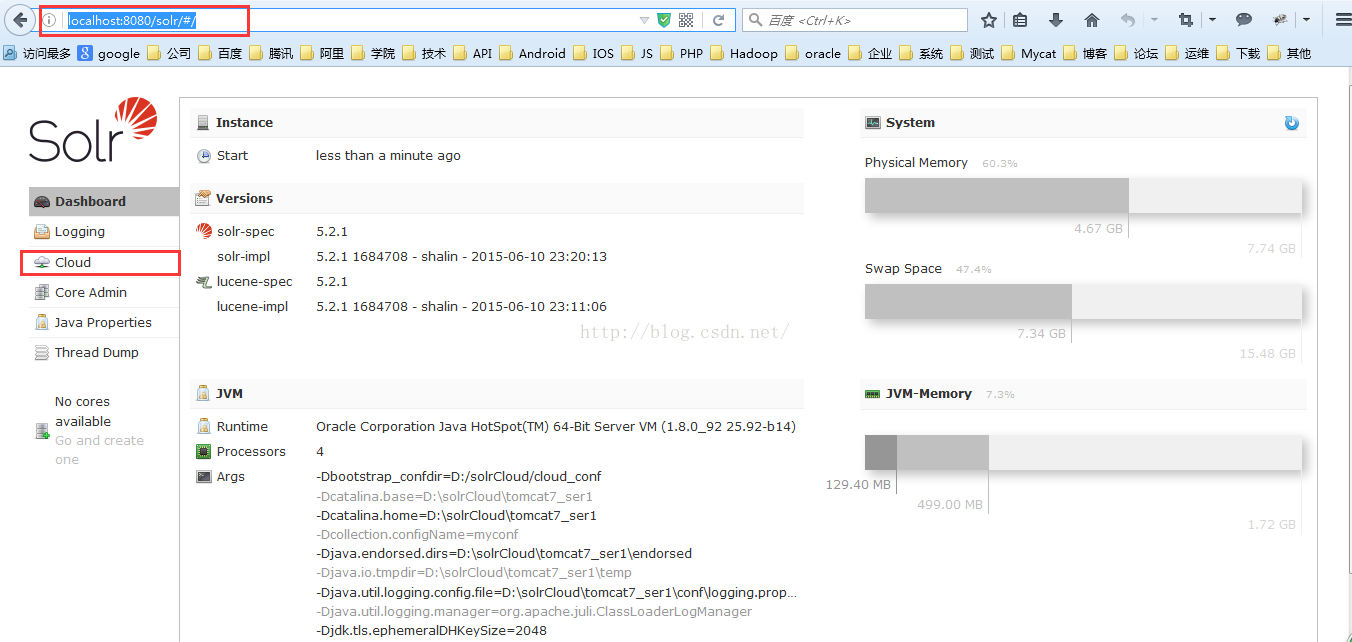
</shardhandlerfactory>

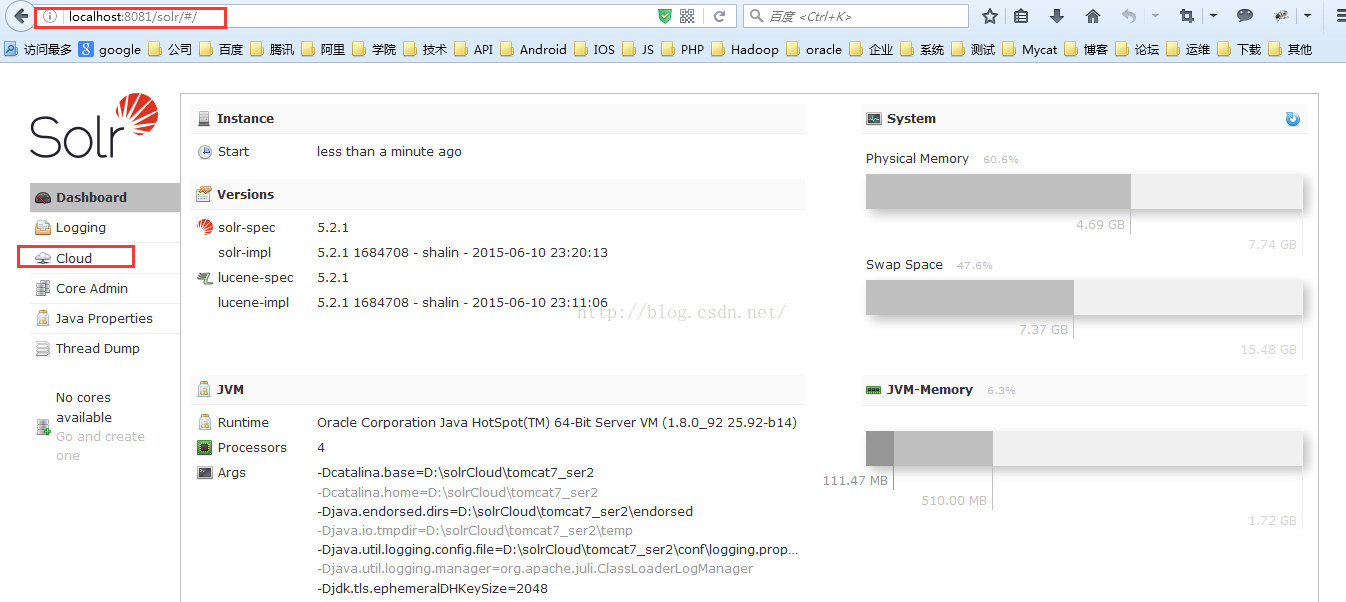
</solr>

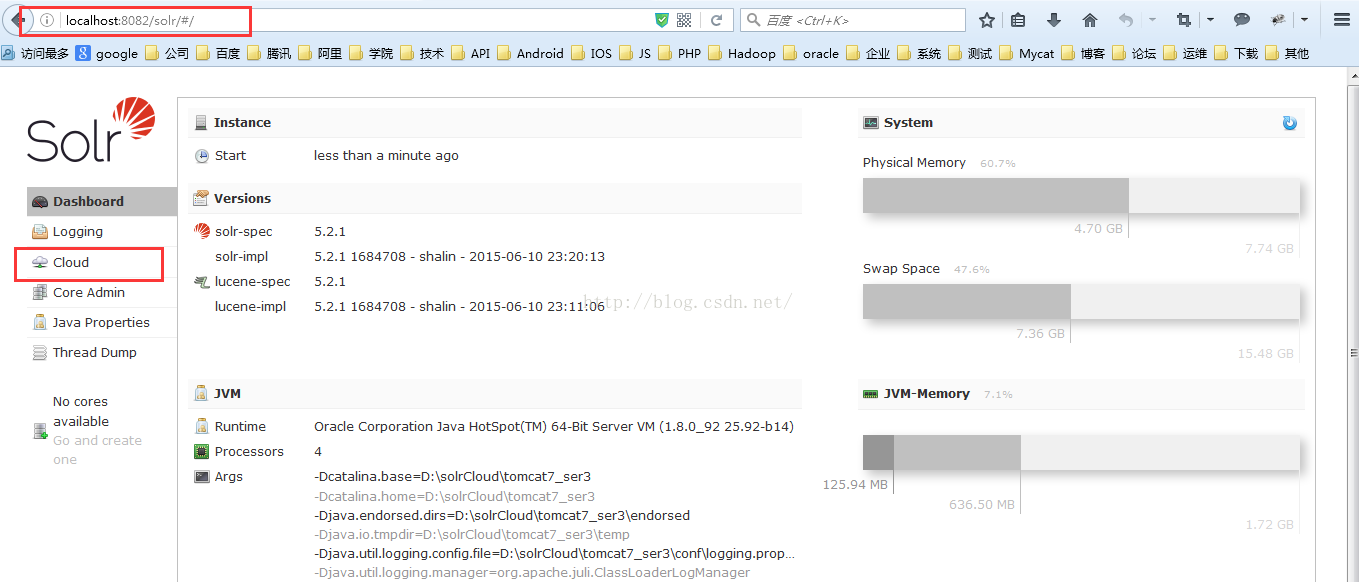


hostPort 修改为对应的tomcat端口号 8080 80818082

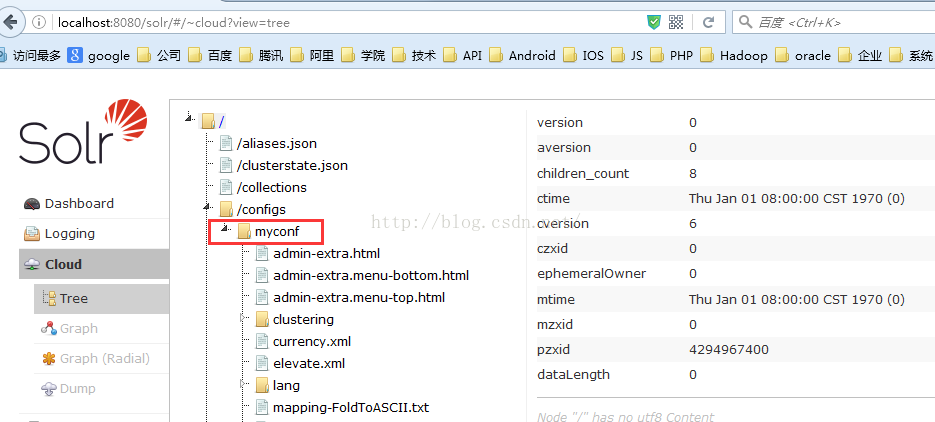
第四步:重启tomcat。访问任何一个端口都成，比如： [http://localhost:8081](http://localhost:8081/),能正常访问，且菜单中，出现了Cloud说明我们已经部署成功了







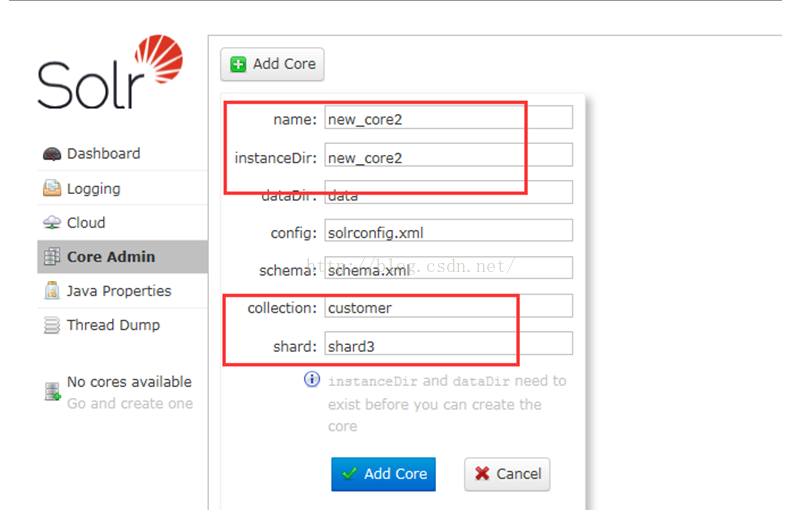
我们可以看下Cloud 的 Tree 下面就可以看到我们上传的配置文件，我上传时 -Dcollection.configName=myconf 所以这里也是显示myconf。



问题二： 如何创建Core？

答：输入 name , instanceDir, collection, shard，点击AddCore。 说明下，当collection 名称样的core 索引是通用的。（反之collection不一样则索引之前不可相互访问了）

即 你在 new\_core 下面创建的索引，在new\_core1下面也是能够查询到的。 当我们创建完core之后，分支图就已经出现了。如下：





## 七、schema介绍

solr在早期的版本中，是用scheam.xml文件管理集群中字段的配置的，但是在solr5之后，将配置文件的名字改成了manage-schema，所以网上大多数资 料都是在讲schema.xml文件，这两个文件都是一回事。manage-schema是用于配置集群中字段的属性的配置文件，主要是设置字段能提供的各种功能 ，比如字段能否被索引，能否返回，是不是需要分词，是不是必填字段等等。

field标签：属性  
name：字段名，就是我们查询的字段名  
type：字段类型，主要是string,int,long,time，文本，文件等，每个类型对应不同的特性，包括分词和高亮  
indexed：true/false,是否能被索引  
stored：true/false，是否被存储，就是查询的时候返回值里是否包含这一列  
required：true/false，是否是必填项  
multiValued：true/false，是否是多值字段，一遍是用在聚合查询  
  
dynamicField标签：属性  
dynamicField标签和field标签一样，不同的地方在于dynamicField的name可以动态匹配那么的值，比如\*is，这样可匹配以\_is为结尾的所有列。  
  
copyField标签：属性  
copyField标签是一个起到一个复制的功能，我们可以把多个字段的值copy到一个字段上，查询这个copy的字段时，就可以匹配查询多个字段  
  
fieldType标签：属性  
fieldType标签是定义了field的type类型，包括一些已经定义好的常见基本类型，也可以自定义类型，在这个标签中，我们可以设置当前类型是否需要 分词，是否需要设置过滤器做一些前置或者后置的操作。  
  
unionkey标签：属性  
unionkey标签指定了当前配置文件的主键字段是什么，和[**数据库**](http://lib.csdn.net/base/mysql)中的主键功能一样。

## 八、collections 、core管理

Collections API 地址

<https://cwiki.apache.org/confluence/display/solr/Collections+API#CollectionsAPI-api1>

记录一些常用的访问地址：

删除索引：http://127.0.0.1:8080/solr/datatest/update/?stream.body=<delete><query>\*:\*</query></delete>&stream.contentType=text/xml;charset=utf-8&commit=true

## 九、solr从数据库中导入数据全量索引和增量索引

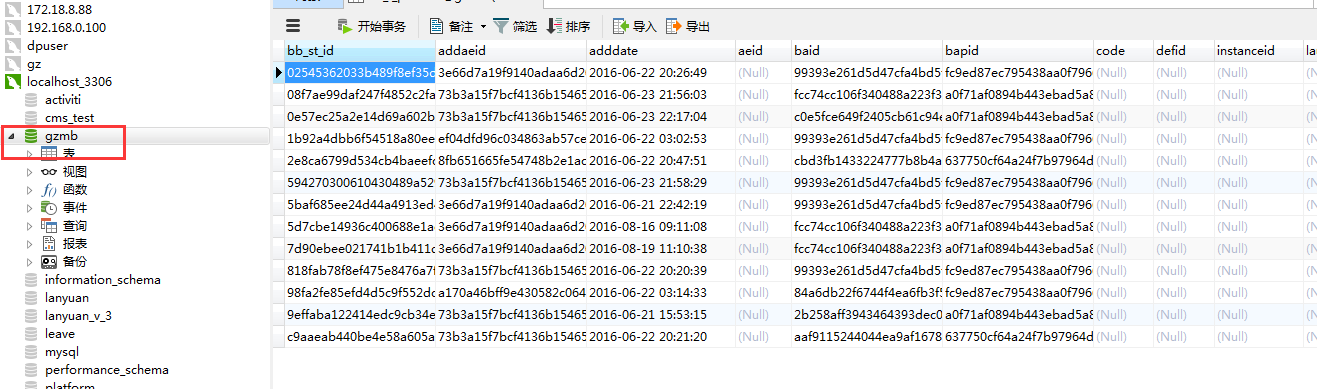
### 一、全量索引

1.使用mysql做测试库

需要的jar包有：solr-dataimporthandler-5.2.1.jar solr-dataimporthandler-extras-5.2.1.jar 这两个包主要是做solr数据库导入，还需要Mysql连接jdbc jar包

mysql-connector-java-5.1.18-bin.jar 将这三个包放到tomcat solr webapp的WEB-INF\lib目录下。

2.随便建一个测试表。



3.修改配置文件，solrconfig.xm信息。

加入节点：

<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">

<lst name="defaults">

<str name="config">data-config.xml</str>

</lst>

</requestHandler>

1. 新建一个data-config.xml文件，与solrconfig.xml同一个目录下。内容为

<dataConfig>

<dataSource type="JdbcDataSource"

driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost/gzmb"

user="root"

password="123456" />

<document>

<entity name="bb\_t\_person" transformer="DateFormatTransformer" pk="bb\_st\_id"

query="SELECT bb\_st\_id, bb\_st\_name, adddate FROM bb\_t\_person WHERE bb\_st\_id >= ${dataimporter.request.bb\_st\_id}">

<field column="bb\_st\_id" name="bb\_st\_id"/>

<field column="bb\_st\_name" name="bb\_st\_name"/>

<field column='adddate' dateTimeFormat='yyyy-MM-dd HH:mm:ss' />

</entity>

</document>

</dataConfig>注意：其中蓝色部分是参数条件，后面可以在传入参数，做条件建索引。

1. 修改schema.xml文件,配置索引字段和select字段一样。

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<schema name="work" version="1.0">

<fields>

<field indexed="true" multiValued="false" name="bb\_st\_name" stored="true" type="text\_cn"/>

<field indexed="true" multiValued="false" name="adddate" stored="true" type="date"/>

<field indexed="true" multiValued="false" name="update" stored="true" type="date"/>

<field indexed="true" multiValued="false" name="bb\_st\_id" stored="true" type="string"/>

<field indexed="true" multiValued="false" name="\_version\_" stored="true" type="long"/>

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

</fields>

<uniqueKey>bb\_st\_id</uniqueKey>

<defaultSearchField>text</defaultSearchField>

<copyField dest="text" source="bb\_st\_name"/>

<types>

<fieldType class="solr.TrieIntField" name="int" positionIncrementGap="0" precisionStep="0"/>

<fieldType class="solr.StrField" name="string" omitNorms="true" sortMissingLast="true"/>

<fieldType class="solr.TrieDateField" name="date" positionIncrementGap="0" precisionStep="0"/>

<fieldType class="solr.TrieDoubleField" name="double" positionIncrementGap="0" precisionStep="0"/>

<fieldType class="solr.TrieLongField" name="long" positionIncrementGap="0" precisionStep="0"/>

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

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

</fieldType>

<fieldType class="solr.TextField" name="text\_en" positionIncrementGap="100">

<analyzer type="index">

<tokenizer class="solr.StandardTokenizerFactory"/>

<filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords\_en.txt"/>

<filter class="solr.LowerCaseFilterFactory"/>

<filter class="solr.EnglishPossessiveFilterFactory"/>

<filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>

<filter class="solr.PorterStemFilterFactory"/>

</analyzer>

<analyzer type="query">

<tokenizer class="solr.StandardTokenizerFactory"/>

<filter class="solr.SynonymFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.txt"/>

<filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords\_en.txt"/>

<filter class="solr.LowerCaseFilterFactory"/>

<filter class="solr.EnglishPossessiveFilterFactory"/>

<filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>

<filter class="solr.PorterStemFilterFactory"/>

</analyzer>

</fieldType>

</types>

</schema>

1. 从数据库建索引



说明：

Custom Parameters填入参数条件

Clean选项，是指是否删除未匹配到的数据。也就是在数据库select结果中没有，而solr索引库中存在，则删除。

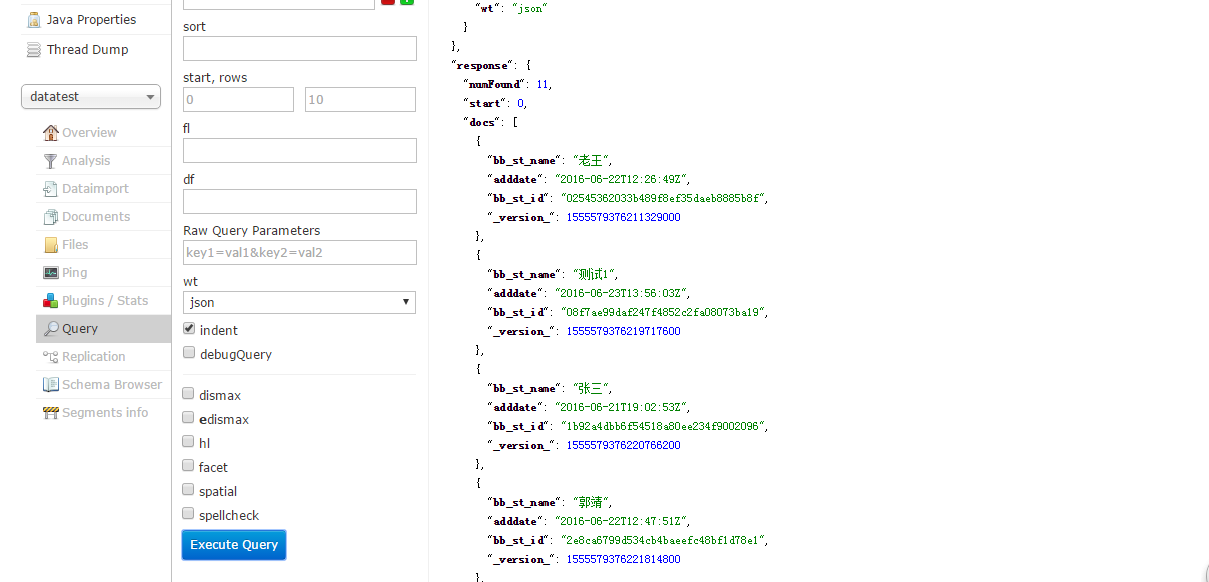
也可以使用这个地址直接访问：

http://localhost:8899/solr/mycore/dataimport?command=full-import&clean=true&commit=true&wt=json&indent=true&entity=solr\_test&verbose=false&optimize=false&debug=false&id=1

返回

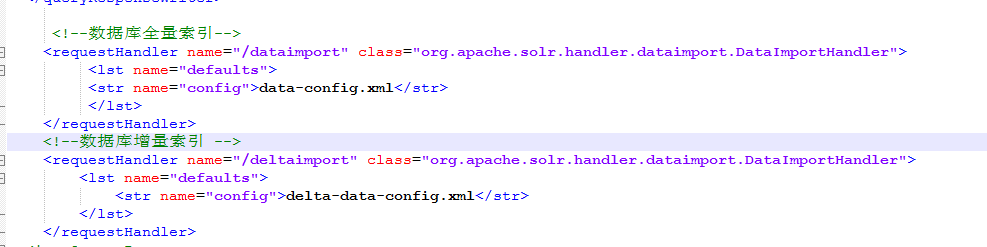


这时查询索引，就已经把满足条件的数据建好索引了。



### 二、增量索引

1.在solrConfig.xml配置文件中再增加一个节点。



代码：

<!--数据库增量索引 -->

<requestHandler name="/deltaimport" class="org.apache.solr.handler.dataimport.DataImportHandler">

<lst name="defaults">

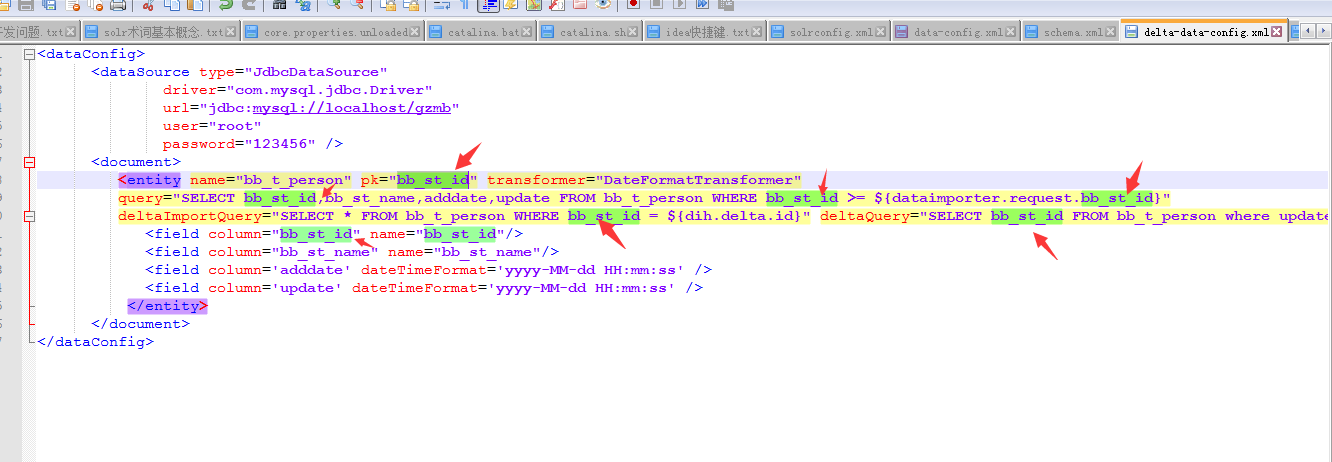
<str name="config">delta-data-config.xml</str>

</lst>

</requestHandler>

2．在solrConfig同级目录下新建文件delta-data-config.xml

并配置：



\*注意pk要和数据库一致。

代码：<dataConfig>

<dataSource type="JdbcDataSource"

driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost/gzmb"

user="root"

password="123456" />

<document>

<entity name="bb\_t\_person" pk="bb\_st\_id" transformer="DateFormatTransformer"

query="SELECT bb\_st\_id,bb\_st\_name,adddate,update FROM bb\_t\_person WHERE bb\_st\_id >= ${dataimporter.request.bb\_st\_id}"

deltaImportQuery="SELECT \* FROM bb\_t\_person WHERE bb\_st\_id = ${dih.delta.id}" deltaQuery="SELECT bb\_st\_id FROM bb\_t\_person where update > '${dataimporter.update}' ">

<field column="bb\_st\_id" name="bb\_st\_id"/>

<field column="bb\_st\_name" name="bb\_st\_name"/>

<field column='adddate' dateTimeFormat='yyyy-MM-dd HH:mm:ss' />

<field column='update' dateTimeFormat='yyyy-MM-dd HH:mm:ss' />

</entity>

</document>

</dataConfig>

update字段要在schema.xml文件中配置，<field name="update" type="date" indexed=true" stored="true"/>

query : 首先查询出所有满足bb\_st\_id > ${dataimporter.request.id} 的数据

deltaQuery ： 从query的数据中查询所有update大于上一次同步${dataimporter.last\_index\_time}时间的bb\_st\_id（即为需要增量同步的数据），该bb\_st\_id = ${dih.delta. bb\_st\_id }

deltaImportQuery ：根据 deltaQuery 返回的id查找所有信息，为增量同步提供数据源

${dih.delta. bb\_st\_id } ： 记录本次要索引的id bb\_st\_id ${dataimporter. last\_index\_time} ： 最后一次索引的时间，即上一次同步的时间

**注意：update字段很重要，为增量同步提供数据变更依据，一定为时间类型，具体字段名可自定义，同时也要注意solr服务器与mysql服务器时间同步问题**

**3.在schema.xml配置文件中新增字段update**

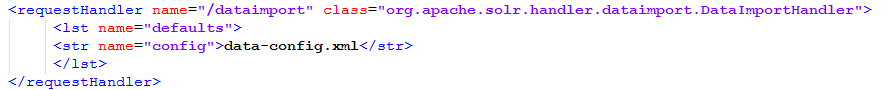
****

**4.配置完毕，启动solr服务器。**

### 三、文件内容建索引

主要针对(doc)|(pdf)|(xls)|(ppt)|(docx)|(pptx)格式建索引

1. solrCloud.xml配置信息



代码:

<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">

<lst name="defaults">

<str name="config">data-config.xml</str>

</lst>

</requestHandler>

1. 在solrCloud.xml同级目录下创建data-config.xml 并配置



代码：

<dataConfig>

<dataSource type="BinFileDataSource" />

<document>

<entity name="files" dataSource="binary" rootEntity="false"

processor="FileListEntityProcessor"

baseDir="E:/temp" fileName=".\*.(doc)|(pdf)|(xls)|(ppt)|(docx)"

recursive="true">

<field column="fileAbsolutePath" name="id" />

<field column="fileSize" name="size" />

<field column="fileLastModified" name="lastModified" />

<entity

name="documentImport"

processor="TikaEntityProcessor"

url="${files.fileAbsolutePath}"

format="text">

<field column="file" name="fileName"/>

<field column="Author" name="author" meta="true"/>

<field column="title" name="title" meta="true"/>

<field column="text" name="text"/>

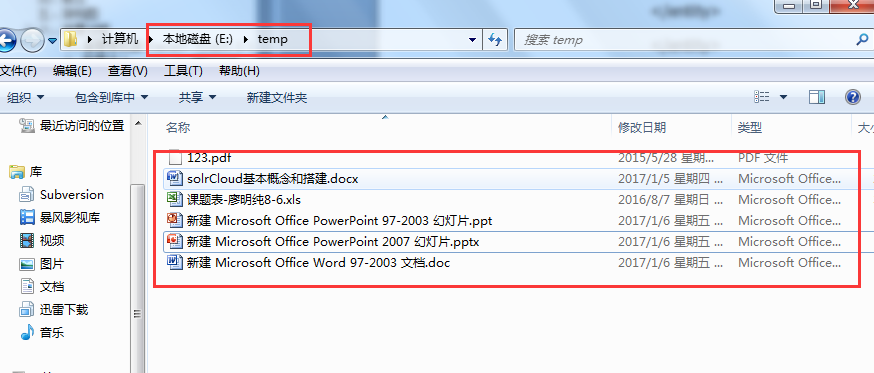
</entity>

</entity>

</document>

</dataConfig>

在**baseDir** 的位置添加一些满足格式的文件，供测试。



说明：

**fileName** :（必选）使用正则表达式匹配文件

·        **baseDir** : (必选) 文件目录

·        **recursive** : 是否递归的获取文件，默认false

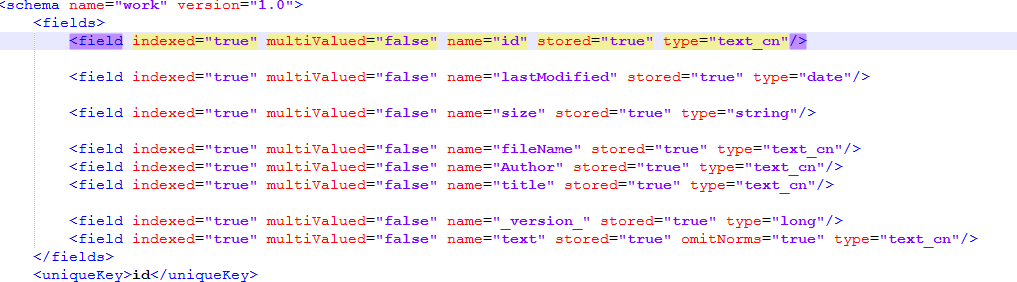
·        **rootEntity** :在这里必须是false(除非你只想索引文件名)。在默认情况下，document元素下就是根实体了，如果没有根实体的话，直接在实体下面的实体将会被看做跟实体。对于根实体对应的数据库中返回的数据的每一行，solr都将生成一个document

·        **dataSource** :如果你是用solr1.3，那就必须设为"null"，因为它没使用任何dataSourde。不需要在solr1.4中指定它，它只是意味着我们不创建一个dataSource实例。在大多数情况下，只有一个DataSource（JdbcDataSource），当使用FileListEntityProcessor 的时候DataSource不是必须的

·        **processor**:只有当datasource不是RDBMS时才是必须的

·        **onError** :默认是"abort"，"skip"表示跳过当前文档，"continue"表示对错误视而不见

1. 配置schema.xml文件



代码：

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<schema name="work" version="1.0">

<fields>

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

<field indexed="true" multiValued="false" name="lastModified" stored="true" type="date"/>

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

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

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

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

<field indexed="true" multiValued="false" name="\_version\_" stored="true" type="long"/>

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

</fields>

<uniqueKey>id</uniqueKey>

<!--<defaultSearchField>text</defaultSearchField>-->

<!--<copyField dest="text" source="bb\_st\_name"/>-->

<types>

<fieldType class="solr.TrieIntField" name="int" positionIncrementGap="0" precisionStep="0"/>

<fieldType class="solr.StrField" name="string" omitNorms="true" sortMissingLast="true"/>

<fieldType class="solr.TrieDateField" name="date" positionIncrementGap="0" precisionStep="0"/>

<fieldType class="solr.TrieDoubleField" name="double" positionIncrementGap="0" precisionStep="0"/>

<fieldType class="solr.TrieLongField" name="long" positionIncrementGap="0" precisionStep="0"/>

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

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

</fieldType>

<fieldType class="solr.TextField" name="text\_en" positionIncrementGap="100">

<analyzer type="index">

<tokenizer class="solr.StandardTokenizerFactory"/>

<filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords\_en.txt"/>

<filter class="solr.LowerCaseFilterFactory"/>

<filter class="solr.EnglishPossessiveFilterFactory"/>

<filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>

<filter class="solr.PorterStemFilterFactory"/>

</analyzer>

<analyzer type="query">

<tokenizer class="solr.StandardTokenizerFactory"/>

<filter class="solr.SynonymFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.txt"/>

<filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords\_en.txt"/>

<filter class="solr.LowerCaseFilterFactory"/>

<filter class="solr.EnglishPossessiveFilterFactory"/>

<filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>

<filter class="solr.PorterStemFilterFactory"/>

</analyzer>

</fieldType>

</types>

</schema>

1. 启动solr服务

有可能会报两个 找不到类的错误

C:\Users\Administrator\Documents\Tencent Files\308402945\Image\C2C\C}JK%Y512}F01_PZK]0V}$5.png

需要在solr/webapp/lib下添加jar包

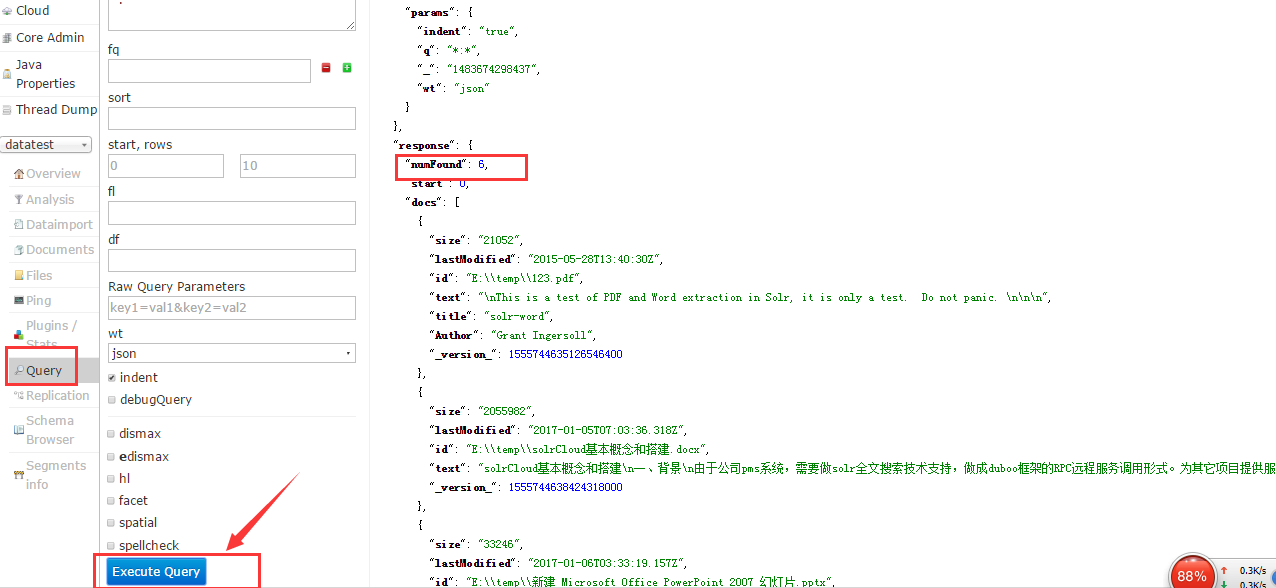




再重启，访问solr，显示。



Solr后台没报错，再查询索引，确认是否已经添加好了。



## 十、Solr -- 查询语法/参数

#### 1. 常用查询参数

| **参数** | **描述** |
| --- | --- |
| defType | 指定用于处理查询语句(参数q的内容)的查询解析器，eg:defType=lucene |
| sort | 指定响应的排序方式：升序asc或降序desc.同时需要指定按哪个字段进行排序。eg: sort=price desc,score asc |
| start | 指定显示查询结果的开始位置，默认是0 |
| rows | 指定一次显示多少行查询结果，默认是10 |
| fq | 指定用于对查询结果进行过滤的过滤器(也看作是一种query) eg: fq=price:[100 To \*]&fq=setction:0 |
| fl | 指定查询结果中返回的字段，该字段的stored=”true”或docValues=”true” ,eg:fl=id,title,product(price, popularity) |
| debug | 指定查询结果中携带额外的调试信息：时间信息debug=timing，“explain”信息debug=results,所有调试信息debug=query |
| explainOther | Allows clients to specify a Lucene query to identify a set of documents. If non-blank,the explain info of each document which matches this query, relative to the main query (specified by the q parameter) will be returned along with the rest of the debugging information. |
| timeAllowed | 指定查询处理的时间，单位毫秒。如果查询在指定的时间未完成，则只返回部分信息 |
| segmentTerminateEarly | Indicates that, if possible, Solr should stop collecting documents from each individual (sorted) segment once it can determine that any subsequent documents in that segment will not be candidates for the rows being returned. The default is false. |
| omitHeader | 当设为true时，返回结果不包含头部信息(例如请求花费的时间等信息)，默认是false |
| wt | 执行响应的输出格式：xml或json等 |
| logParamsList | 指定哪些参数需要写入log, eg:logParamsList=q,fq |
| echoParams | 指定响应头部包含哪些参数，取值为none/all/explicit(默认值) |

#### 2. 标准查询参数

| **参数** | **描述** |
| --- | --- |
| q | 使用标准查询语法定义的查询语句，必填 |
| q.op | 查询表达式的默认操作符，取值AND或者OR |
| df | 默认查找的字段 |

#### 3. 标准查询操作符

| **参数** | **描述** |
| --- | --- |
| ： | 指定要查找的字段，比如：title:“The Right Way” AND text:go |
| ？ | 匹配单一字符，比如:te?t匹配test/text |
| \* | 匹配0或多个字符，比如：tes\*匹配test/testing/tester |
| ~ | 基于编辑距离的模糊查询，比如：roam~匹配roams/foam/foams/roam. roam~1(指定距离必须是1)匹配roams/foam，但不会匹配foams |
| ~n | 邻近查询，查找相隔一定距离的单词，比如：”jakarta apache”~10（相隔10个单词） |
| to | 范围查询，{}不包含边界，[]包含边界，比如：title:{Aida TO Carmen} |
| ^ | 加权因子，比如:jakarta^4 apache 查找结果中jakarta更相关 |
| ^= | 指定查询语句的score得分为常量，比如：(description:blue OR color:blue)^=1.0 text:shoes |
| AND(&&) | 运算符两边的查询词同时出现 比如：”jakarta apache” AND “Apache Lucene” |
| OR | 运算符两边的查询词至少一个出现, 默认运算符，比如 “jakarta apache” jakarta 等价于 “jakarta apache” OR jakarta |
| NOT(!) | 运算符后面的查询词不出现，比如”jakarta apache” NOT “Apache Lucene” |
| + | 运算符后面的查询词出现(known as the “required” operator),比如+jakarta lucene查询必须包含jakarta,而lucene可以出现可不出现 |
| - | 不能包含运算符后面的查询词 “jakarta apache” -“Apache Lucene” |
| [] | 包含范围边界 |
| {} | 不包含范围边界 |

#### 4. 转义字符

+ - && || ! ( ) { } [ ] ^ " ~ \* ? : /

这些字符在solr中具有特殊的含义，如果要使用这么字符本身含义，需要利用反斜杠进行转义，比如： \(1\+1\):2

#### 5. 子查询语句

(jakarta OR apache) AND website 查询jakarta或apache出现，并且website必须出现 title:(+return +"pink panther") 查询title中包含return和“pink panther”

#### 6. 查询语句中使用注释

solr的查询语句支持C语言格式的注释。

"jakarta apache" /\* this is a comment in the middle of a normal query string \*/ OR jakarta

#### 7. 时间和日期的表示

timestamp:[\* TO NOW]

createdate:[1976-03-06T23:59:59.999Z TO \*]

createdate:[1995-12-31T23:59:59.999Z TO 2007-03-06T00:00:00Z]

pubdate:[NOW-1YEAR/DAY TO NOW/DAY+1DAY]

createdate:[1976-03-06T23:59:59.999Z TO 1976-03-06T23:59:59.999Z+1YEAR]

createdate:[1976-03-06T23:59:59.999Z/YEAR TO 1976-03-06T23:59:59.999Z]

#### 8. solr标准的查询解析与Lucene的查询解析的不同点如下：

1. \*号可以用在范围查询的开始或结束   
   field:[\* TO 100] field值小于等于100   
   field:[100 TO \*] field值大于等于100   
   field:[\* TO \*] 匹配包含field字段的所有文档
2. 支持单独出现的否定查询   
   -inStock:false field值inStack是false   
   -field:[\* TO \*] 匹配field无值的所有文档
3. 支持函数查询   
   val:myfield   
   val:”recip(rord(myfield),1,2,3)”
4. 支持多种类型的查询解析器   
   inStock:true OR {!dismax qf=’name manu’ v=’ipod’}
5. 支持过滤器缓存   
   inStock:true会被缓存并在以下三个查询中重用。   
   q=features:songs OR filter(inStock:true)   
   q=+manu:Apple +filter(inStock:true)   
   q=+manu:Apple & fq=inStock:true
6. 范围查询、前缀查询和通配符查询都可以指定文档的score是常量

#### 9. 查询语句中的局部参数

solr支持在基本查询语句中加入局部参数，

比如可以在 q=solr rocks 中加入参数以修改操作符和默认的查找字段：

q={!q.op=AND df=title}solr rocks

语法规定如下：

(1)以{!开始

(2)后跟任意数量的用空格分隔的key=value对

(3)以}结束，后跟查询语法

## 十一、solr之functionQuery(函数查询)

**怎样使用函数查询**

     这里主要有两种方法可以使用函数查询，这两种方法都是通过solr http 接口的。

1. 内嵌在正常的solr查询表达式中。即，将函数查询写在 q这个参数中，这时候，我们使用\_val\_将函数与其他的查询加以区别。至于具体怎样使用，请读者留意下面的例子。
2. 使用明确为函数查询的参数，比如说dismax中的bf（boost function）这个参数。  注意：bf这个参数是可以接受多个函数查询的，它们之间用空格隔开，它们还可以带上权重。所以，当我们使用bf这个参数的时候，我们必须保证单个函数中是没有空格出现的，不然程序有可能会以为是两个函数。

                例如：  q=dismax&bf="ord(popularity)^0.5 recip(rord(price),1,1000,1000)^0.3

**函数的格式（Function Query Syntax)**

        目前，function query 并不支持 a+b 这样的形式，我们得把它写成一个方法形式，这就是 sum(a,b).

**可以利用的函数 （available function）**

**constant**

                      支持有小数点的常量

                      例如：1.5 ；

                              SolrQuerySyntax:\_val\_:1.5

**fieldvalue**

                        这个函数将会返回numeric field的值，这个域必须是indexd的，非multiValued的。格式很简单，就是该域的名字。如果这个域中没有这样的值，那么将会返回0。

**ord**

                      对于一个域，它所有的值都将会按照字典顺序排列，这个函数返回你要查询的那个特定的值在这个顺序中的排名。这个域，必须是非multiValued的，当没有值存在的时候，将返回0.

               例如：某个特定的域只能去三个值，“apple”、“banana”、“pear”，那么ord（“apple”）=1，ord（“banana”）=2，ord（“pear”）=3.

                需要注意的是，ord（）这个函数，依赖于值在索引中的位置，所以当有文档被删除、或者添加的时候，ord（）的值就会发生变化。当你使用MultiSearcher的时候，这个值也就是不定的了。

**rord**

                      这个函数将会返回与ord相对应的倒排序的排名。

                      格式: rord(myIndexedField).

**sum**

                      这个函数的意思就显而易见啦，它就是表示“和”啦。

                      格式：sum(x,1)

                               sum(x,y)

                               sum(sqrt(x),log(y),z,0.5)

**product**

                        product(x,y,...)将会返回多个函数的乘积。

                     格式：product(x,2)

                              product(x,y)

**div**

                       div(x,y)表示x除以y的值

                     格式：div（1,x）

                              div(sum(x,100),max(y,1))

**pow**

                       pow表示幂值。pow(x,y) =x^y。

                       例如：pow(x,0.5) 表示开方

                               pow(x,log(y))

**abs**

                        abs(x)将返回表达式的绝对值

                       格式：abs(-5)

                       格式：abs(x)

**log**

                         log(x)将会返回基数为10，x的对数

                        格式： log(x)

                                   log(sum(x,100))

**sqrt**

                        sqrt(x) 返回 一个数的平方根

                        格式：sqrt（2）

                                 sqrt(sum(x,100))

**map**

                        如果 x>=min,且x<=max,那么map(x,min,max,target)=target.

                        如果 x不在[min,max]这个区间内，那么map(x,min,max,target)=x.

                       格式：map(x,0,0,1)

**scale**

                         scale(x,minTarget,maxTarget) 这个函数将会把x的值限制在[minTarget,maxTarget]范围内。

**query**

                        query(subquery,default)将会返回给定subquery的分数，如果subquery与文档不匹配，那么将会返回默认值。任何的查询类型都是受支持的。

                        可以通过引用的方式，也可以直接指定查询串。

                        例子：**q=product(popularity, query({!dismax v='solr rocks'})** 将会返回popularity和通过dismax 查询得到的分数的乘积。

**q=product(popularity, query($qq)&qq={!dismax}solr rocks** 跟上一个例子的效果是一样的。不过这里使用的是引用的方式

**q=product(popularity, query($qq,0.1)&qq={!dismax}solr rocks** 在前一个例子的基础上又加了一个默认值。

**linear**

                        linear(x,m,c)表示 m\*x+c ,其中m和c都是常量，x是一个变量也可以是一个函数。

                        例如： linear(x,2,4)=2\*x+4.

**recip**

                        recip(x,m,a,b)=a/(m\*x+b)其中，m、a、b是常量，x是变量或者一个函数。

                        当a=b，并且x>=0的时候，这个函数的最大值是1，值的大小随着x的增大而减小。

                       例如：recip(rord(creationDate),1,1000,1000)

**max**

                        max(x,c)将会返回一个函数和一个常量之间的最大值。

                       例如：max(myfield,0)

## 十二、solr分类搜索(Facet) 和 权重打分

### 一.Facet 简介

Facet 是 solr 的高级搜索功能之一 , 可以给用户提供更友好的搜索体验 . 在搜索关键字的同时 , 能够按照 Facet 的字段进行分组并统计 .

### 二.Face字段

1.       适宜被Facet 的字段

一般代表了实体的某种公共属性 , 如商品的分类 , 商品的制造厂家 , 书籍的出版商等等 .

2.       Facet 字段的要求

Facet 的字段必须被索引 . 一般来说该字段无需分词 , 无需存储 .

无需分词是因为该字段的值代表了一个整体概念 , 如电脑的品牌 ” 联想 ” 代表了一个整体概念 , 如果拆成 ” 联 ”,” 想 ” 两个字都不具有实际意义 . 另外该字段的值无需进行大小写转换等处理 , 保持其原貌即可 .

无需存储是因为一般而言用户所关心的并不是该字段的具体值 , 而是作为对查询结果进行分组的一种手段 , 用户一般会沿着这个分组进一步深入搜索 .

3.       特殊情况

对于一般查询而言 , 分词和存储都是必要的 . 比如 CPU 类型 ”Intel 酷睿 2 双核 P7570”,拆分成 ”Intel”,” 酷睿 ”,”P7570” 这样一些关键字并分别索引 , 可能提供更好的搜索体验 . 但是如果将 CPU 作为 Facet 字段 , 最好不进行分词 . 这样就造成了矛盾 , 解决方法为 ,将 CPU 字段设置为不分词不存储 , 然后建立另外一个字段为它的 COPY, 对这个 COPY 的字段进行分词和存储 .

schema.xml

|  |
| --- |
| <types>           <fieldType name="string" class="solr.StrField" omitNorms="true"/>           <fieldType name="tokened" class="solr.TextField" >                    <analyzer>                     ……                     </analyzer>           </fieldType>           ……  </types>  <fields>           <field name=”cpu” type=”string” indexed=”true” stored=”false”/>           <field name=”cpuCopy” type=” tokened” indexed=”true” stored=”true”/>           ……  </fields>  <copyField source="cpu" dest="cpuCopy"/> |

### 三.Facet组件

Solr 的默认 requestHandler(org.apache.solr.handler.component.SearchHandler) 已经包含了 Facet 组件 (org.apache.solr.handler.component.FacetComponent). 如果自定义 requestHandler 或者对默认的 requestHandler 自定义组件列表 , 那么需要将 Facet 加入到组件列表中去 .

solrconfig.xml

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <requestHandler name="standard" class="solr.SearchHandler" default="true">  ……  <arr name="components">  <str>自定义组件名</str>  <str>facet</str>  ……  </arr>  </requestHandler> 四.Facet查询 进行 Facet 查询需要在请求参数中加入 ”facet=on” 或者 ”facet=true” 只有这样 Facet 组件才起作用 .  1.       Field Facet  Facet 字段通过在请求中加入 ”facet.field” 参数加以声明 , 如果需要对多个字段进行 Facet查询 , 那么将该参数声明多次 . 比如   |  | | --- | | /select?q=联想  &facet=on  &facet.field=cpu  &facet.field=videoCard |   返回结果 :   |  | | --- | | <lst name="facet\_counts">           <lst name="facet\_queries"/>           <lst name="facet\_fields">                     <lst name="cpu">                              <int name="Intel 酷睿2双核 T6600">48</int>                              <int name="Intel 奔腾双核 T4300">28</int>  <int name="Intel 酷睿2双核 P8700">18</int>  <int name="Intel 酷睿2双核 T6570">11</int>  <int name="Intel 酷睿2双核 T6670">11</int>  <int name="Intel 奔腾双核 T4400">9</int>  <int name="Intel 酷睿2双核 P7450">9</int>  <int name="Intel 酷睿2双核 T5870">8</int>  <int name="Intel 赛扬双核 T3000">7</int>  <int name="Intel 奔腾双核 SU4100">6</int>  <int name="Intel 酷睿2双核 P8400">6</int>  <int name="Intel 酷睿2双核 SU7300">5</int>  <int name="Intel 酷睿 i3 330M">4</int>                     </lst>                     <lst name="videoCard">                              <int name="ATI Mobility Radeon HD 4">63</int>                              <int name="NVIDIA GeForce G 105M">24</int>  <int name="NVIDIA GeForce GT 240M">21</int>  <int name="NVIDIA GeForce G 103M">8</int>  <int name="NVIDIA GeForce GT 220M">8</int>  <int name="NVIDIA GeForce 9400M G">7</int>  <int name="NVIDIA GeForce G 210M">6</int>  </lst>           </lst>           <lst name="facet\_dates"/>  </lst> |   各个 Facet 字段互不影响 , 且可以针对每个 Facet 字段设置查询参数 . 以下介绍的参数既可以应用于所有的 Facet 字段 , 也可以应用于每个单独的 Facet 字段 . 应用于单独的字段时通过   |  | | --- | | f.字段名.参数名=参数值 |   这种方式调用 . 比如 facet.prefix 参数应用于 cpu 字段 , 可以采用如下形式   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | f.cpu.facet.prefix=Intel  1.1   facet.prefix  表示 Facet 字段值的前缀 . 比如 ”facet.field=cpu&facet.prefix=Intel”, 那么对 cpu字段进行 Facet 查询 , 返回的 cpu 都是以 ”Intel” 开头的 ,”AMD” 开头的 cpu 型号将不会被统计在内 .  1.2   facet.sort  表示 Facet 字段值以哪种顺序返回 . 可接受的值为 true(count)|false(index,lex). true(count) 表示按照 count 值从大到小排列 . false(index,lex) 表示按照字段值的自然顺序 (字母 , 数字的顺序 ) 排列 . 默认情况下为 true(count). 当 facet.limit 值为负数时 ,默认 facet.sort= false(index,lex).  1.3   facet.limit  限制 Facet 字段返回的结果条数 . 默认值为 100. 如果此值为负数 , 表示不限制 .  1.4   facet.offset  返回结果集的偏移量 , 默认为 0. 它与 facet.limit 配合使用可以达到分页的效果 .  1.5   facet.mincount  限制了 Facet 字段值的最小 count, 默认为 0. 合理设置该参数可以将用户的关注点集中在少数比较热门的领域 .  1.6   facet.missing  默认为 ””, 如果设置为 true 或者 on, 那么将统计那些该 Facet 字段值为 null 的记录.  1.7   facet.method  取值为 enum 或 fc, 默认为 fc. 该字段表示了两种 Facet 的[**算法**](http://lib.csdn.net/base/datastructure) , 与执行效率相关 .  enum 适用于字段值比较少的情况 , 比如字段类型为布尔型 , 或者字段表示中国的所有省份.Solr 会遍历该字段的所有取值 , 并从 filterCache 里为每个值分配一个 filter( 这里要求 solrconfig.xml 里对 filterCache 的设置足够大 ). 然后计算每个 filter 与主查询的交集 .  fc( 表示 Field Cache) 适用于字段取值比较多 , 但在每个文档里出现次数比较少的情况 .Solr 会遍历所有的文档 , 在每个文档内搜索 Cache 内的值 , 如果找到就将 Cache 内该值的count 加 1.  1.8   facet.enum.cache.minDf  当 facet.method=enum 时 , 此参数其作用 ,minDf 表示 minimum document frequency. 也就是文档内出现某个关键字的最少次数 . 该参数默认值为 0. 设置该参数可以减少 filterCache 的内存消耗 , 但会增加总的查询时间 ( 计算交集的时间增加了 ). 如果设置该值的话 ,官方文档建议优先尝试 25-50 内的值 .  2.       Date Facet  日期类型的字段在文档中很常见 , 如商品上市时间 , 货物出仓时间 , 书籍上架时间等等 . 某些情况下需要针对这些字段进行 Facet. 不过时间字段的取值有无限性 , 用户往往关心的不是某个时间点而是某个时间段内的查询统计结果 . Solr 为日期字段提供了更为方便的查询统计方式 .当然 , 字段的类型必须是 DateField( 或其子类型 ).  需要注意的是 , 使用 Date Facet 时 , 字段名 , 起始时间 , 结束时间 , 时间间隔这 4 个参数都必须提供 .  与 Field Facet 类似 ,Date Facet 也可以对多个字段进行 Facet. 并且针对每个字段都可以单独设置参数 .  2.1   facet.date  该参数表示需要进行 Date Facet 的字段名 , 与 facet.field 一样 , 该参数可以被设置多次 , 表示对多个字段进行 Date Facet.  2.2   facet.date.start  起始时间 , 时间的一般格式为 ” 1995-12-31T23:59:59Z”, 另外可以使用 ”NOW”,”YEAR”,”MONTH” 等等 , 具体格式可以参考 org.apache.solr.schema. DateField 的 [**Java**](http://lib.csdn.net/base/java)doc.  2.3   facet.date.end  结束时间 .  2.4   facet.date.gap  时间间隔 . 如果 start 为 2009-1-1,end 为 2010-1-1.gap 设置为 ”+1MONTH” 表示间隔1 个月 , 那么将会把这段时间划分为 12 个间隔段 . 注意 ”+” 因为是特殊字符所以应该用 ”%2B” 代替 .  2.5   facet.date.hardend  取值可以为 true|false, 默认为 false. 它表示 gap 迭代到 end 处采用何种处理 . 举例说明 start 为 2009-1-1,end 为 2009-12-25,gap 为 ”+1MONTH”,hardend 为 false 的话最后一个时间段为 2009-12-1 至 2010-1-1;hardend 为 true 的话最后一个时间段为 2009-12-1 至 2009-12-25.  2.6   facet.date.other  取值范围为 before|after|between|none|all, 默认为 none.  before 会对 start 之前的值做统计 .  after 会对 end 之后的值做统计 .  between 会对 start 至 end 之间所有值做统计 . 如果 hardend 为 true 的话 , 那么该值就是各个时间段统计值的和 .  none 表示该项禁用 .  all 表示 before,after,all 都会统计 .  举例 :   |  | | --- | | &facet=on  &facet.date=date  &facet.date.start=2009-1-1T0:0:0Z  &facet.date.end=2010-1-1T0:0:0Z  &facet.date.gap=%2B1MONTH  &facet.date.other=all |   返回结果 :   |  | | --- | | <lst name="facet\_counts">           <lst name="facet\_queries"/>           <lst name="facet\_fields"/>           <lst name="facet\_dates">                     <int name="2009-01-01T00:00:00Z">5</int>  <int name="2009-02-01T00:00:00Z">7</int>  <int name="2009-03-01T00:00:00Z">4</int>  <int name="2009-04-01T00:00:00Z">3</int>  <int name="2009-05-01T00:00:00Z">7</int>  <int name="2009-06-01T00:00:00Z">3</int>  <int name="2009-07-01T00:00:00Z">6</int>  <int name="2009-08-01T00:00:00Z">7</int>  <int name="2009-09-01T00:00:00Z">2</int>  <int name="2009-10-01T00:00:00Z">4</int>  <int name="2009-11-01T00:00:00Z">1</int>  <int name="2009-12-01T00:00:00Z">5</int>  <str name="gap">+1MONTH</str>  <date name="end">2010-01-01T00:00:00Z</date>  <int name="before">180</int>  <int name="after">5</int>  <int name="between">54</int>  </lst>  </lst> |   3.       Facet Query  Facet Query 利用类似于 filter query 的语法提供了更为灵活的 Facet. 通过 facet.query 参数 , 可以对任意字段进行筛选 .  例 1:   |  | | --- | | &facet=on  &facet.query=date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]  &facet.query=date:[2009-4-1T0:0:0Z TO 2009-5-1T0:0:0Z] |   返回结果 :   |  | | --- | | <lst name="facet\_counts">           <lst name="facet\_queries">                     <int name="date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]">5</int>  <int name="date:[2009-4-1T0:0:0Z TO 2009-5-1T0:0:0Z]">3</int>  </lst>           <lst name="facet\_fields"/>           <lst name="facet\_dates"/>  </lst> |   例 2:   |  | | --- | | &facet=on  &facet.query=date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]  &facet.query=price:[\* TO 5000] |   返回结果 :   |  | | --- | | <lst name="facet\_counts">           <lst name="facet\_queries">                     <int name="date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]">5</int>  <int name="price:[\* TO 5000]">116</int>  </lst>           <lst name="facet\_fields"/>           <lst name="facet\_dates"/>  </lst> |   例 3:   |  | | --- | | &facet=on  &facet.query=cpu:[A TO G] |   返回结果 :   |  | | --- | | <lst name="facet\_counts">           <lst name="facet\_queries">                     <int name="cpu:[A TO G]">11</int>  </lst>           <lst name="facet\_fields"/>           <lst name="facet\_dates"/>  </lst> |   4.       key 操作符  可以用 key 操作符为 Facet 字段取一个别名 .  例 :   |  | | --- | | &facet=on  &facet.field={!key=中央处理器}cpu  &facet.field={!key=显卡}videoCard |   返回结果 :   |  | | --- | | <lst name="facet\_counts">           <lst name="facet\_queries"/>           <lst name="facet\_fields">                     <lst name="中央处理器">                              <int name="Intel 酷睿2双核 T6600">48</int>                              <int name="Intel 奔腾双核 T4300">28</int>  <int name="Intel 酷睿2双核 P8700">18</int>  <int name="Intel 酷睿2双核 T6570">11</int>  <int name="Intel 酷睿2双核 T6670">11</int>  <int name="Intel 奔腾双核 T4400">9</int>  <int name="Intel 酷睿2双核 P7450">9</int>  <int name="Intel 酷睿2双核 T5870">8</int>  <int name="Intel 赛扬双核 T3000">7</int>  <int name="Intel 奔腾双核 SU4100">6</int>  <int name="Intel 酷睿2双核 P8400">6</int>  <int name="Intel 酷睿2双核 SU7300">5</int>  <int name="Intel 酷睿 i3 330M">4</int>                     </lst>                     <lst name="显卡">                              <int name="ATI Mobility Radeon HD 4">63</int>                              <int name="NVIDIA GeForce G 105M">24</int>  <int name="NVIDIA GeForce GT 240M">21</int>  <int name="NVIDIA GeForce G 103M">8</int>  <int name="NVIDIA GeForce GT 220M">8</int>  <int name="NVIDIA GeForce 9400M G">7</int>  <int name="NVIDIA GeForce G 210M">6</int>  </lst>           </lst>           <lst name="facet\_dates"/>  </lst> |   5.       tag 操作符和 ex 操作符  当查询使用 filter query 的时候 , 如果 filter query 的字段正好是 Facet 字段 , 那么查询结果往往被限制在某一个值内 .  例 :   |  | | --- | | &fq=screenSize:14  &facet=on  &facet.field=screenSize |   返回结果 :   |  | | --- | | <lst name="facet\_counts">           <lst name="facet\_queries"/>           <lst name="facet\_fields">                     <lst name=" screenSize">                              <int name="14.0">107</int>  <int name="10.2">0</int>  <int name="11.1">0</int>  <int name="11.6">0</int>  <int name="12.1">0</int>  <int name="13.1">0</int>  <int name="13.3">0</int>  <int name="14.1">0</int>  <int name="15.4">0</int>  <int name="15.5">0</int>  <int name="15.6">0</int>  <int name="16.0">0</int>  <int name="17.0">0</int>  <int name="17.3">0</int>  </lst>           </lst>           <lst name="facet\_dates"/>  </lst> |   可以看到 , 屏幕尺寸 (screenSize) 为 14 寸的产品共有 107 件 , 其它尺寸的产品的数目都是0, 这是因为在 filter 里已经限制了 screenSize:14. 这样 , 查询结果中 , 除了 screenSize=14 的这一项之外 , 其它项目没有实际的意义 .  有些时候 , 用户希望把结果限制在某一范围内 , 又希望查看该范围外的概况 . 比如上述情况 ,既要把查询结果限制在 14 寸屏的笔记本 , 又想查看一下其它屏幕尺寸的笔记本有多少产品 . 这个时候需要用到 tag 和 ex 操作符 .  tag 就是把一个 filter 标记起来 ,ex(exclude) 是在 Facet 的时候把标记过的 filter 排除在外 .  例 :   |  | | --- | | &fq={!tag=aa}screenSize:14  &facet=on  &facet.field={!ex=aa}screenSize |   返回结果 :   |  | | --- | | <lst name="facet\_counts">           <lst name="facet\_queries"/>           <lst name="facet\_fields">                     <lst name=" screenSize">                              <int name="14.0">107</int>  <int name="14.1">40</int>  <int name="13.3">34</int>  <int name="15.6">22</int>  <int name="15.4">8</int>  <int name="11.6">6</int>  <int name="12.1">5</int>  <int name="16.0">5</int>  <int name="15.5">3</int>  <int name="17.0">3</int>  <int name="17.3">3</int>  <int name="10.2">1</int>  <int name="11.1">1</int>  <int name="13.1">1</int>  </lst>           </lst>           <lst name="facet\_dates"/>  </lst> |   这样其它屏幕尺寸的统计信息就有意义了 . 五.SolrJ 对 Facet 的支持  |  | | --- | | SolrServer server = getSolrServer();//获取SolrServer  SolrQuery query = **new** SolrQuery();//建立一个新的查询  query.setQuery("\*:\*");  query.setFacet(**true**);//设置facet=on  query.addFacetField(**new** String[] { "cpu", "videoCard" });//设置需要facet的字段  query.setFacetLimit(10);//限制facet返回的数量  QueryResponse response = server.query(query);  List<FacetField> facets = response.getFacetFields();//返回的facet列表  **for** (FacetField facet : facets) {  System.out.println(facet.getName());      System.out.println("----------------");      List<Count> counts = facet.getValues();  **for** (Count count : counts) {          System.out.println(count.getName() + ":" + count.getCount());      }      System.out.println();  } 六、查询权重和分类排序 Solr的defType有dismax/edismax两种，这两种的区别，可参见：<http://blog.csdn.net/duck_genuine/article/details/8060026>    **下面示例用于演示如下场景：**  有一网站，在用户查询的结果中，需要按这样排序：   1. VIP的付费信息需要排在免费信息的前头 2. 点击率越高越靠前 3. 发布时间越晚的越靠前     这样的查询排序使用普通的查询结果的Order by是做不到的，必需使用solr的defType。  **做法：**  1、先看schema.xml的定义：  复制代码  <?xml version="1.0" ?>  <schema name="sample5" version="1.1">  <fieldtype name="string" class="solr.StrField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="long" class="solr.TrieLongField" precisionStep="0" positionIncrementGap="0"/>  <fieldType name="tdate" class="solr.TrieDateField" precisionStep="6" positionIncrementGap="0"/>  <fieldType name="int" class="solr.TrieIntField" precisionStep="0" positionIncrementGap="0"/>  <fieldType name="float" class="solr.TrieFloatField" precisionStep="0" positionIncrementGap="0"/>  <fieldType name="double" class="solr.TrieDoubleField" precisionStep="0" positionIncrementGap="0"/>  <fieldType name="boolean" class="solr.BoolField" sortMissingLast="true"/>  <fieldtype name="binary" class="solr.BinaryField"/>  <fieldType name="text\_cn" class="solr.TextField">  <analyzer type="index" class="org.wltea.analyzer.lucene.IKAnalyzer" useSmart="false" />  <analyzer type="query" class="org.wltea.analyzer.lucene.IKAnalyzer" useSmart="true" />  <analyzer>  <tokenizer class="solr.KeywordTokenizerFactory"/>  <filter class="solr.LowerCaseFilterFactory" ignoreCase="true"/>  </analyzer>  </fieldType>    <!-- general -->  <fields>  <field name="id" type="long" indexed="true" stored="true" multiValued="false" required="true"/>  <field name="subject" type="text\_cn" indexed="true" stored="true" />  <field name="content" type="text\_cn" indexed="true" stored="true" />  <field name="regionId" type="int" indexed="true" stored="true" />  <field name="region" type="text\_cn" indexed="true" stored="true" />  <field name="categoryId" type="int" indexed="true" stored="true" />  <field name="category" type="text\_cn" indexed="true" stored="true" />  <field name="price" type="float" indexed="true" stored="true" />  <field name="createTime" type="tdate" indexed="true" stored="true" />  <field name="point" type="long" indexed="true" stored="true" />  <field name="vip" type="boolean" indexed="true" stored="true" />  <field name="\_version\_" type="long" indexed="true" stored="true"/>  <field name="searchText" type="text\_cn" indexed="true" stored="false" multiValued="true" />  </fields>    <copyField source="subject" dest="searchText" />  <copyField source="content" dest="searchText" />  <copyField source="region" dest="searchText" />  <copyField source="category" dest="searchText" />    <!-- field to use to determine and enforce document uniqueness. -->  <uniqueKey>id</uniqueKey>  <!-- field for the QueryParser to use when an explicit fieldname is absent -->  <defaultSearchField>searchText</defaultSearchField>  <!-- SolrQueryParser configuration: defaultOperator="AND|OR" -->  <solrQueryParser defaultOperator="AND"/>  </schema>  复制代码  说明：  a)里头定义了一个copyField：searchText，此字段为：subject+content+region+category，并把这个字段设置为默认查询字段。意思是查询时，默认查询四个字段的内容。  b)把solrQueryParser设置为AND，事实上，大多情况下，我们是习惯使用AND为条件查询，而非OR  c)text\_cn字段类型中的：useSmart  <analyzer type="index" class="org.wltea.analyzer.lucene.IKAnalyzer" useSmart="false" />  <analyzer type="query" class="org.wltea.analyzer.lucene.IKAnalyzer" useSmart="true" />  意思是：useSmart =true ，分词器使用智能切分策略， =false则使用细粒度切分。详细，可下载IK分词器的源码看看。    2、加入一个查询Handler到solrconfig.xml的<config/>当中：  复制代码  <requestHandler name="/browse" class="solr.SearchHandler" default="true" >  <lst name="defaults">  <str name="defType">edismax</str>  <str name="bf">  sum(linear(vip,1000,0),linear(sqrt(log(linear(point,1,2))),100,0),sqrt(log(ms(createTime))))  </str>  <!--<str name="pf">  searchText  </str>  <str name="qf">  subject^1 content^0.8  </str>-->  </lst>  </requestHandler>  复制代码  说明：  a)上面的default="true"意思为设置为默认的查询handler（记得把原standard中的default="true"删除掉）  b)见已经被注释的这段：  <!--<str name="pf">  searchText  </str>  <str name="qf">  subject^1 content^0.8  </str>-->  这是简单的不使用bf的排序加权方式，可以用于应付简单的排序，具体pf/qf的使用，可以上网上搜搜应用。这里演示的功能相对“复杂”，不适用它。  c)见这句公式：  sum(linear(vip,1000,0),linear(sqrt(log(linear(point,1,2))),100,0),sqrt(log(ms(createTime))))  公式中的函数定义和意思，可以参考：  官方文档：  <http://wiki.apache.org/solr/FunctionQuery>  中文说明：  <http://mxsfengg.iteye.com/blog/352191>  这里的函数意思是：   * 如果是vip信息=值+1000，非vip信息=值+0 * 点击率(point)的值范围为：50~500之间 * 发布时间(createTime)值范围为：50以内   以上三个值相加得出最统权重分从高到低排序    3、Java bean:  复制代码  package com.my.entity;  import java.util.Date;  import org.apache.solr.client.solrj.beans.Field;  public class Item {  @Field  private long id;  @Field  private String subject;  @Field  private String content;  @Field  private int regionId;  @Field  private int categoryId;  @Field  private float price;  @Field  private Date createTime;  @Field  private long point;  @Field  private boolean vip;    public long getId() {  return id;  }  public void setId(long id) {  this.id = id;  }  public String getSubject() {  return subject;  }  public void setSubject(String subject) {  this.subject = subject;  }  public String getContent() {  return content;  }  public void setContent(String content) {  this.content = content;  }  public int getRegionId() {  return regionId;  }  public void setRegionId(int regionId) {  this.regionId = regionId;  }  public int getCategoryId() {  return categoryId;  }  public void setCategoryId(int categoryId) {  this.categoryId = categoryId;  }  public float getPrice() {  return price;  }  public void setPrice(float price) {  this.price = price;  }  public Date getCreateTime() {  return createTime;  }  public void setCreateTime(Date createTime) {  this.createTime = createTime;  }  public long getPoint() {  return point;  }  public void setPoint(long point) {  this.point = point;  }  public boolean isVip() {  return vip;  }  public void setVip(boolean vip) {  this.vip = vip;  }  }  复制代码    4、Java测试代码：  复制代码  package com.my.solr;  import java.io.IOException;  import java.util.ArrayList;  import java.util.Calendar;  import java.util.Date;  import java.util.HashMap;  import java.util.Iterator;  import java.util.List;  import org.apache.solr.client.solrj.SolrQuery;  import org.apache.solr.client.solrj.SolrQuery.ORDER;  import org.apache.solr.client.solrj.SolrQuery.SortClause;  import org.apache.solr.client.solrj.SolrServerException;  import org.apache.solr.client.solrj.impl.HttpSolrServer;  import org.apache.solr.client.solrj.impl.XMLResponseParser;  import org.apache.solr.client.solrj.response.FacetField;  import org.apache.solr.client.solrj.response.FacetField.Count;  import org.apache.solr.client.solrj.response.QueryResponse;  import org.apache.solr.common.params.AnalysisParams;  import org.apache.solr.common.params.CommonParams;  import org.apache.solr.common.params.FacetParams;  import org.apache.solr.common.util.NamedList;  import org.apache.solr.common.util.SimpleOrderedMap;  import com.my.entity.Item;  public class TestSolr {  private static HashMap<Integer, String> mapRegion = new HashMap<Integer, String>();  private static HashMap<Integer, String> mapCategory = new HashMap<Integer, String>();  @SuppressWarnings("unchecked")  public static void main(String[] args) throws IOException,  SolrServerException {  // ------------------------------------------------------  // Set map  // ------------------------------------------------------  mapRegion.put(1, "罗湖区");  mapRegion.put(2, "南山区");  mapRegion.put(3, "龙岗区");  mapRegion.put(4, "福田区");  mapCategory.put(1, "单间");  mapCategory.put(2, "2房1厅");  mapCategory.put(3, "3房2厅");  mapCategory.put(4, "1房1厅");    String url = "http://localhost:8983/solr/sample5";  HttpSolrServer core = new HttpSolrServer(url);  core.setMaxRetries(1);  core.setConnectionTimeout(5000);  core.setParser(new XMLResponseParser()); // binary parser is used by  // default  core.setSoTimeout(1000); // socket read timeout  core.setDefaultMaxConnectionsPerHost(100);  core.setMaxTotalConnections(100);  core.setFollowRedirects(false); // defaults to false  core.setAllowCompression(true);  // ------------------------------------------------------  // remove all data  // ------------------------------------------------------  core.deleteByQuery("\*:\*");  List<Item> items = new ArrayList<Item>();  items.add(makeItem(items.size() + 1, "龙城公寓一房一厅", "豪华城城公寓1房1厅，拧包入住", 1, 1, 1200f, 10, false));  items.add(makeItem(items.size() + 1, "兴新宿舍楼 1室0厅", " 中等装修 招女性合租", 1, 1, 1000f, 11, false));  items.add(makeItem(items.size() + 1, "西丽新屋村新宿舍楼单间", " 无敌装修只招女性", 2, 1, 1000f, 2, true));  items.add(makeItem(items.size() + 1, "大芬村信和爱琴居地铁口2房1厅", " 地铁口 + 出行便利=居家首选", 3, 2, 2000f, 5, false));  items.add(makeItem(items.size() + 1, "龙岗富豪花园3房2厅出租", " 离地铁口只要5分钟，快来秒杀吧", 3, 3, 4500f, 21, true));  items.add(makeItem(items.size() + 1, "海景房园3房2厅出租", "海景房园出租，无敌海景，可以看到伦敦", 4, 3, 8500f, 12, false));  items.add(makeItem(items.size() + 1, "天域花园1房1厅", "天域花园，男女不限，入住免水电一月", 2, 4, 1500f, 13, true));  items.add(makeItem(items.size() + 1, "神一样的漂亮，玉馨山庄3房2厅", "心动不如行动，拧包即可入住，来吧！", 1, 3, 9500f, 8, false));  items.add(makeItem(items.size() + 1, "玉馨山庄2房1厅，情侣最爱", "宅男宅女快来吧只要2500，走过路过，别再错过", 1, 2, 2500f, 5, false));  items.add(makeItem(items.size() + 1, "天域花园3房2厅", "天域花园出租，都来看看，都来瞄瞄，3房出租只要7500.", 4, 3, 7500f, 6, true));  items.add(makeItem(items.size() + 1, "深都花园出租3房2厅", "找爱干净的人氏，全新装修", 4, 3, 5200f, 31, false));  items.add(makeItem(items.size() + 1, "This is Mobile test", "haha Hello world!", 4, 3, 1200f, 31, false));  core.addBeans(items);  // commit  core.commit();    // ------------------------------------------------------  // Set search text  // ------------------------------------------------------  String searchText = AnalysisSearchText(core, "出租花园"); //subject:\*出租\* && price:[1000 TO 8000]  System.out.println("Search Text:" + searchText);    // ------------------------------------------------------  // Set query text  // ------------------------------------------------------  String queryText = searchText + "&& price:[1000 TO 8000]";  System.out.println("Query Text:" + queryText);  // ------------------------------------------------------  // search  // ------------------------------------------------------  SolrQuery query = new SolrQuery();  query.setQuery(queryText);  query.setStart(0); // query的开始行数(分页使用)  query.setRows(100); // query的返回行数(分页使用)  query.setFacet(true); // 设置使用facet  query.setFacetMinCount(0); // 设置facet最少的统计数量  query.setFacetLimit(10); // facet结果的返回行数  query.addFacetField("categoryId", "regionId"); // facet的字段  query.setFacetSort(FacetParams.FACET\_SORT\_COUNT);  //query.addSort(new SortClause("id", ORDER.asc)); // 排序  query.setRequestHandler("/browse");  QueryResponse response = core.query(query);  List<Item> items\_rep = response.getBeans(Item.class);  List<FacetField> facetFields = response.getFacetFields();  // 因为上面的start和rows均设置为0，所以这里不会有query结果输出  System.out.println("--------------------");  System.out.println("Search result:");  for (Item item : items\_rep) {  System.out.println("id=" + item.getId() + "\tsubject=" + item.getSubject()  + "\tregion=" + mapRegion.get(item.getRegionId())  + "\tcategory=" + mapCategory.get(item.getCategoryId())  + "\tprice=" + item.getPrice());  }  // 打印所有facet  for (FacetField ff : facetFields) {  System.out.println("--------------------");  System.out.println("name=" + ff.getName() + "\tcount=" + ff.getValueCount());  System.out.println("--------------------");  switch (ff.getName()) {  case "regionId":  printOut(mapRegion, ff.getValues());  break;  case "categoryId":  printOut(mapCategory, ff.getValues());  break;  }  }  }    @SuppressWarnings({ "rawtypes" })  private static void printOut(HashMap map, List<Count> counts) {  for (Count count : counts) {  System.out.println("name=" + map.get(Integer.parseInt(count.getName())) + "\tcount=" + count.getCount());  }  System.out.println("--------------------");  }  private static Item makeItem(long id, String subject, String content, int regionId, int categoryId, float price,  long point, boolean vip) {  Calendar cale = Calendar.getInstance();  cale.setTime(new Date());  cale.add(Calendar.DATE, (int)id);  Item item = new Item();  item.setId(id);  item.setSubject(subject);  item.setContent(content);  item.setRegionId(regionId);  item.setCategoryId(categoryId);  item.setPrice(price);  item.setCreateTime(cale.getTime());  item.setPoint(point);  item.setVip(vip);  return item;  }    @SuppressWarnings("unchecked")  /\*\*  \* 重新将需要查询的文本内容解析成分词  \* @param core  \* @param searchText  \* @return  \* @throws SolrServerException  \*/  private static String AnalysisSearchText(HttpSolrServer core, String searchText) throws SolrServerException {  StringBuilder strSearchText = new StringBuilder();  final String STR\_FIELD\_TYPE = "text\_cn";  SolrQuery queryAnalysis = new SolrQuery();  queryAnalysis.add(CommonParams.QT, "/analysis/field"); // query type  queryAnalysis.add(AnalysisParams.FIELD\_VALUE, searchText);  queryAnalysis.add(AnalysisParams.FIELD\_TYPE, STR\_FIELD\_TYPE);  QueryResponse responseAnalysis = core.query(queryAnalysis);  //对响应进行解析  NamedList<Object> analysis = (NamedList<Object>) responseAnalysis.getResponse().get("analysis");// analysis node  NamedList<Object> field\_types = (NamedList<Object>) analysis.get("field\_types");// field\_types node  NamedList<Object> fieldType = (NamedList<Object>) field\_types.get(STR\_FIELD\_TYPE);// text\_cn node  NamedList<Object> index = (NamedList<Object>) fieldType.get("index");// index node  List<SimpleOrderedMap<String>> list = (ArrayList<SimpleOrderedMap<String>>)index.get("org.wltea.analyzer.lucene.IKTokenizer");// tokenizer node  // 在每个词条中间加上空格，为每个词条进行或运算  for(Iterator<SimpleOrderedMap<String>> iter = list.iterator(); iter.hasNext();)  {  strSearchText.append(iter.next().get("text") + " ");  }  return strSearchText.toString();  }  }  复制代码  说明：  a)AnalysisSearchText(...)方法：此方法会把需要查询的语句先使用分词分析，如上例子“出租花园”，调用AnalysisSearchText(...)后，会得到“出租 花园”，会把两个词分拆成以空格分隔的字符串。不然solr会以“出租花园”整体做为词做查询而得不到结果。  b)使用自定义的Handler，需要在代码中加入这句：  query.setRequestHandler("/browse");  对应的是solrconfig.xml中的requestHandler的:/browse http://images.cnitblog.com/blog/6088/201410/141054254517237.png | | |