# 淘淘商城-搜索系统实现、Solr集群搭建

## 内容回顾

1. 向业务逻辑中添加缓存
   1. Redis做缓存工具
   2. 添加缓存不能影响正常的业务逻辑
   3. 查询流程：
      1. 从缓存中取内容，如果有直接返回
      2. 如果没有查询数据库
      3. 把查询结果添加到缓存。
   4. 缓存同步的问题：
      1. 发布服务
      2. 后台系统中修改内容信息后，调用服务，清空缓存。
2. 搜索系统的搭建。
3. 使用solr实现搜索。
   1. Linux系统下安装solr
   2. 安装中文分析器及添加业务域。
   3. 把数据库数据导入到索引库中。

## 课程计划

1. 搜索系统的实现
   1. 发布搜索的服务
   2. Portal系统中调用搜索服务实现搜索功能。
2. Solr集群搭建

## 搜索系统的实现

### 搜索服务发布

调用服务传递过来一个查询条件，根据查询条件进行查询。返回查询结果。参数中包括分页条件。

参数：

String queryString

Int page

Int rows

返回结果：返回json数据。

包含查询结果的列表。使用商品的pojo来描述。SearchItem

包含查询结果总记录数。

包含查询结果的总页数。

包含当前页码。

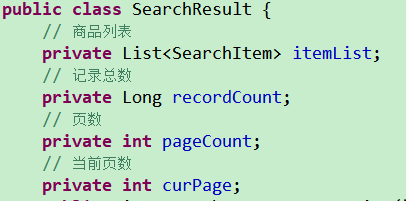
包含查询的状态。

包含错误信息。

创建一个SearchResult

包含四个属性：

1. 商品列表
2. 查询结果总记录数
3. 查询结果的总页数
4. 当前页码



|  |
| --- |
| **public** **class** SearchResult {  **private** List<SearchItem> itemList;  **private** Long recordCount;  **private** **int** pageCount;  **private** **int** curPage;  } |

使用TaotaoResult包装一个SearchResult返回结果

#### Dao层

根据查询条件进行查询，返回查询结果。

参数：SolrQuery对象

返回结果：

1. 查询结果的商品列表
2. 查询结果的总记录数

返回SearchResult

在taotao-search工程下创建com.taotao.search.dao、com.taotao.search.dao.impl层

|  |
| --- |
| **public** **interface** SearchDao {  **public** SearchResult search(SolrQuery query) **throws** Exception;  }  SearchDaoImpl：  @Repository  **public** **class** SearchDaoImpl **implements** SearchDao {  @Autowired  **private** SolrServer solrServer;  @Override  **public** SearchResult search(SolrQuery query) **throws** Exception {  //执行查询  QueryResponse response = solrServer.query(query);  //取查询结果列表  SolrDocumentList solrDocumentList = response.getResults();  List<SearchItem> itemList = **new** ArrayList<>();  **for** (SolrDocument solrDocument : solrDocumentList) {  //创建一个SearchItem对象  SearchItem item = **new** SearchItem();  item.setCategory\_name((String) solrDocument.get("item\_category\_name"));  item.setId((String) solrDocument.get("id"));  item.setImage((String) solrDocument.get("item\_image"));  item.setPrice((Long) solrDocument.get("item\_price"));  item.setSell\_point((String) solrDocument.get("item\_sell\_point"));  //取高亮显示  Map<String, Map<String, List<String>>> highlighting = response.getHighlighting();  List<String> list = highlighting.get(solrDocument.get("id")).get("item\_title");  String itemTitle = "";  **if** (list != **null** && list.size() > 0) {  //取高亮后的结果  itemTitle = list.get(0);  } **else** {  itemTitle = (String) solrDocument.get("item\_title");  }  item.setTitle(itemTitle);  //添加到列表  itemList.add(item);  }  SearchResult result = **new** SearchResult();  result.setItemList(itemList);  //查询结果总数量  result.setRecordCount(solrDocumentList.getNumFound());  **return** result;  }  } |

配置对应的包扫描器，扫描相关的包：



#### Service层

1. 接收查询条件、分页条件。
2. 创建SolrQuery对象，设置查询条件、分页条件。
3. 调用dao进行搜索
4. 计算总页数，把总页数设置到SearchResult对象中，设置当前页属性。
5. 返回SearchResult

参数：

1. 查询条件
2. Page
3. Rows

返回结果：

SearchResult

|  |
| --- |
| **public** **interface** SearchService {  **public** SearchResult search(String queryString, **int** page, **int** rows) **throws** Exception;    }  SearchServiceImpl：  @Service  **public** **class** SearchServiceImpl **implements** SearchService {  @Autowired  **private** SearchDao searchDao;  @Override  **public** SearchResult search(String queryString, **int** page, **int** rows) **throws** Exception {  //创建查询条件  SolrQuery query = **new** SolrQuery();  //设置查询条件  query.setQuery(queryString);  //设置分页条件  query.setStart((page-1)\*rows);  query.setRows(rows);  //设置默认搜索域  query.set("df", "item\_title");  //设置高亮  query.setHighlight(**true**);  query.addHighlightField("item\_title");  query.setHighlightSimplePre("<font class=\"skcolor\_ljg\">");  query.setHighlightSimplePost("</font>");  //执行查询  SearchResult searchResult = searchDao.search(query);  //计算总页数  Long recordCount = searchResult.getRecordCount();  **int** pageCount = (**int**) (recordCount / rows);  **if** (recordCount % rows > 0) {  pageCount++;  }  searchResult.setPageCount(pageCount);  searchResult.setCurPage(page);  **return** searchResult;  }  } |

#### Controller层

发布服务。

搜索服务的url：/search/q?keyword=xxx&page=1&rows=30

参数keyword、page、rows

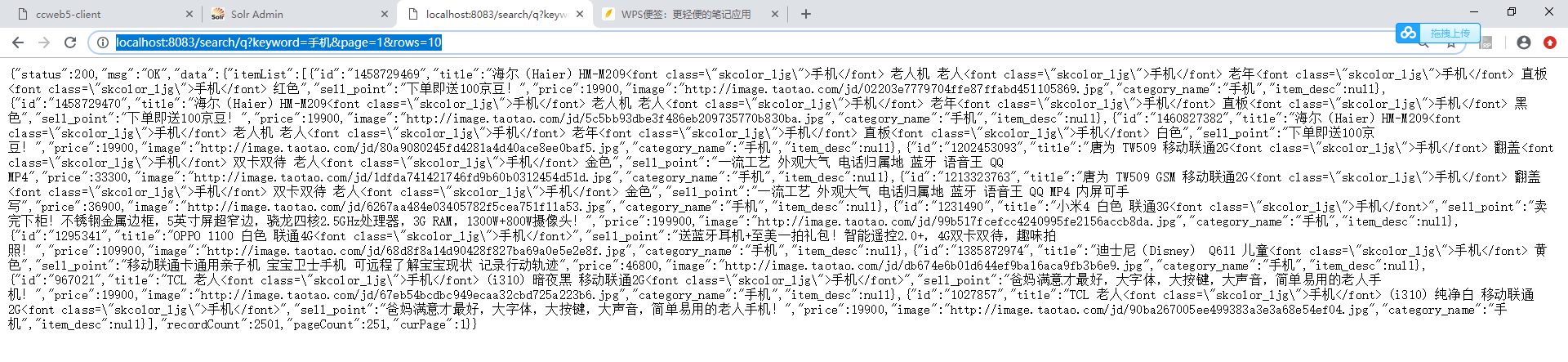
返回结果：json数据，使用TaotaoResult包装SearchResult。

|  |
| --- |
| **package** com.taotao.search.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestParam;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.common.utils.ExceptionUtil;  **import** com.taotao.search.pojo.SearchResult;  **import** com.taotao.search.service.SearchService;  @Controller  **public** **class** SearchController {    @Autowired  **private** SearchService searchService;  @RequestMapping("/q")  @ResponseBody  **public** TaotaoResult search(@RequestParam(defaultValue="")String keyword,  @RequestParam(defaultValue="1")Integer page,  @RequestParam(defaultValue="30")Integer rows) {  **try** {  //转换字符集  keyword = **new** String(keyword.getBytes("iso8859-1"), "utf-8");  SearchResult searchResult = searchService.search(keyword, page, rows);  **return** TaotaoResult.*ok*(searchResult);  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  }  } |

测试：访问链接

http://localhost:8083/search/q?keyword=手机&page=1&rows=10

如果访问出现乱码则需要手动将数据转码

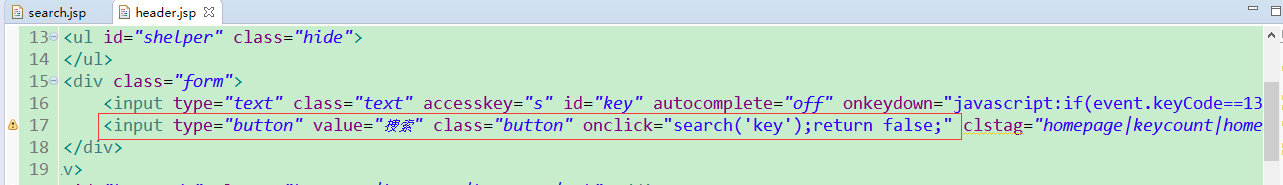


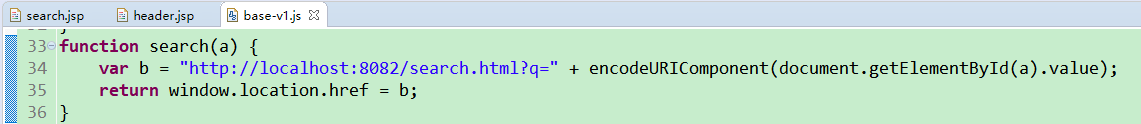
### 在portal中实现搜索

#### 分析

调用taotao-search发布的服务，实现搜索。使用HttpClient调用服务。返回json数据。需要把json转换成java对象。把java对象传递给页面。

Index.jsp中引用来了common/header.jsp，其中定义了搜索按钮





请求的url：http://localhost:8082/search.html

参数：q：查询条件

返回结果：jsp页面（search.jsp）

Search.jsp分析：

数据：

Query：查询条件

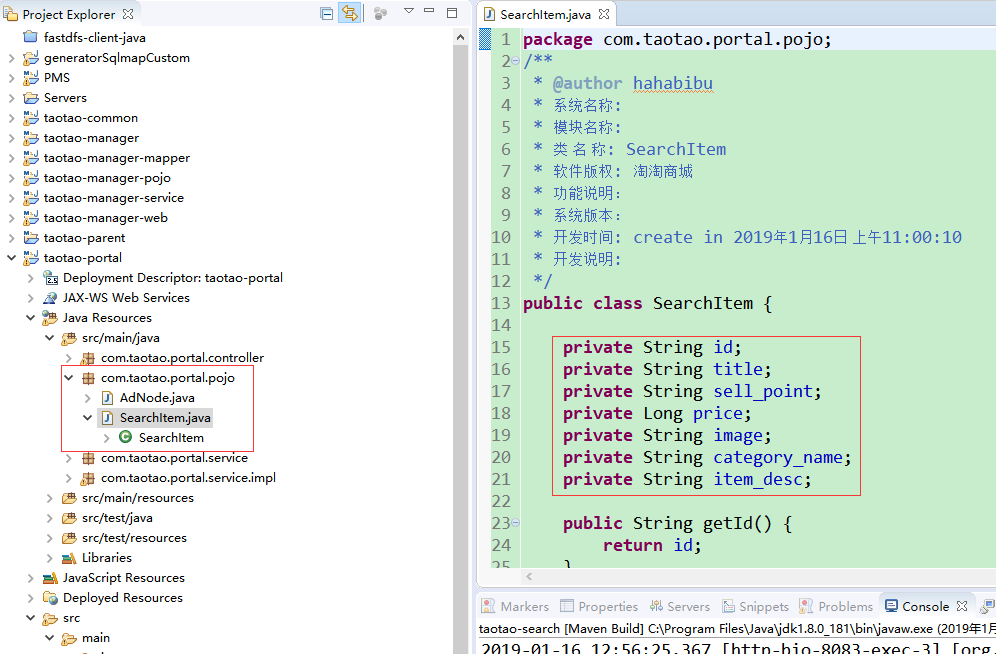
totalPages：总页数

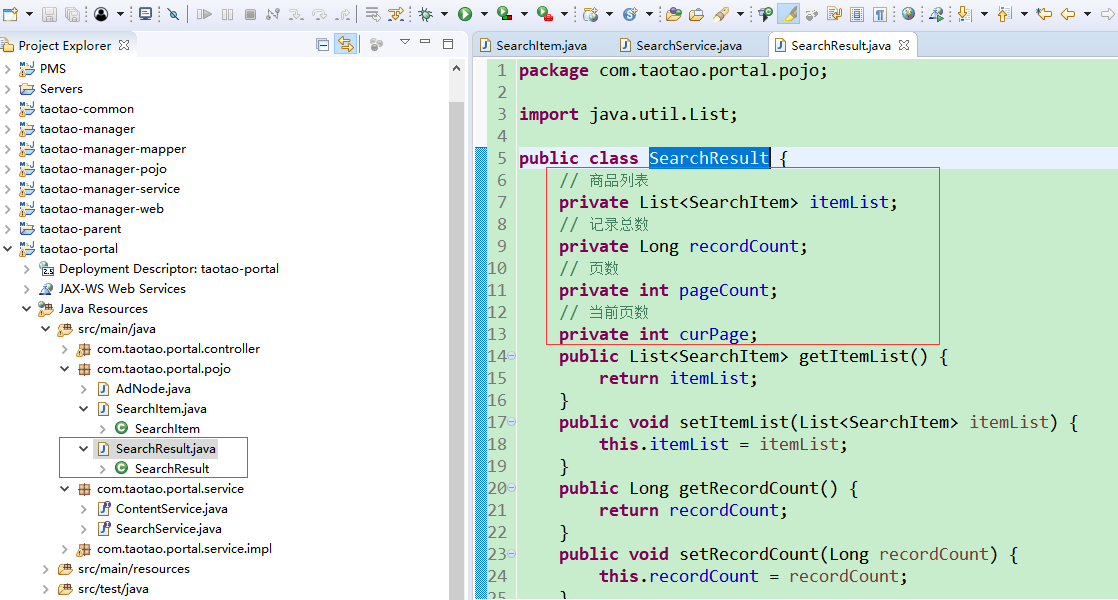
itemList：商品列表（每个元素可以是SearchItem）

Page：当前页



此处将taotao-search工程下的SearchItem.java、SearchResult.java复制到taotao-potral工程下





也可考虑将其复制到taotao-common下，然后将taotao-common重新打包

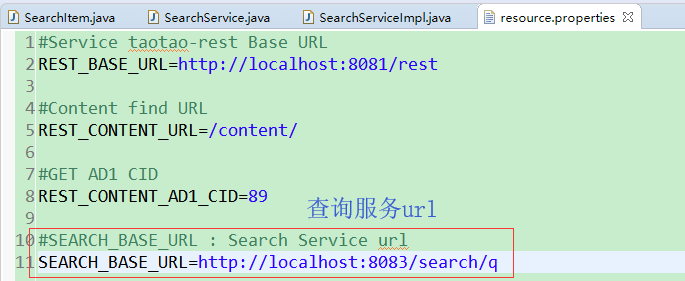
#### Service层

参数：查询条件、page、rows。

根据查询调用taotao-search发布的服务，查询商品列表。得到json数据，需要把json转换成java对象，返回SearchResult。

|  |
| --- |
| **public** **interface** SearchService {  **public** SearchResult search(String keyword, **int** page, **int** rows);  }  **SearchServiceImpl：**  **package** com.taotao.portal.service.impl;  **import** java.util.HashMap;  **import** java.util.Map;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** com.taotao.common.pojo.HttpClientUtil;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.portal.pojo.SearchResult;  **import** com.taotao.portal.service.SearchService;  @Service  **public** **class** SearchServiceImpl **implements** SearchService {  @Value("${SEARCH\_BASE\_URL}")  **private** String SEARCH\_BASE\_URL;  @Override  **public** SearchResult search(String keyword, **int** page, **int** rows) {  //调用服务查询商品列表  Map<String, String> param = **new** HashMap<>();  param.put("keyword", keyword);  param.put("page", page + "");  param.put("rows", rows + "");  //调用服务  String json = HttpClientUtil.*doGet*(SEARCH\_BASE\_URL, param);  //转换成java对象  TaotaoResult taotaoResult = TaotaoResult.*formatToPojo*(json, SearchResult.**class**);  //取返回的结果  SearchResult searchResult = (SearchResult) taotaoResult.getData();  **return** searchResult;  }  } |

在resource.properties配置SEARCH\_BASE\_URL



#### Controller

接收三个参数：查询条件、page、rows

调用服务查询商品列表。

把商品列表传递给jsp、参数回显。

返回逻辑视图（search.jsp）

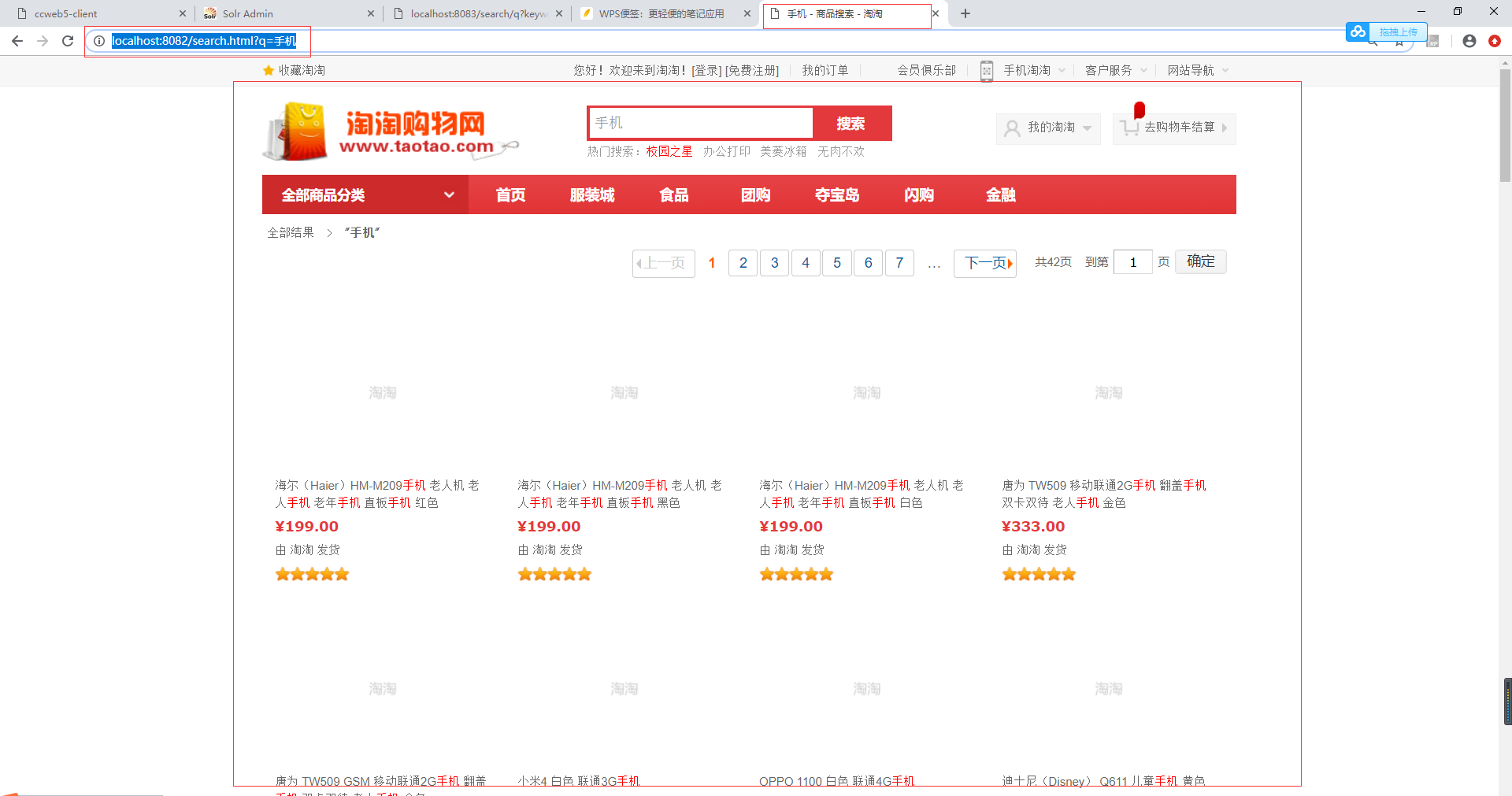
请求的url：/search

|  |
| --- |
| **package** com.taotao.portal.controller;  **import** java.io.UnsupportedEncodingException;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.Model;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestParam;  **import** com.taotao.portal.pojo.SearchResult;  **import** com.taotao.portal.service.SearchService;  @Controller  **public** **class** SearchController {  @Autowired  **private** SearchService searchService;  @RequestMapping("/search")  **public** String search(@RequestParam("q")String keyword,  @RequestParam(defaultValue="1")Integer page,  @RequestParam(defaultValue="60")Integer rows, Model model) {  //get乱码处理  **try** {  keyword = **new** String(keyword.getBytes("iso8859-1"), "utf-8");  } **catch** (UnsupportedEncodingException e) {  keyword = "";  e.printStackTrace();  }  SearchResult searchResult = searchService.search(keyword, page, rows);  //参数传递 给页面  model.addAttribute("query", keyword);  model.addAttribute("totalPages", searchResult.getPageCount());  model.addAttribute("itemList", searchResult.getItemList());  model.addAttribute("page", searchResult.getCurPage());  //返回逻辑视图  **return** "search";  }  } |

访问测试：<http://localhost:8082/search.html?q=%E6%89%8B%E6%9C%BA>

<http://localhost:8082/search.html?q=手机>

（需要启动taotao-search、再启动taotao-portal进行测试）



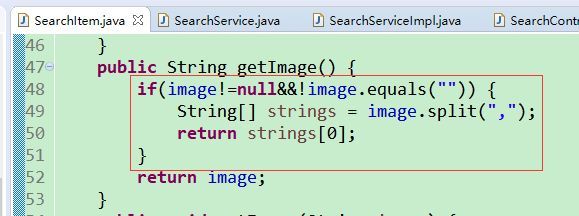
#### 问题分析：

异常处理，可能是传入的数据为乱码导致出现空指针异常



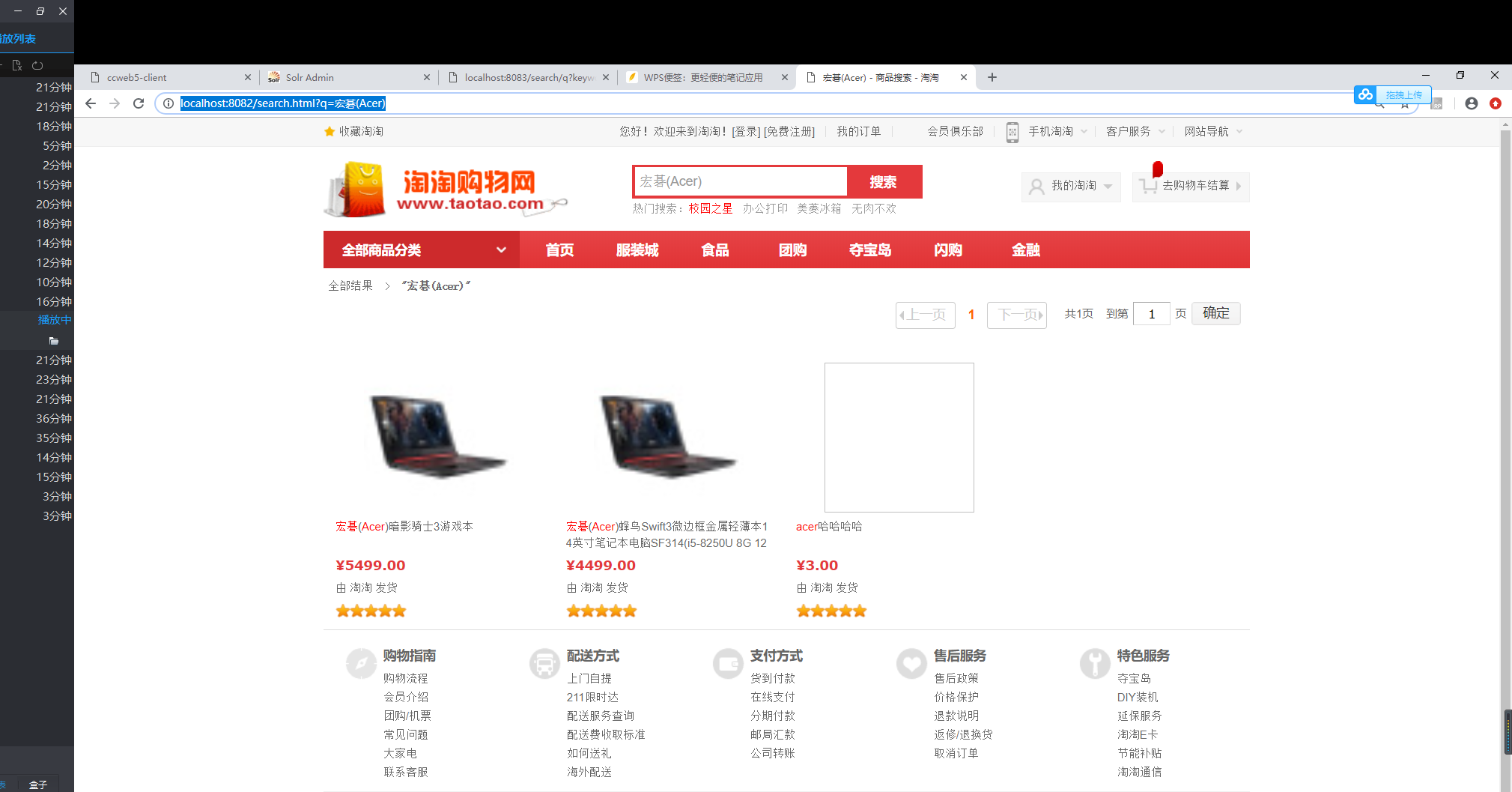
解决图片显示不出来的问题：一个商品可能对应有多个图片信息，在数据库中存储的形式是以“，”进行分割，此处在taotao-portal对查找出来的图片信息进行截取处理（切勿动taotao-search层，否则不具有通用性）

修改SearchItem.java：



后台管理系统自定义添加商品信息，查找测试：

<http://localhost:8082/search.html?q=%E5%AE%8F%E7%A2%81(Acer)>



## Solr集群

### Solr集群的架构

SolrCloud

需要用到solr+zookeeper



要完成的集群结构:



### Zookeeper

1. 集群管理

主从的管理、负载均衡、高可用的管理。集群的入口。Zookeeper必须是集群才能保证高可用。Zookeeper有选举和投票的机制。集群中至少应该有三个节点。

1. 配置文件的集中管理

搭建solr集群时，需要把Solr的配置文件上传zookeeper，让zookeeper统一管理。每个节点都到zookeeper上取配置文件。

1. 分布式锁

### 集群需要的服务器

Zookeeper：3台

Solr：4台

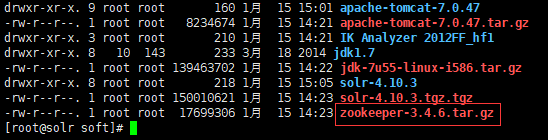
没有足够的服务器，实现伪分布式，在一台服务器上实现zookeeper三个实例、tomcat（solr）需要四个实例。**Zookeeper需要安装jdk**

在单机版solr服务器上搭建集群，192.168.187.129

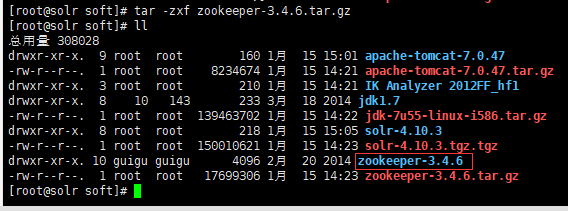
### 集群搭建步骤

#### 第一部分：Zookeeper集群搭建

##### 第一步：需要把zookeeper的安装包上传到服务器



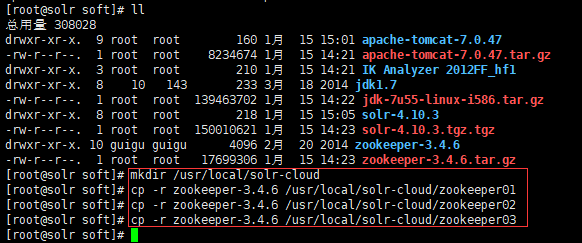
##### 第二步：把zookeeper解压

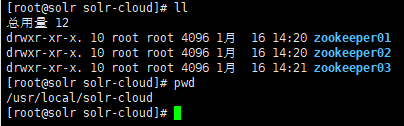


##### 第三步：把zookeeper向/usr/local/solr-cloud目录下复制三份

在/usr/local/下创建solr-cloud目录存放分布式solr集群搭建的内容

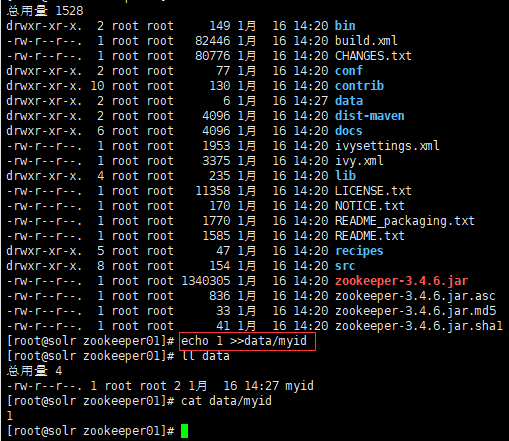
将解压后的zookeeper分别复制到对应的文件夹



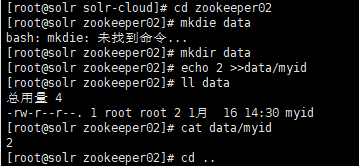


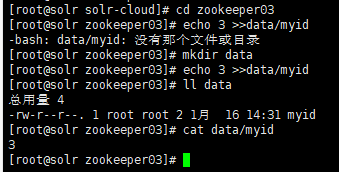
###### 配置zookeeper

1. 在zookeeper01目录下创建一个data文件夹。
2. 在data目录下创建一个myid的文件
3. Myid的内容为1（02对应“2”，03对应“3”）

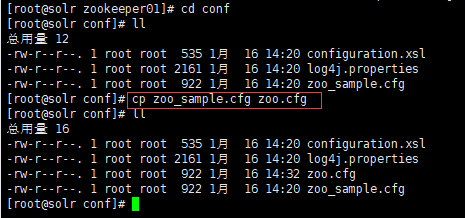


1. Zookeeper02、03以此类推





1. 进入conf文件，把zoo\_sample.cfg文件改名为zoo.cfg



1. 修改zoo.cfg，把dataDir=属性指定为刚创建的data文件夹
2. 修改zoo.cfg，把clientPort指定为不冲突的端口号（01:2181、02:2182、03:2183）

分别为：

dataDir：修改

/usr/local/solr-cloud/zookeeper01/data

/usr/local/solr-cloud/zookeeper02/data

/usr/local/solr-cloud/zookeeper03/data

ClientPort：修改（客户端连接端口）

zookeeper01：2181

zookeeper02：2182

zookeeper03：2183

1. 在zoo.cfg中添加如下内容：

服务器ip：投票端口、选举端口

server.1=192.168.25.154:2881:3881

server.2=192.168.25.154:2882:3882

server.3=192.168.25.154:2883:3883

实际配置：

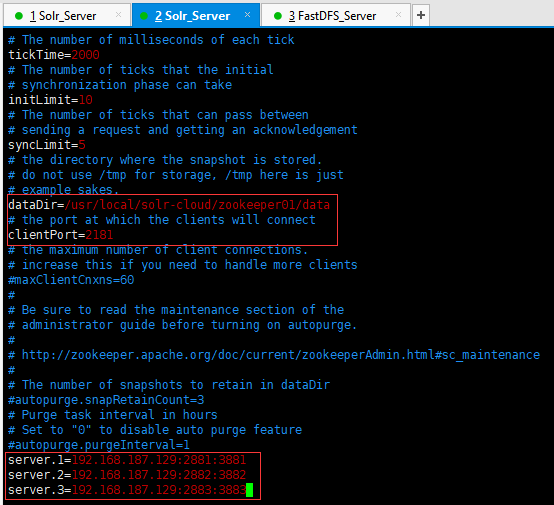
server.1=192.168.187.129:2881:3881

server.2=192.168.187.129:2882:3882

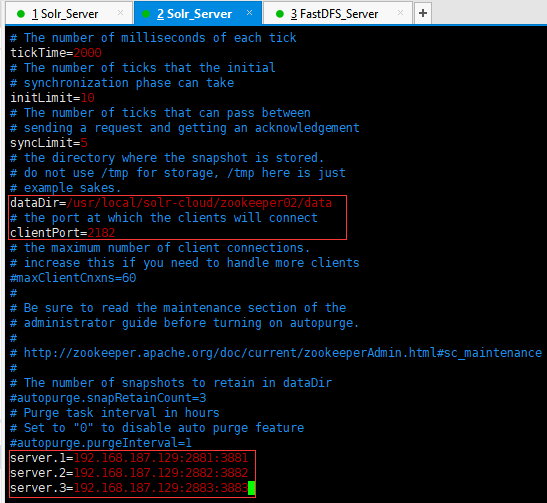
server.3=192.168.187.129:2883:3883

每个zookeeper节点都要配置上述内容，分别对应：

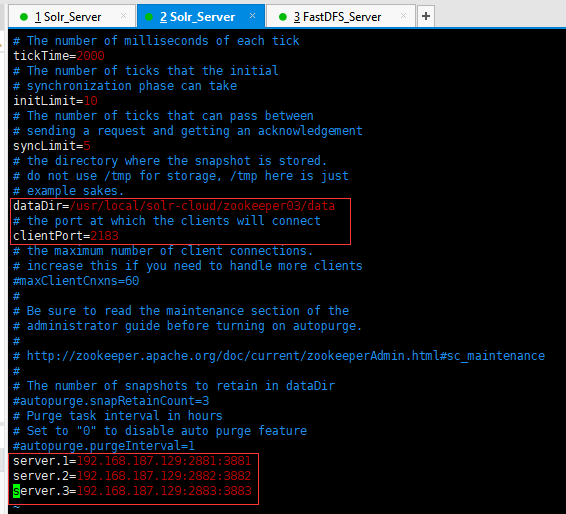
Zookeeper01/zoo.cfg：



Zookeeper02/zoo.cfg:



Zookeeper03/zoo.cfg:



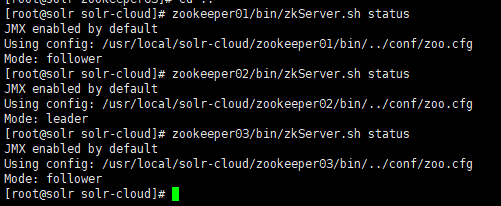
##### 第四步：启动zookeeper

Zookeeper的目录下有一个bin目录。使用zkServer.sh启动zookeeper服务。

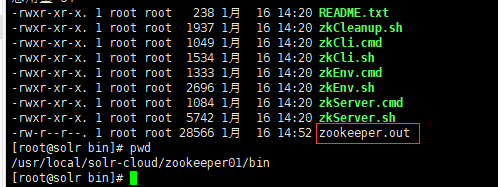
启动：./zkServer.sh start

关闭：./zkServer.sh stop

查看服务状态：./zkServer.sh status



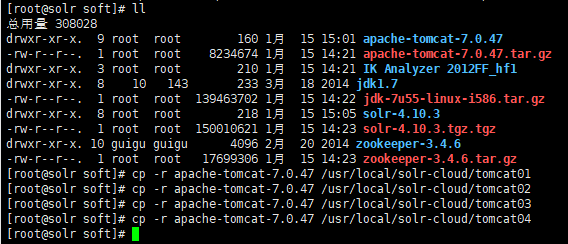
说明：如果是在zookeeper0x中启动，再其对应的bin目录下会生成相应的日志文件，如果是在外级目录去启动zookeeper则其日志文件会生成在对应的当前目录下，显示为zookeeper.out，如果是多个zookeeper则记录在相同的日志文件中

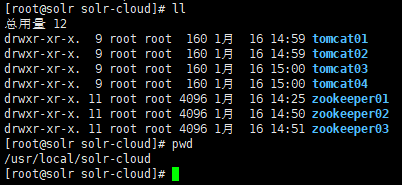


#### 第二部分：搭建solr集群

##### 第一步：安装四个tomcat，修改其端口号不能冲突。8080~8083

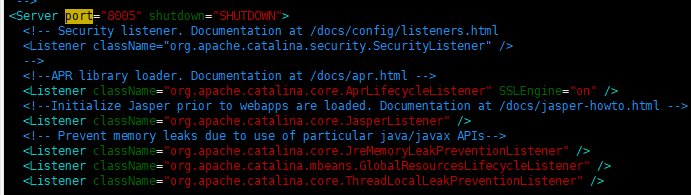
在/root/soft目录下将解压后的tomcat分别复制到solr集群搭建的位置，分别进行命名

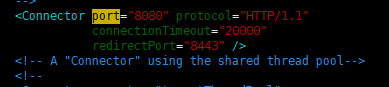




tomcat01可以不变，依次改变tomcat02(端口号+1)、tomcat03（端口号+2）、tomcat04（端口号+3）下conf/server.xml

修改3个位置的端口号信息，此处简单显示说明：

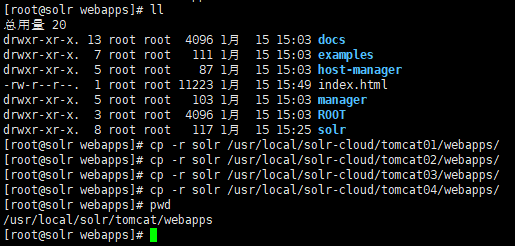




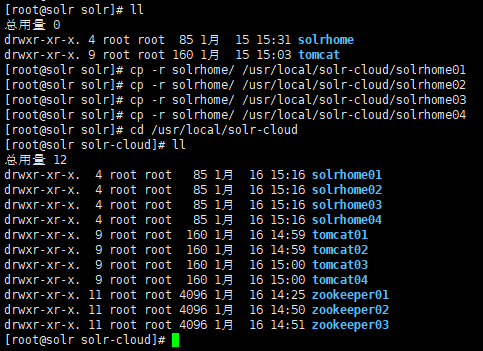


##### 第二步：向tomcat下部署solr。把单机版的solr工程复制到tomcat下即可

先关闭单机版的solr服务器，随后将相关的工程复制到tomcat目录下



##### 第三步：为每个solr实例创建一个相应的solrhome



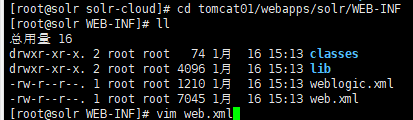
##### 第四步：为每个solr实例关联对应的solrhome，修改web.xml

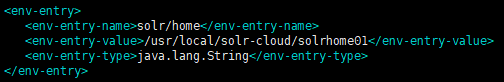
vim /usr/local/solr-cloud/tomcat01/webapps/solr/WEB-INF/web.xml

vim /usr/local/solr-cloud/tomcat02/webapps/solr/WEB-INF/web.xml

vim /usr/local/solr-cloud/tomcat03/webapps/solr/WEB-INF/web.xml

vim /usr/local/solr-cloud/tomcat04/webapps/solr/WEB-INF/web.xml





以此类推tomcat02、tomcat03、tomcat04依次一一对应，此处不做赘述

##### 第五步：修改每个solrhome下的solr.xml文件。修改host、hostPort两个属性，分别是对应的ip及端口号。

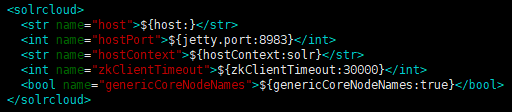
由于是伪分布，此处搭建的主机ip均一致为“192.168.187.129”，如果是不同的服务器上部署则修改为相应的ip，端口号则为相应的访问的端口号，依次为8080、8081、8082、8083

vim /usr/local/solr-cloud/solrhome01/solr.xml

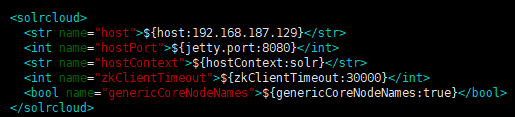
vim /usr/local/solr-cloud/solrhome02/solr.xml

vim /usr/local/solr-cloud/solrhome03/solr.xml

vim /usr/local/solr-cloud/solrhome04/solr.xml



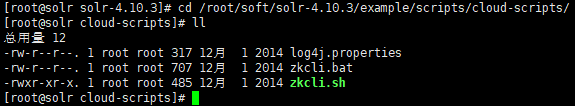
修改为以下内容：（solrhome02、solrhome03、solrhome04修改以此类推）



##### 第六步：把配置文件上传到zookeeper，需要使用

需要先启动zookeeper集群（分别启动每个zookeeper节点）

/root/solr-4.10.3/example/scripts/cloud-scripts/zkcli.sh命令上传配置文件



把/usr/local/solr-cloud/solrhome01/collection1/conf目录上传到zookeeper

（上传任意一个solrhome下的collection1/conf目录，conf下有两个比较重要的文件schema.xml、solrconfig.xml）需要zookeeper集群已经启动

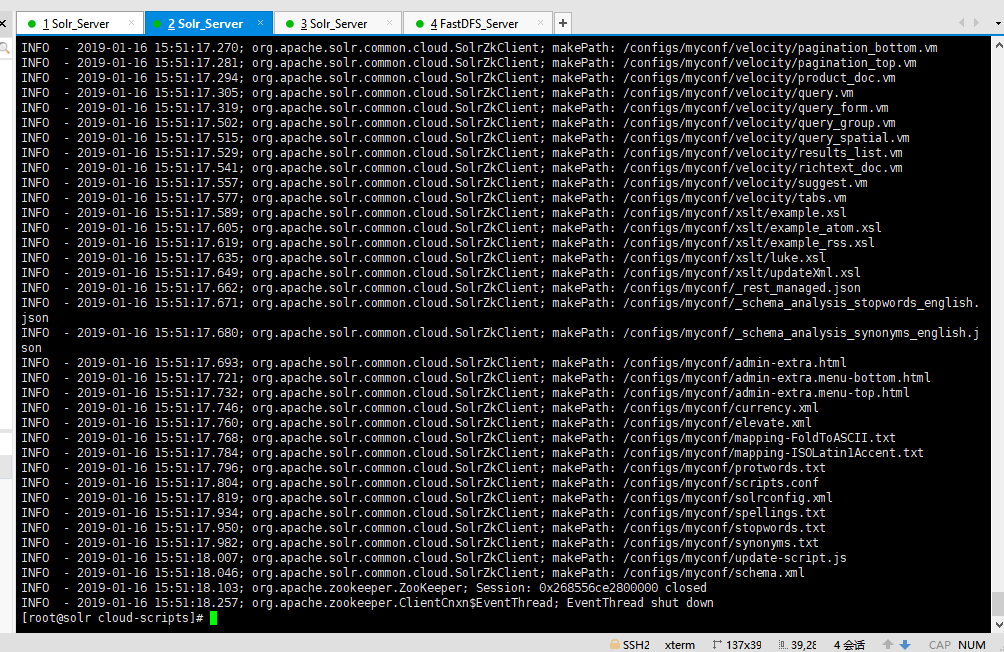
|  |
| --- |
| ./zkcli.sh -zkhost 192.168.25.154:2181,192.168.25.154:2182,192.168.25.154:2183 -cmd upconfig -confdir /usr/local/solr-cloud/solrhome01/collection1/conf -confname myconf |

实际配置如下：

./zkcli.sh -zkhost 192.168.187.129:2181,192.168.187.129:2182,192.168.187.129:2183 -cmd upconfig -confdir /usr/local/solr-cloud/solrhome01/collection1/conf -confname myconf

分别对应服务器ip和zookeeper配置的端口号

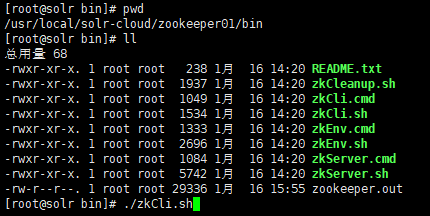
回车，等待传输完成

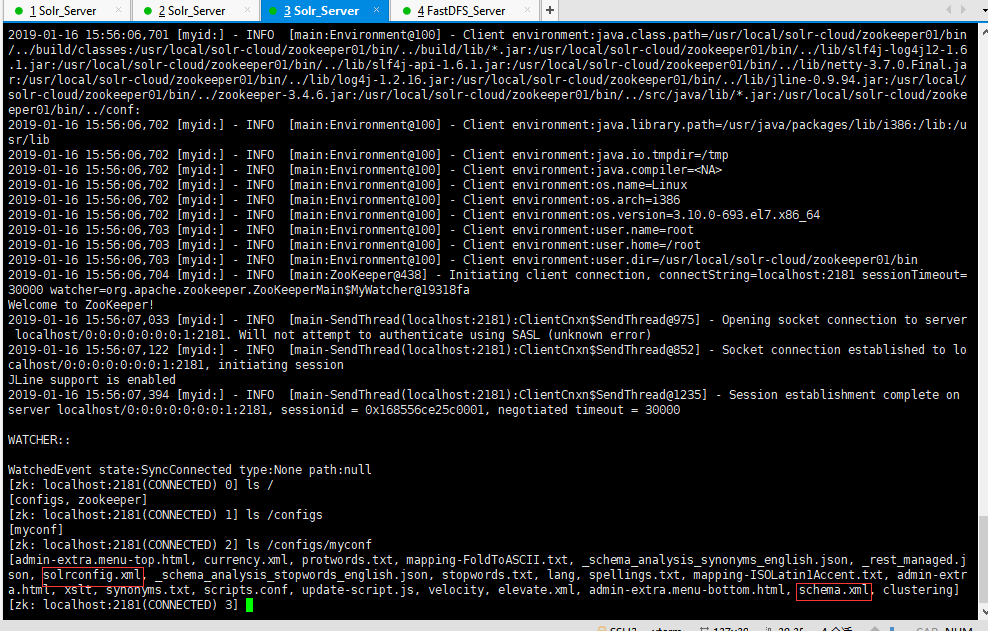


##### 第七步：查看是否上传成功

使用zookeeper的zkCli.sh命令。

查看集群中的信息，进入到任意zookeeper，bin目录下有zkCli.sh指令（每个节点的信息都是保持一致的）



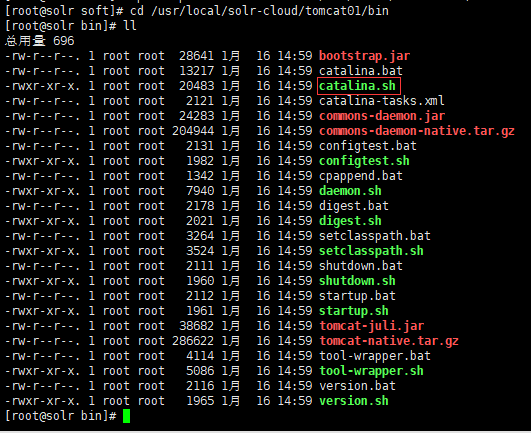


##### 第八步：告诉solr实例zookeeper的位置，需要修改tomcat的catalina.sh

在每个节点中均添加参数：vim /usr/local/solr-cloud/tomcat01/bin/catalina.sh

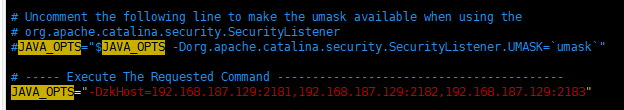
JAVA\_OPTS="-DzkHost=192.168.25.154:2181,192.168.25.154:2182,192.168.25.154:2183"

需要注意参数添加的位置，此处需要在JAVA\_OPTS使用前进行定义，否则可能导致出错



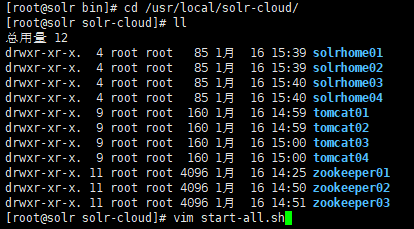
实际修改的内容：

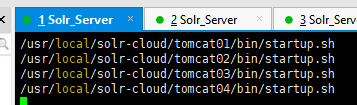
JAVA\_OPTS="-DzkHost=192.168.187.129:2181,192.168.187.129:2182,192.168.187.129:2183"



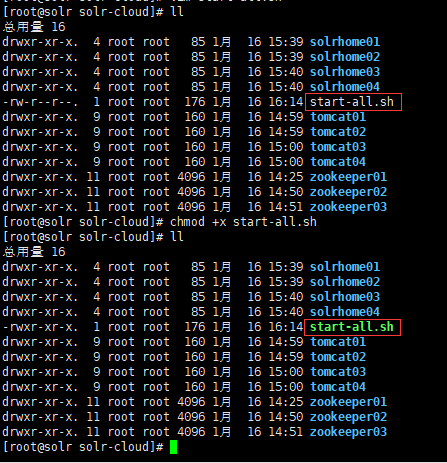
##### 第九步：启动每个solr实例

在/usr/local/solr-cloud下创建start-all.sh脚本

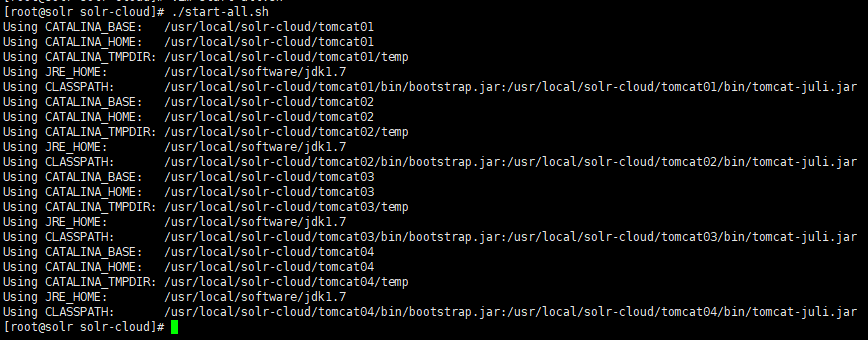




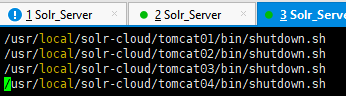
一开始是不可执行的，需要赋予权限



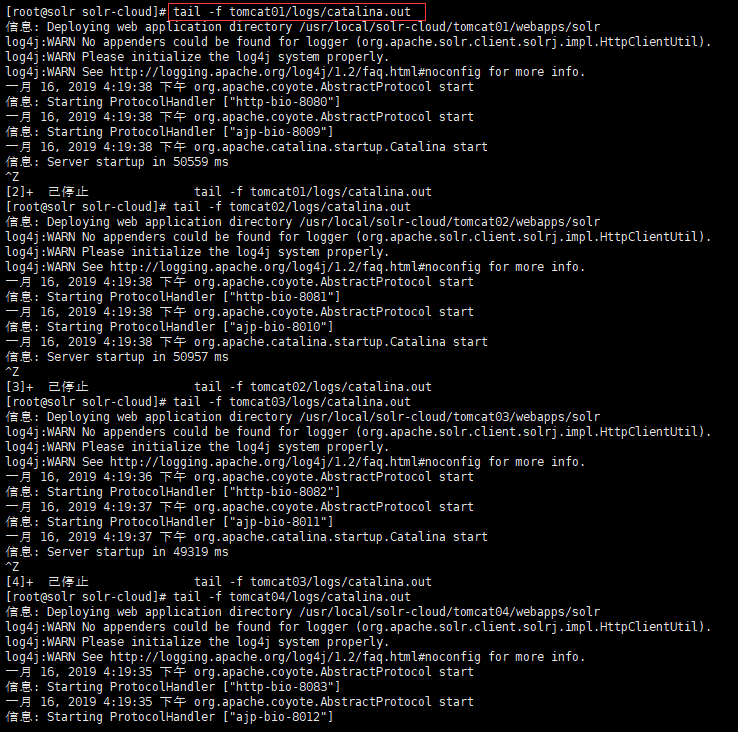
执行./start-all.sh启动tomcat服务



以此类推，也可通过编写shutdown-all.sh，完成tomcat服务的依次关闭，定义方式参考上述



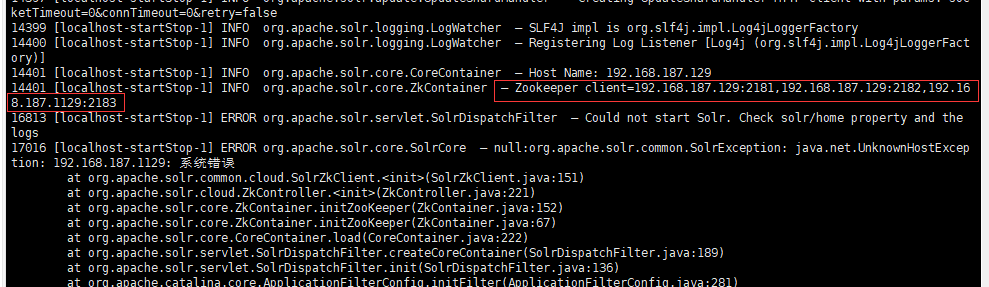
依次查看每个tomcat的启动日志



###### 问题分析

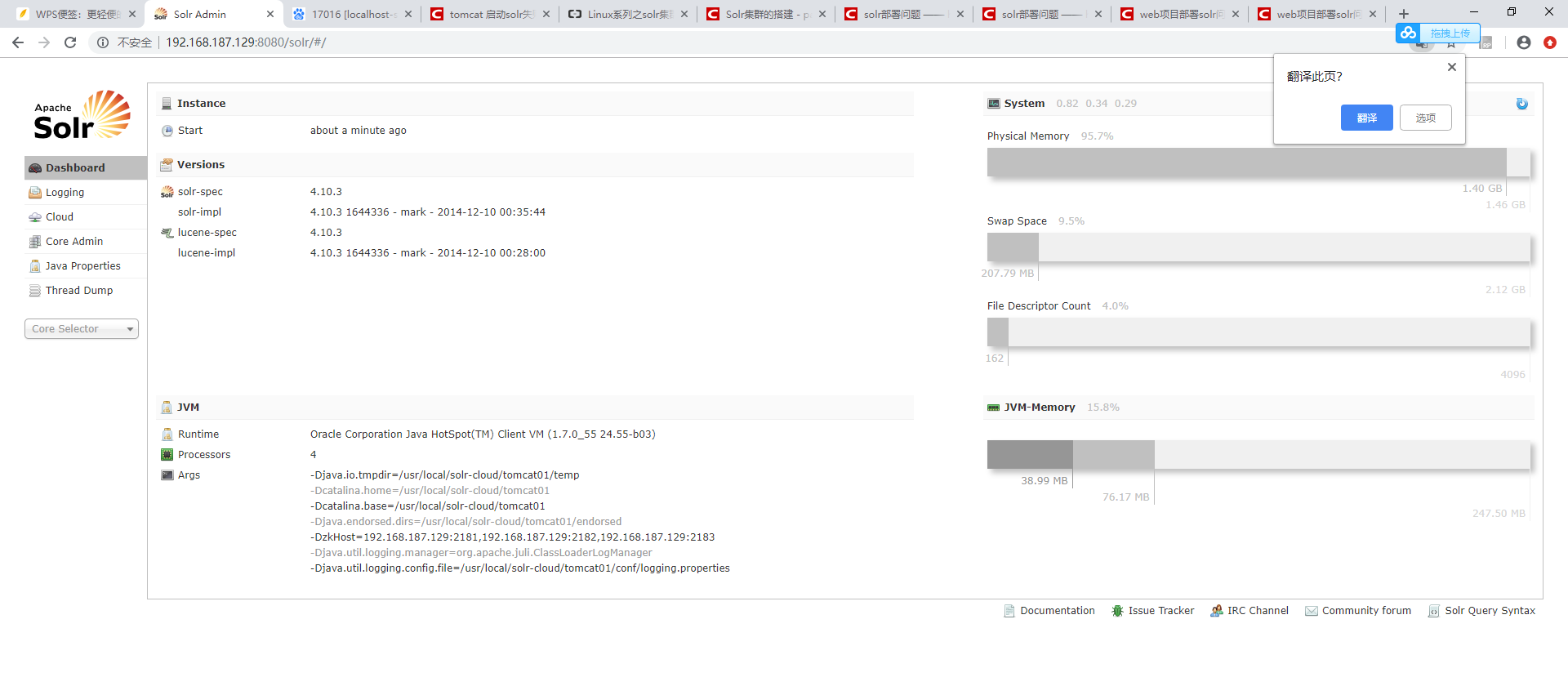
启动出错，查看日志信息，一一进行排除，除却日志信息错误，还报一个严重的错误，zookeeper配置连接ip错误，即在第八步配置时不小心将ip地址输入错误，需要进行修正，修成后重新进行测试：

JAVA\_OPTS="-DzkHost=192.168.187.129:2181,192.168.187.129:2182,192.168.187.129:2183"

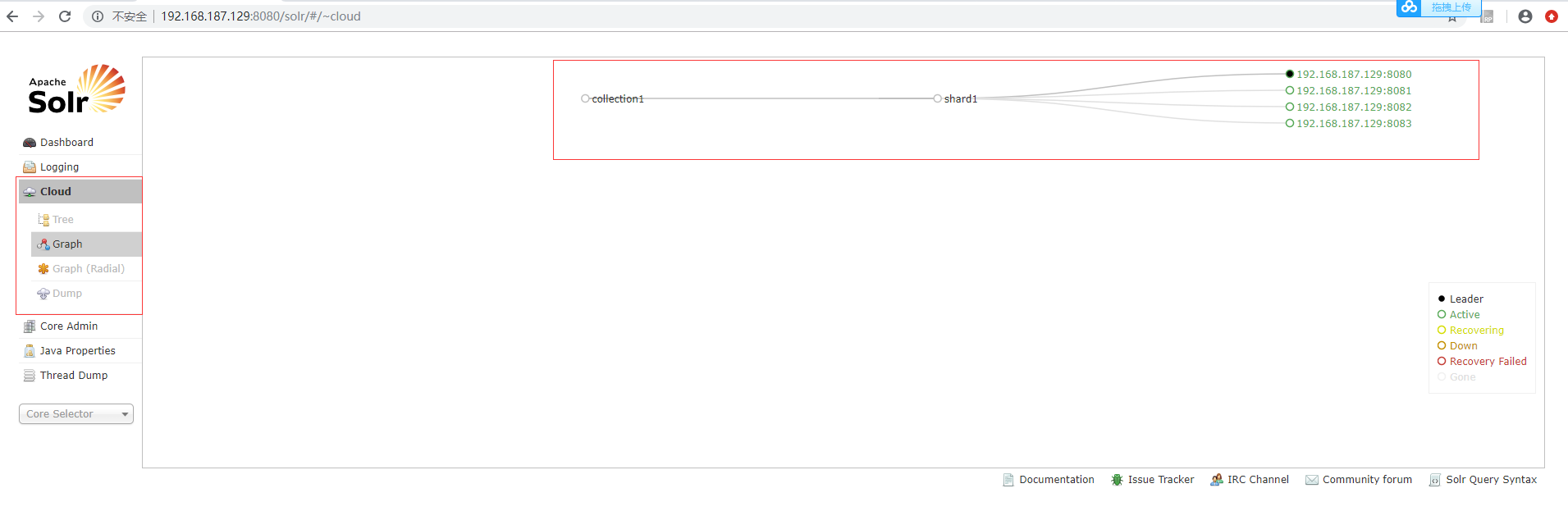


访问网页数据：<http://192.168.187.129:8080/solr/#/>

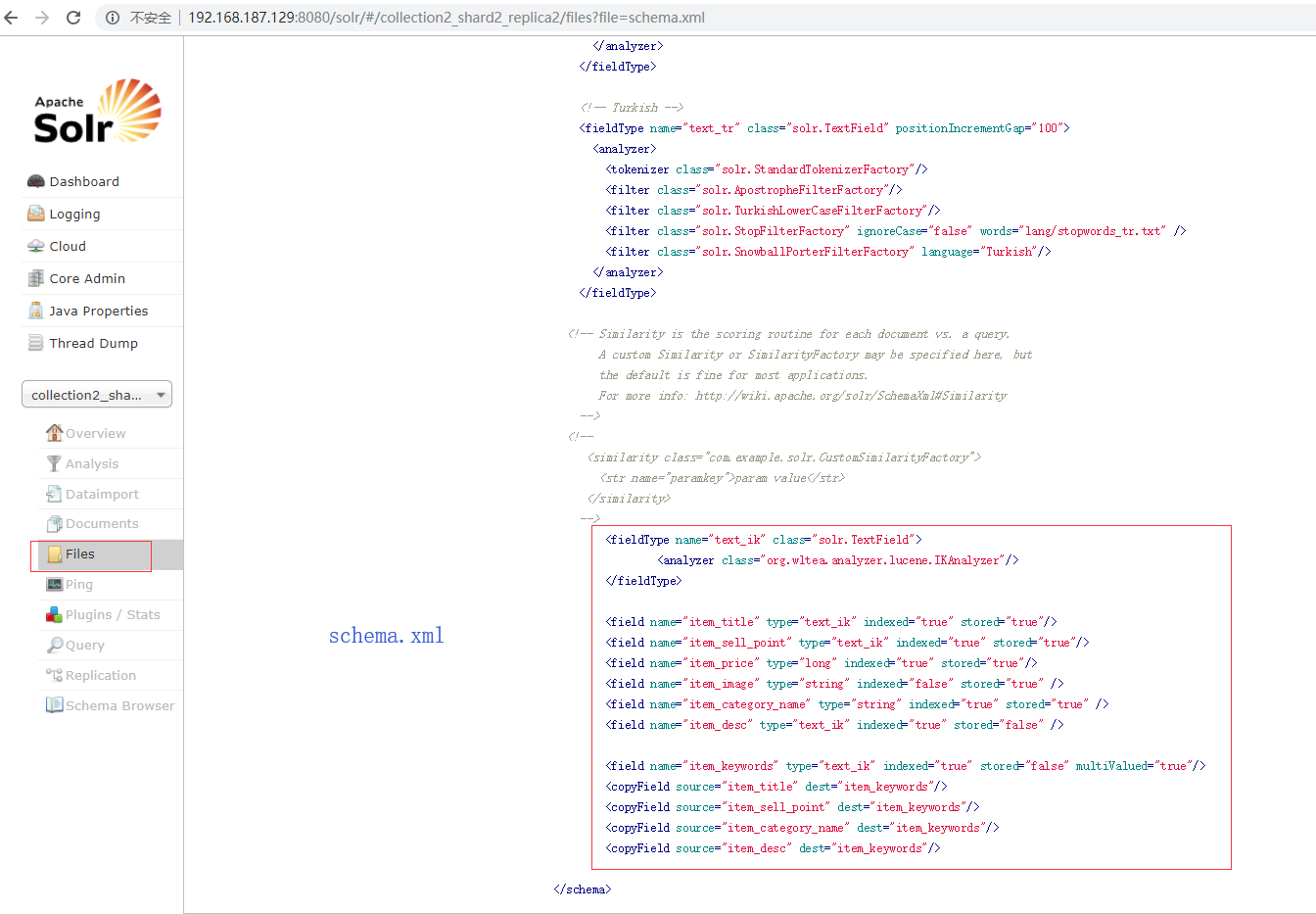
修改完成，每个数据正常访问



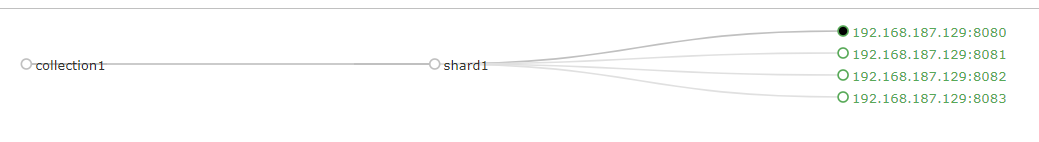
查看分析：



在任意节点都能够访问相关的数据



##### 第十步：集群分片

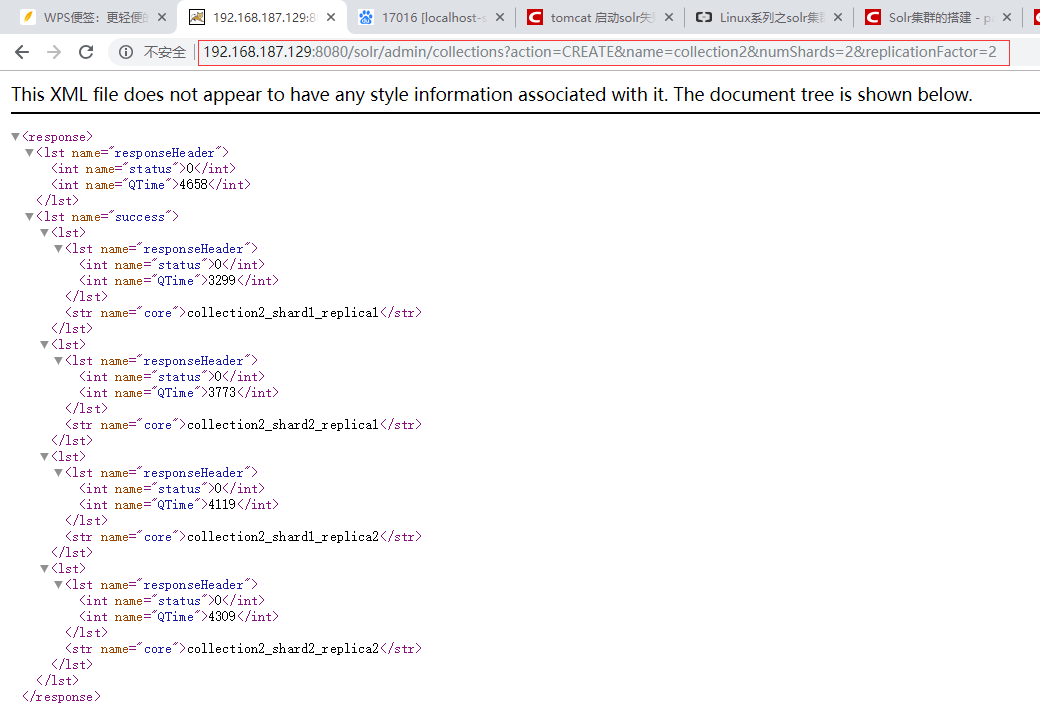


将集群分为两片，每片两个副本。

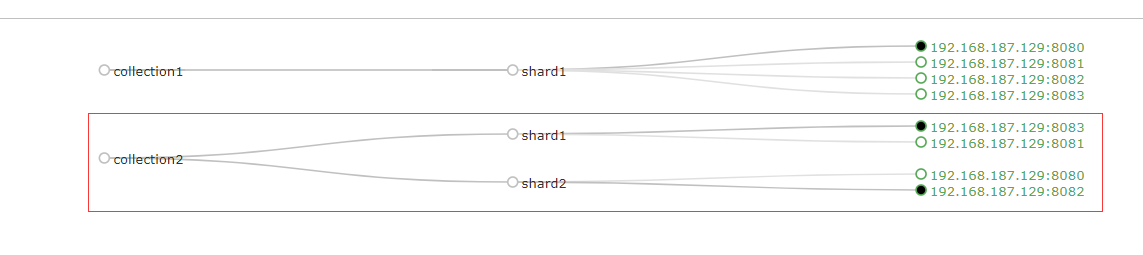
<http://192.168.25.154:8080/solr/admin/collections?action=CREATE&name=collection2&numShards=2&replicationFactor=2>

实际访问连接：

<http://192.168.187.129:8080/solr/admin/collections?action=CREATE&name=collection2&numShards=2&replicationFactor=2>



查看分片后的数据视图：

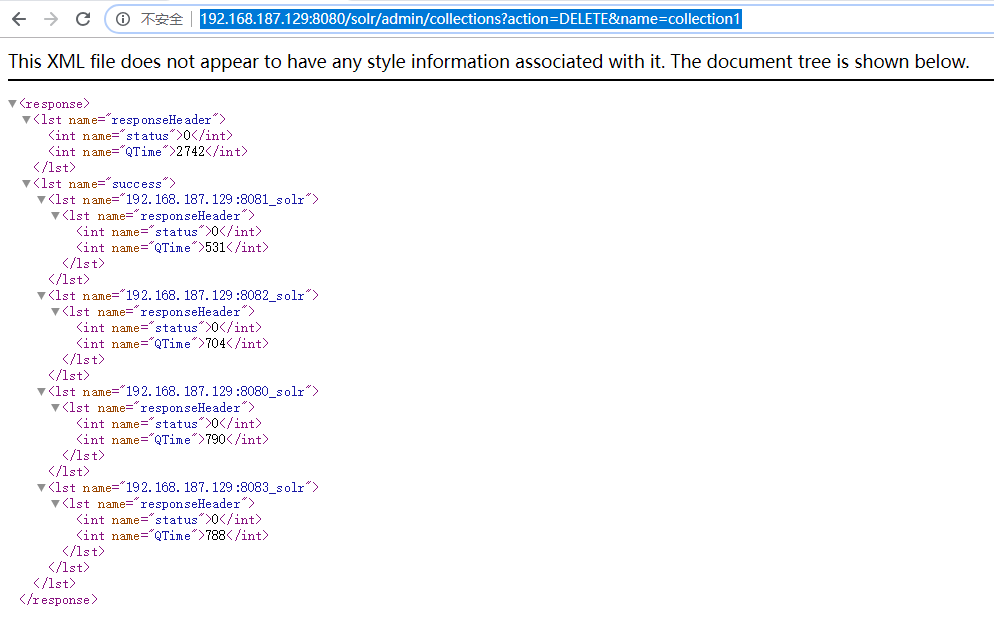


##### 第十一步：删除不用collection1

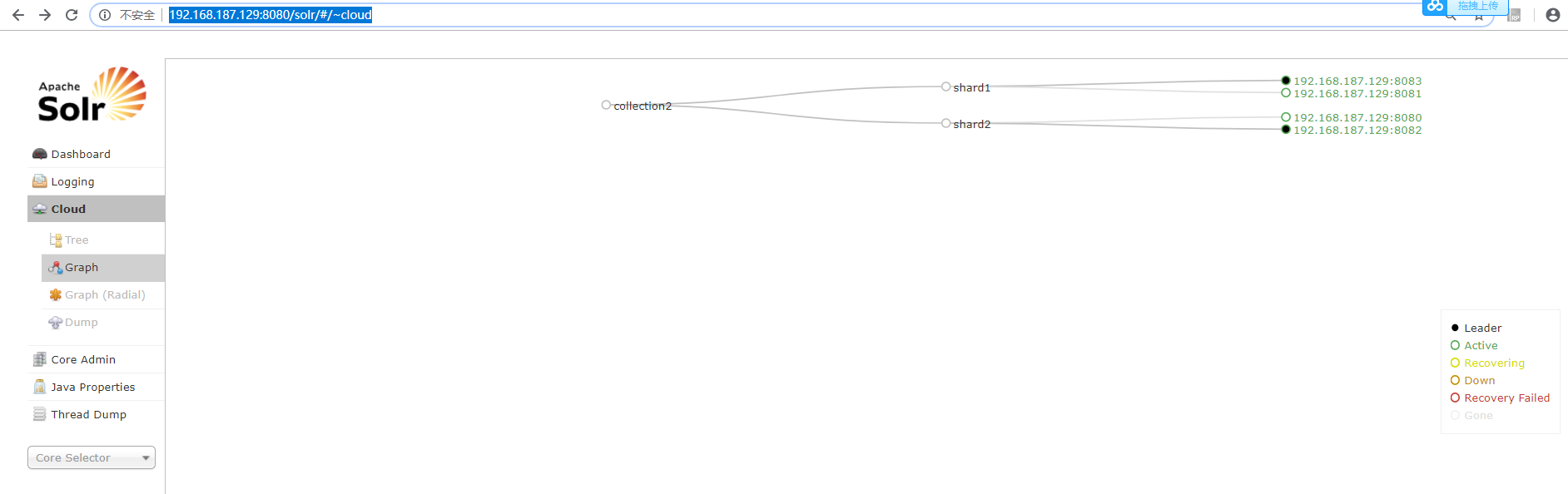
<http://192.168.25.154:8080/solr/admin/collections?action=DELETE&name=collection1>

实际访问连接：

<http://192.168.187.129:8080/solr/admin/collections?action=DELETE&name=collection1>



刷新访问：

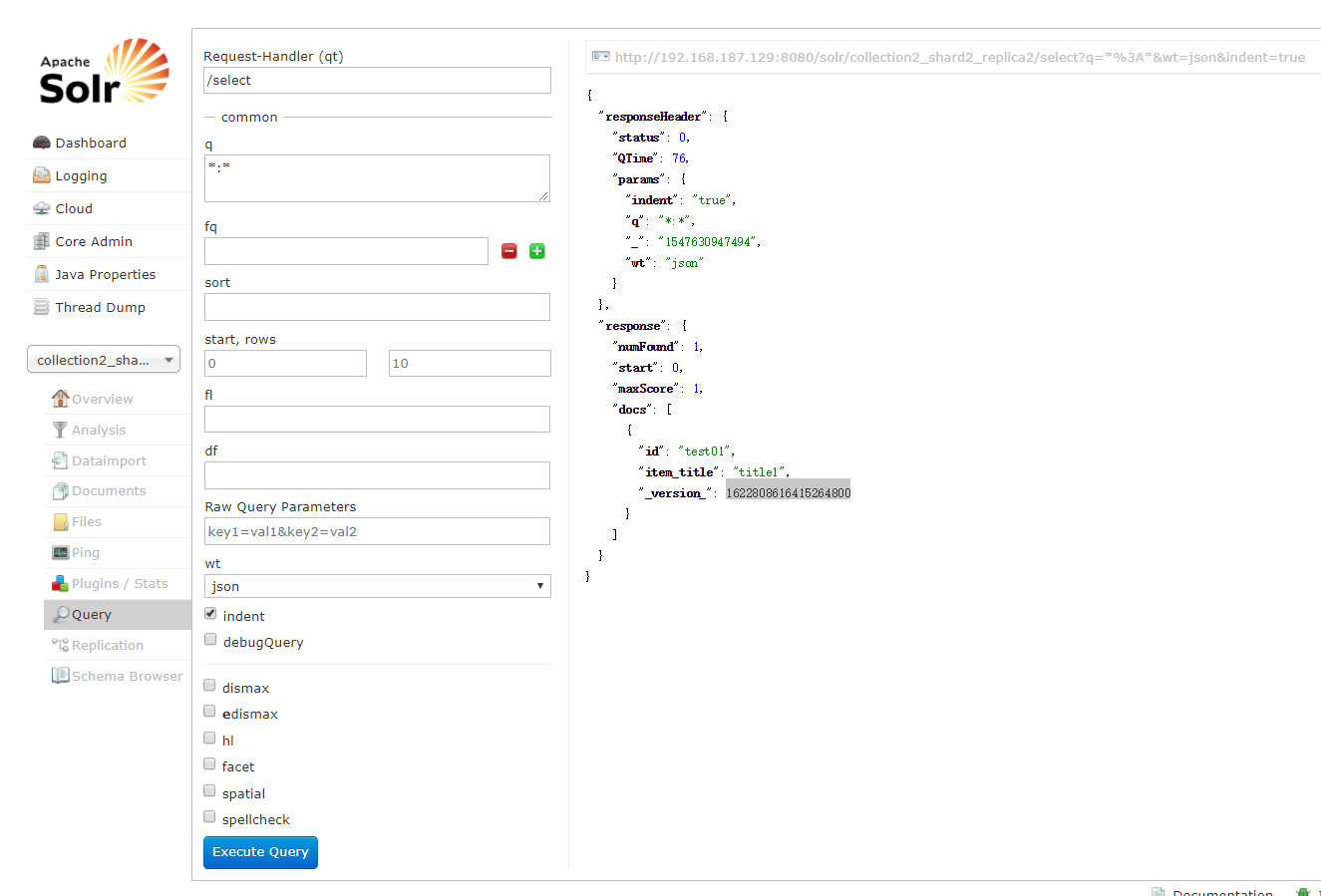


### 使用solrJ连接集群

|  |
| --- |
| @Test  **public** **void** testSolrClout() **throws** Exception {  //创建一个SolrServer对象  CloudSolrServer solrServer = **new** CloudSolrServer("192.168.25.154:2181,192.168.25.154:2182,192.168.25.154:2183");  //设置默认的collection  solrServer.setDefaultCollection("collection2");  //创建一个文档对象  SolrInputDocument document = **new** SolrInputDocument();  document.addField("id", "test01");  document.addField("item\_title", "title1");  //添加文档  solrServer.add(document);  //提交  solrServer.commit();  } |



访问测试：



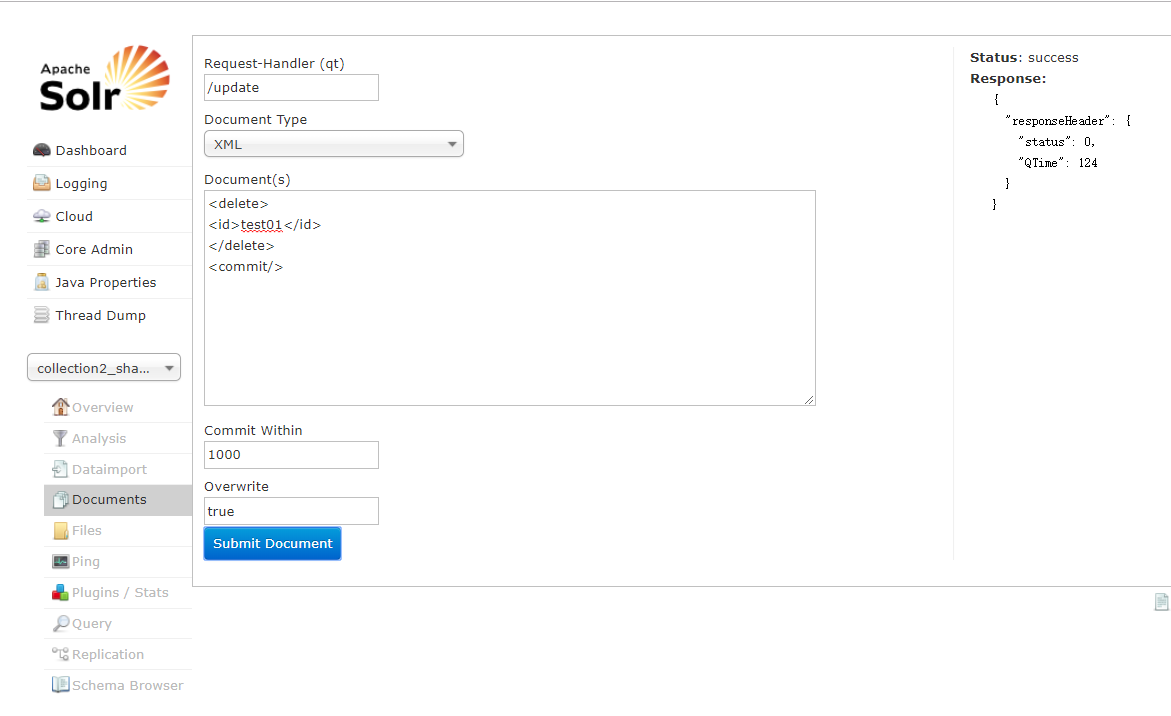
如果访问失败则可能是防火墙的问题，需要关闭防火墙！！

### 项目切换到集群

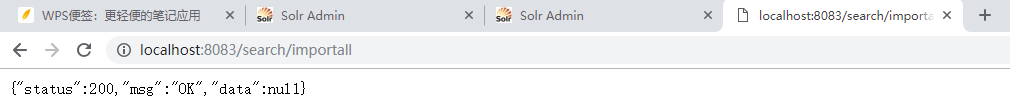
只需要在spring容器中配置一个集群版的SolrServer对象即可。



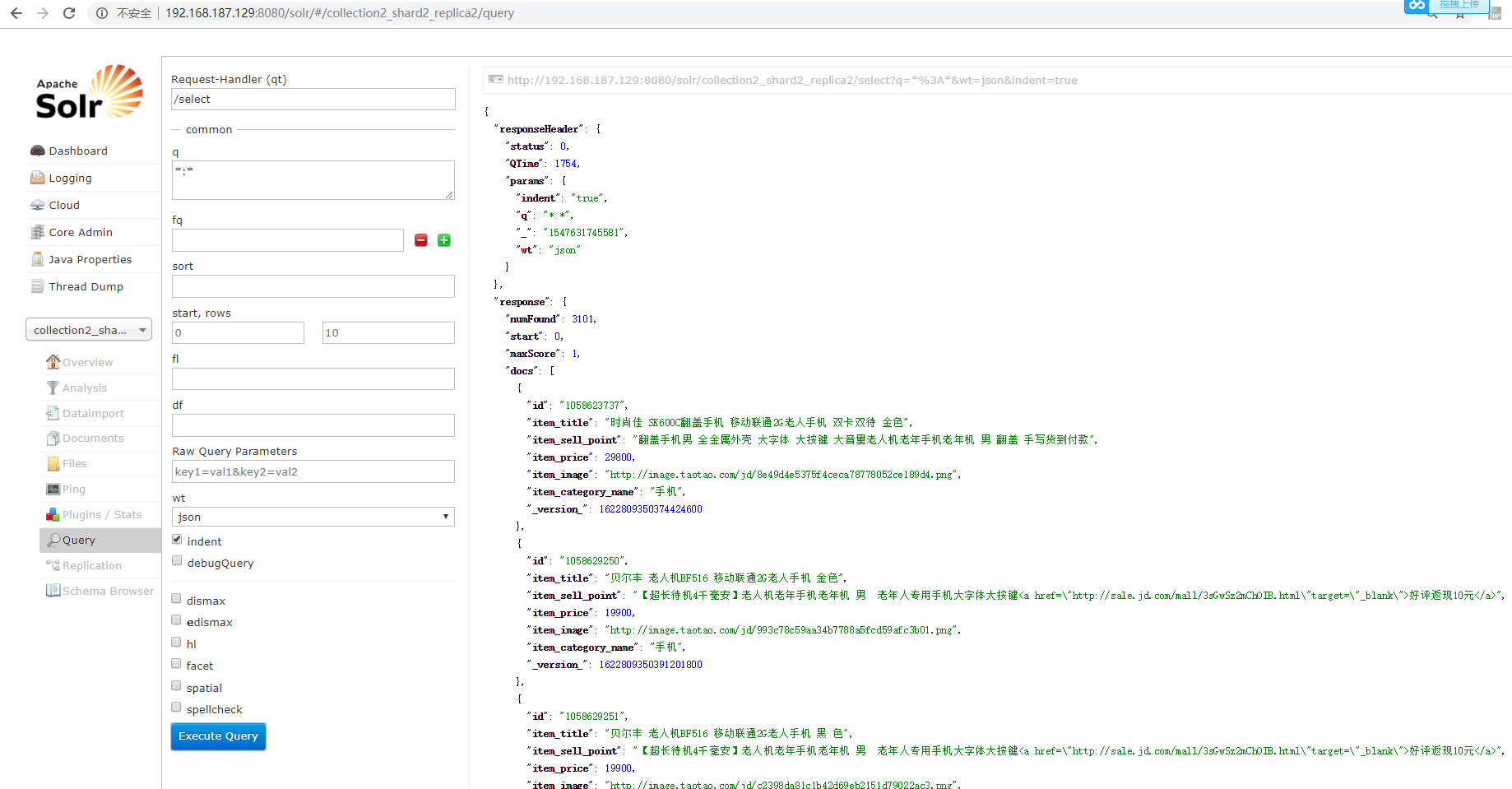
删除集合中的数据：



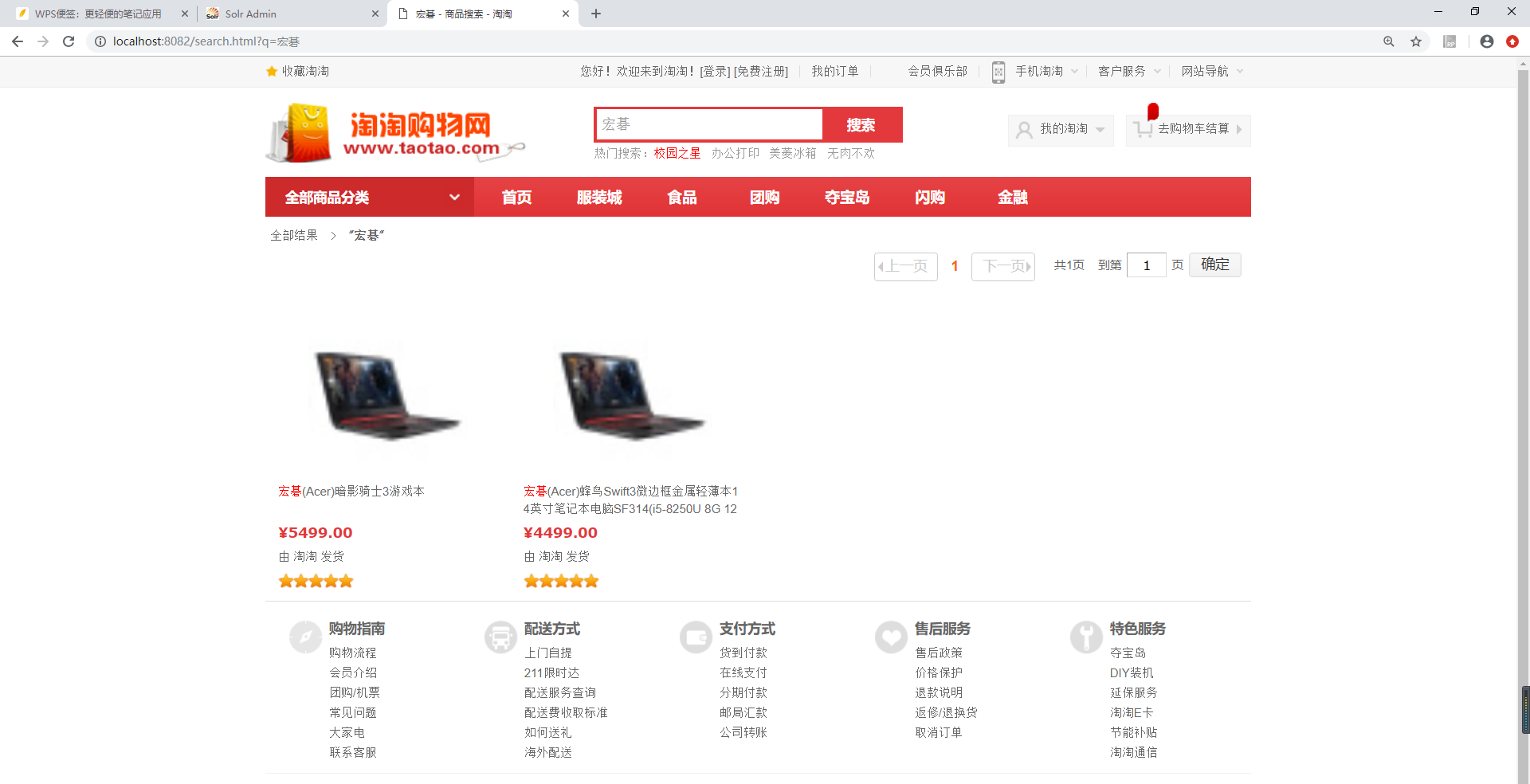
启动taotao-search，访问http://localhost:8083/search/importall测试，集群版访问较单机版慢



访问测试：



启动taotao-rest、taotao-portal,测试查找服务



如果重现错误，重启、刷新、耐心等待！！

## 商品同步-待定

修改商品信息后，需要把商品信息同步到索引库。需要search工程发布服务，在后台系统如果商品信息发生修改后，可以在后台系统的Controller中调用发布的服务。把商品id传递给服务，服务进行索引库的内容同步。