淘淘商城第八天

# 第七天内容回顾

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。

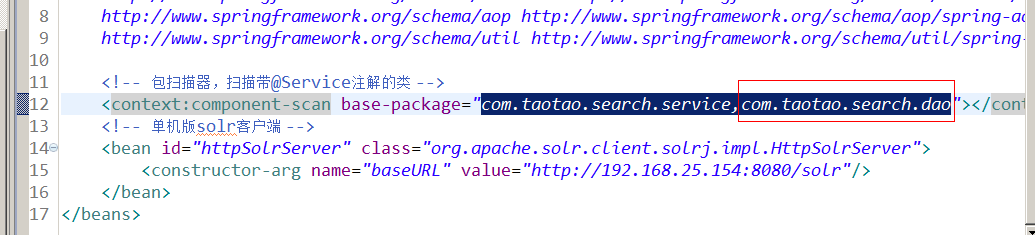
参数：SolrQuery对象

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

|  |
| --- |
| @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;  }  } |



### 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

|  |
| --- |
| @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

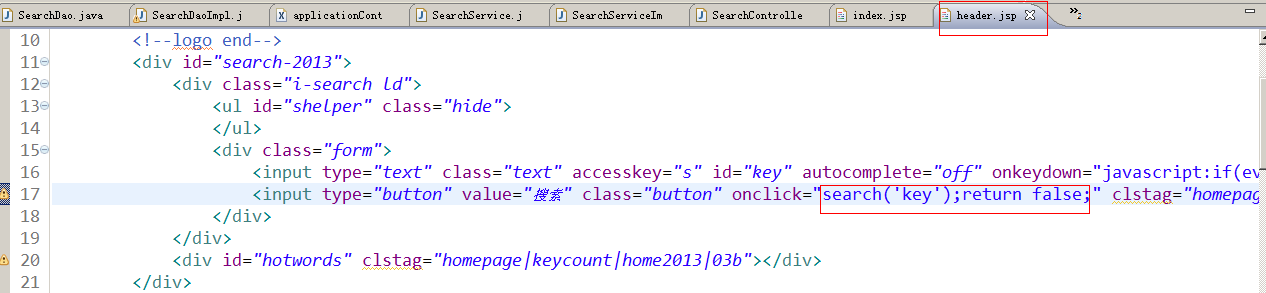
返回结果：json数据，使用TaotaoResult包装SearchResult。

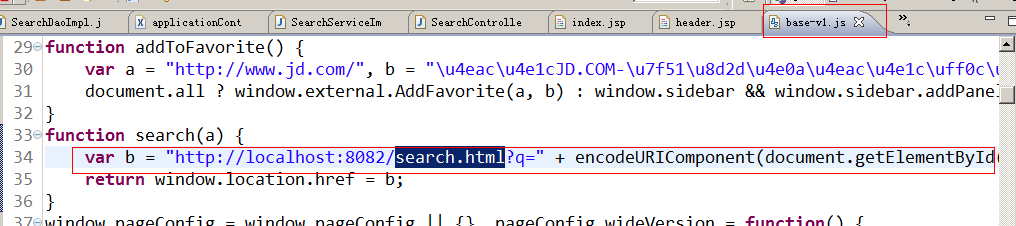
|  |
| --- |
| @Controller  **public** **class** SearchController {    @Autowired  **private** SearchService searchService;    @RequestMapping("/q")  @ResponseBody  **public** TaotaoResult search(@RequestParam(defaultValue="")String keyword,  @RequestParam(defaultValue="1")Integer page,  @RequestParam(defaultValue="30")Integer 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));  }      }  } |

## 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。





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

### 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。

|  |
| --- |
| @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;  }  } |

### 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。

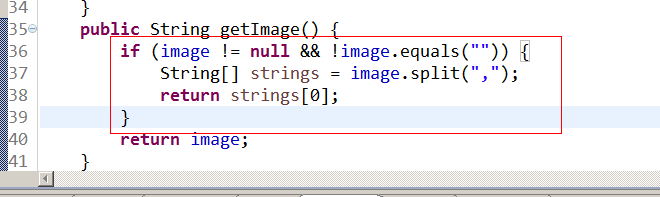
返回逻辑视图（search.jsp）

请求的url：/search

|  |
| --- |
| @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";  }  } |

解决图片显示不出来的问题：

修改ItemSearch：



# 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
2. 忘了

## 集群需要的服务器

Zookeeper：3台

Solr：4台

伪分布式，zookeeper三个实例、tomcat（solr）需要四个实例。

Zookeeper需要安装jdk。

## 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。

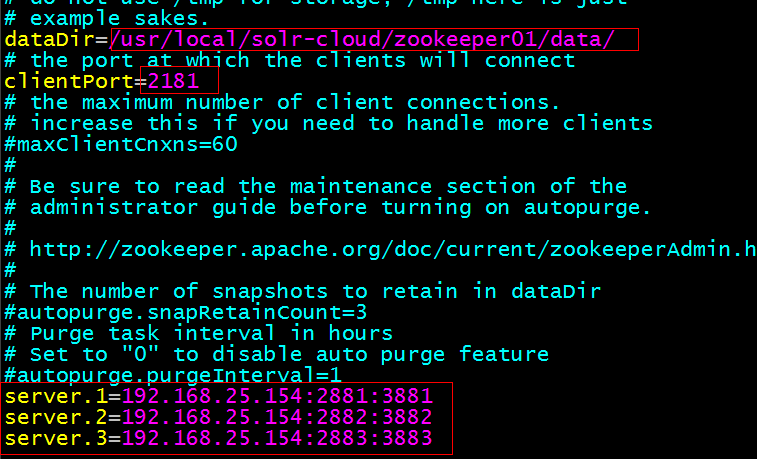
第三步：配置zookeeper。

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”）
4. Zookeeper02、03以此类推。
5. 进入conf文件，把zoo\_sample.cfg文件改名为zoo.cfg
6. 修改zoo.cfg，把dataDir=属性指定为刚创建的data文件夹。
7. 修改zoo.cfg，把clientPort指定为不冲突的端口号（01:2181、02:2182、03:2183）
8. 在zoo.cfg中添加如下内容：

server.1=192.168.25.154:2881:3881

server.2=192.168.25.154:2882:3882

server.3=192.168.25.154:2883:3883



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

第二部分：搭建solr集群

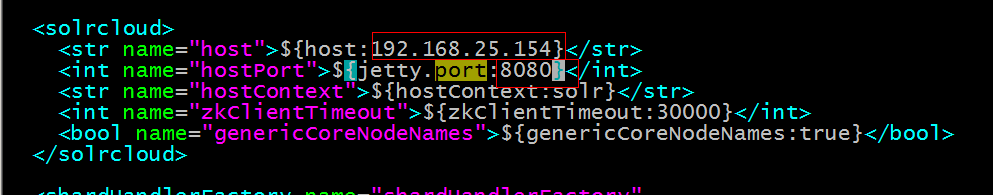
第一步：安装四个tomcat，修改其端口号不能冲突。8080~8083

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

第三步：为每个solr实例创建一solrhome。

第四步：为每个solr实例关联对应的solrhome。修改web.xml

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。



第六步：把配置文件上传到zookeeper。需要使用

/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。

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 |

第七步：查看是否上传成功。

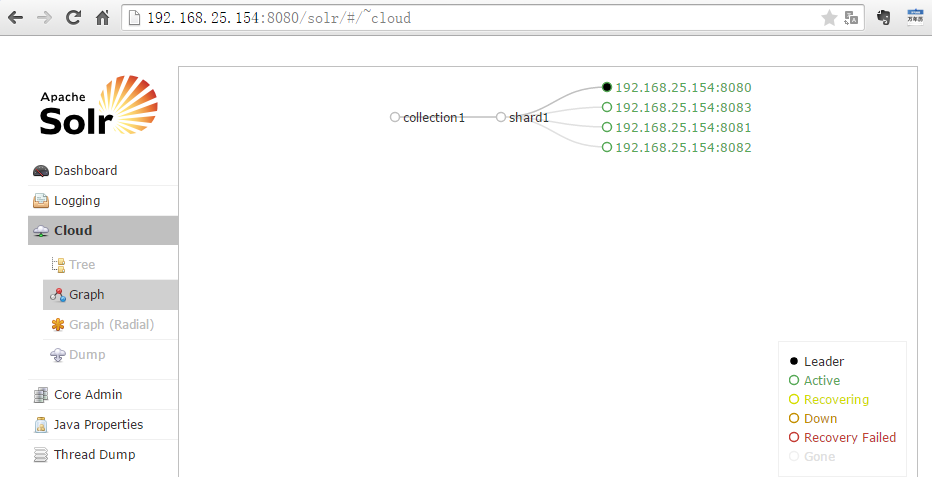
使用zookeeper的zkCli.sh命令。

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

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

每个节点都需要添加。

第九步：启动每个solr实例。



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

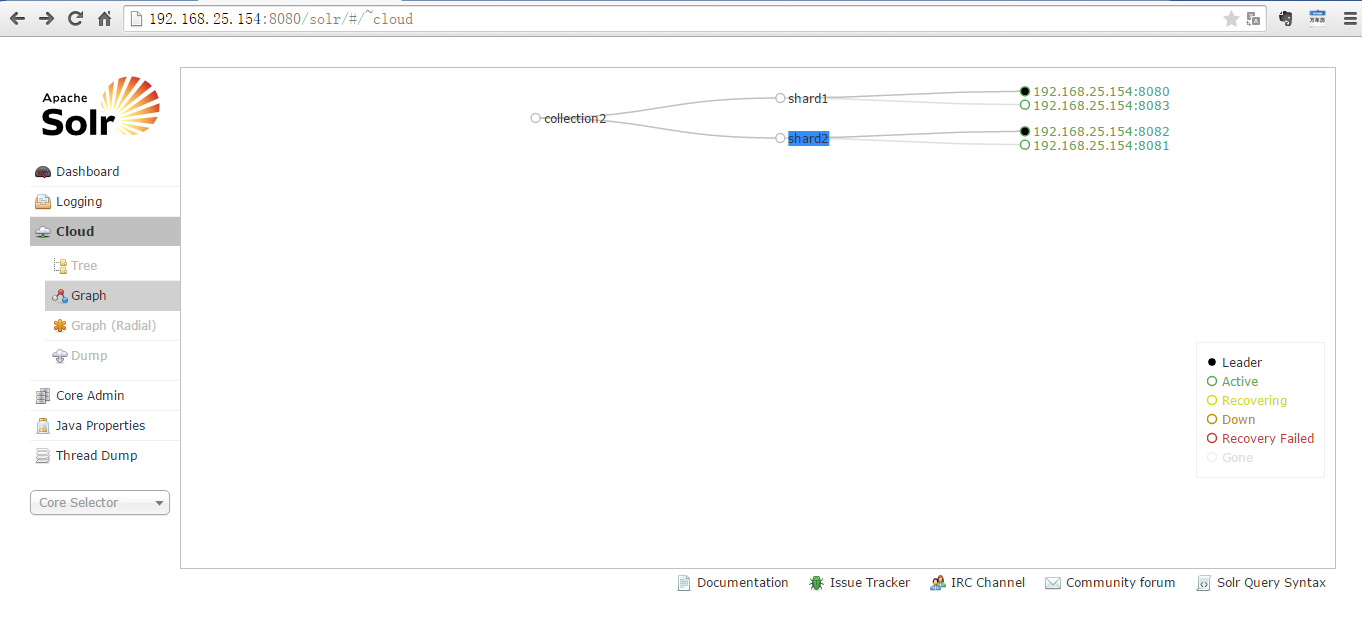




第十一步：删除不用collection1

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



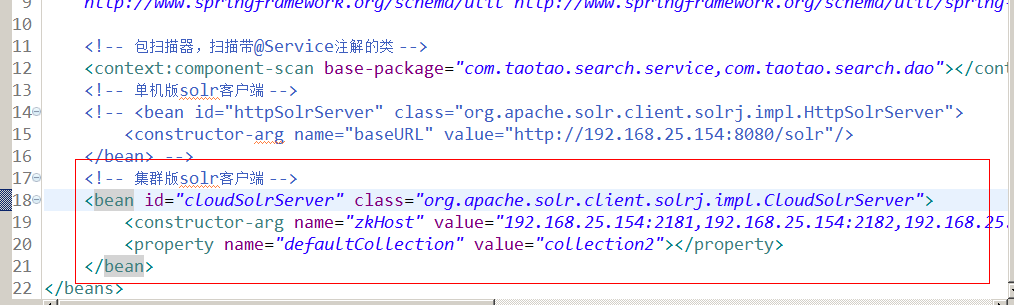


## 使用solrJ连接集群

|  |
| --- |
| @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();    } |

## 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。



# 商品同步

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。

作业！！