# 搜索引擎开发

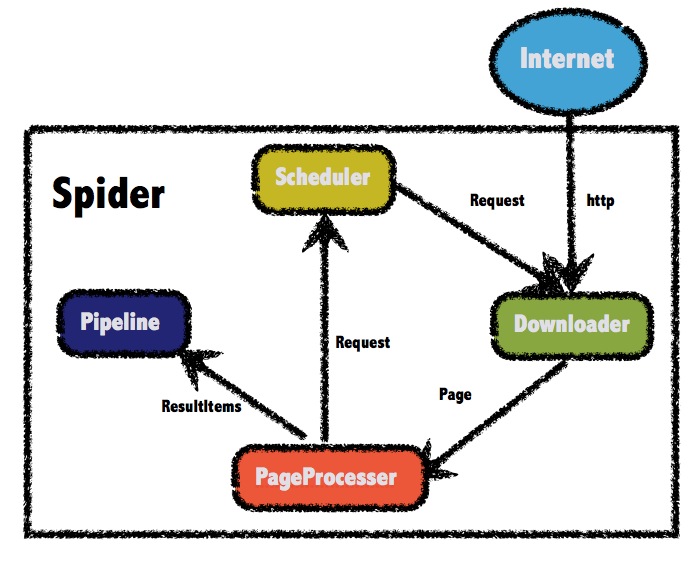
## 使用webmagic爬虫工具获取数据

webmagic的github网址：<https://github.com/code4craft/webmagic>

### 1、使用webmagic的原因

（1）webmagic是一个无须配置、便于二次开发的爬虫框架，它提供简单灵活的API，只需少量代码即可实现一个爬虫。

（2）webmagic采用完全模块化的设计，功能覆盖整个爬虫的生命周期(链接提取、页面下载、内容抽取、持久化)，支持多线程抓取，分布式抓取，url自动去重，并支持自动重试、自定义UA/cookie等功能。



Downloader负责从互联网上下载页面.

PageProcessor负责解析页面，抽取有用信息，以及发现新的链接。

Scheduler负责管理待抓取的URL，以及一些去重的工作。

Pipeline负责抽取结果的处理，包括计算、持久化到文件、数据库。

（3）webmagic包含强大的页面抽取功能，开发者可以便捷的使用css selector、xpath和正则表达式进行链接和内容的提取，支持多个选择器链式调用。

（4）webmagic也可以很方便的作为一个模块，嵌入Java项目中运行。

（5）文档相对齐全。

### 2、安装并配置webmagic

我们可以直接再maven项目中添加web magic的依赖来安装web magic即可

<dependency>

<groupId>us.codecraft</groupId>

<artifactId>webmagic-core</artifactId>

<version>0.5.3</version>

</dependency>

<dependency>

<groupId>us.codecraft</groupId>

<artifactId>webmagic-extension</artifactId>

<version>0.5.3</version>

</dependency>

### 3、编写相应爬虫程序

#### （1）定义目标数据的各个属性

我们的目标数据是古诗词，其属性有：类型、朝代、作者、作者简介、标题、原文、译文、注释、赏析。

对此，我们设计了Article这个模型类，来存储一首古诗词的这些属性：

**public** **class** Article {

/\*\*类型\*\*/

**private** String type;

/\*\*作者\*\*/

**private** String author;

/\*\*朝代\*\*/

**private** String dynasty;

/\*\*作者简介\*\*/

**private** String author\_info;

/\*\*标题\*\*/

**private** String title;

/\*\*原文\*\*/

**private** String content;

/\*\*译文\*\*/

**private** String translation;

/\*\*注释\*\*/

**private** String comment;

/\*\*赏析\*\*/

**private** String appreciation;

getter/setter...

**public** String toString(){

**return** "Article:{type="+type+",dynasty="+dynasty+",author="+author

+",author\_info="+author\_info+",title="+title+",content="

+content+",translation="+translation+",comment="+comment

+",appreciation="+appreciation+"}";

}

}

#### （2）根据目标网站设计相应的PageProcessor

我们这里爬取“好诗文网” <http://www.haoshiwen.org/type.php?> 中从先秦到清代的所有形式的古诗文。通过火狐浏览器的FIrebug工具，我们可以获取Article的各个属性所在的HTML位置，并设计出相应的正则表达式。因为我们需要从列表页获取到古诗文的详情页链接，而古诗文的详情页只包含朝代、作者、作者简介、原文、译文及注释的链接、赏析的链接，类型在列表页上，所以我设计了用articleType这个Map<String,String>（<古诗词的链接,古诗词的类型>）来保存列表页上的古诗文详情页的类型，用articleMap这个Map<String,Article>（<古诗词的链接,对应的Article对象>）来暂存古诗文详情页的各个信息，直到Article的各个信息都齐全了，才被保存到JSON文件中。

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** gushicisearch.model.Article;

**import** us.codecraft.webmagic.Page;

**import** us.codecraft.webmagic.Site;

**import** us.codecraft.webmagic.processor.PageProcessor;

**import** us.codecraft.webmagic.selector.Html;

**import** us.codecraft.webmagic.selector.Selectable;

**public** **class** GuShiWenPageProcessor **implements** PageProcessor{

/\*\*开始链接\*\*/

**private** **final** **static** String ***URL\_START*** =

"http://www\\.haoshiwen\\.org/type\\.php\\?"

+"c=\\d+&x=[1-5]$";

/\*\*列表链接\*\*/

**private** **final** **static** String ***URL\_LIST*** =

"http://www\\.haoshiwen\\.org/type\\.php\\?"

+"c=\\d+&x=[1-5]&page=\\d+";

/\*\*诗词链接\*\*/

**private** **final** **static** String ***URL\_ARTICLE*** =

"http://www\\.haoshiwen\\.org/view\\.php\\?id=\\d+";

/\*\*翻译链接\*\*/

**private** **final** **static** String ***URL\_TRANSLATION*** =

"http://www\\.haoshiwen\\.org/show\\.php\\?t=2&id=\\d+";

/\*\*赏析链接\*\*/

**private** **final** **static** String ***URL\_APPRECIATION*** =

"http://www\\.haoshiwen\\.org/show\\.php\\?t=1&id=\\d+";

/\*\*暂存Article\*\*/

**private** **static** Map<String,Article> *articleMap* =

**new** HashMap<String,Article>();

/\*\*暂存article的类型\*\*/

**private** **static** Map<String,String> *articleType* =

**new** HashMap<String,String>();

/\*\*获取article总数\*\*/

**public** **static** **int** articleCount(){

**return** *articleType*.size();

}

**private** Site site=Site.*me*().setCycleRetryTimes(5)

.setRetryTimes(5).setSleepTime(100)

.setUserAgent("Mo zilla/5.0 (Windows NT 6.1; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0")

.addHeader("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8")

.addHeader("Accept-Language", "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3")

.setCharset("UTF-8");

/\*\*

\* 获取article的类型

\* **@param** url 开始链接或列表链接，从中提取出article类型Num

\* **@return**

\*/

**private** **static** String getType(Selectable url){

String type=**null**;

**int** type\_num=Integer.*parseInt*(url.regex("c=\\d+&x=([1-5)])").toString());

**switch**(type\_num){

**case** 1:

type="诗";**break**;

**case** 2:

type="词";**break**;

**case** 3:

type="曲";**break**;

**case** 4:

type="文言文";**break**;

**case** 5:

type="辞赋";**break**;

}

**return** type;

}

/\*\*保存Article\*\*/

**private** **void** saveArticle(Article article,Page page){

page.putField("type", article.getType());

page.putField("dynasty", article.getDynasty());

page.putField("author", article.getAuthor());

page.putField("author\_info", article.getAuthor\_info());

page.putField("title", article.getTitle());

page.putField("content", article.getContent());

page.putField("translation", article.getTranslation());

page.putField("comment", article.getComment());

page.putField("appreciation", article.getAppreciation());

}

/\*\*提取想要的信息\*\*/

**public** **void** process(Page page){

**if**(page.getUrl().regex(***URL\_START***).match()){

//获取页数OK

String pageStr=page.getHtml()

.xpath("//div[@class='pages']")

.regex("/type.php\\?c=\\d+&amp;x=[1-5]&amp;page=(\\d+)\">尾页</a>")

.toString();

//System.out.println("开始："+page.getUrl()+" 页数"+pageStr);

**if**(pageStr!=**null**){

**int** page\_num=Integer.*parseInt*(pageStr);

//System.out.println(page\_num);

List<String> pageUrl=**new** ArrayList<String>();

//把其余页的url添加到Request队列中

**for**(**int** i=2;i<=page\_num;i++){

pageUrl.add(page.getUrl()+"&page="+i);

}

page.addTargetRequests(pageUrl);

}

//添加起始页的古诗文列表

List<String> articleUrl = page.getHtml()

.xpath("//div[@class='typeleft']/div[@class='sons']")

.regex(***URL\_ARTICLE***)

.all();

//System.out.println(articleUrl);

page.addTargetRequests(articleUrl);

page.setSkip(**true**);//跳过这个页面

//获取类型

String type=*getType*(page.getUrl());

**for**(String url: articleUrl){

*articleType*.put(url, type);

}

}

**if**(page.getUrl().regex(***URL\_LIST***).match()){

//System.out.println("列表："+page.getUrl());

//古诗文列表

List<String> articleUrl = page.getHtml()

.xpath("//div[@class='typeleft']/div[@class='sons']")

.regex(***URL\_ARTICLE***)

.all();

//System.out.println(articleUrl);

page.addTargetRequests(articleUrl);

page.setSkip(**true**);//跳过这个页面

//获取类型

String type=*getType*(page.getUrl());

**for**(String url: articleUrl){

*articleType*.put(url, type);

}

}

**else** **if**(page.getUrl().regex(***URL\_ARTICLE***).match()){

System.***out***.println("诗词："+page.getUrl());

Html html=page.getHtml();

Article article=**new** Article();

//类型

article.setType(*articleType*.get(page.getUrl().toString()));

//朝代

String dynasty=html

.xpath("//div[@class='son2']")

.regex("<span>朝代：</span>(.\*?)</p>").toString();

//System.out.println(dynasty);

article.setDynasty(dynasty);

//作者

String author=html

.xpath("//div[@class='son2']")

.regex("<span>作者：</span>(.\*?)</p>")

.toString().replaceAll("</?a.\*?>", "");

//System.out.println(author);

article.setAuthor(author);

**if**(!author.equals("佚名")){

//作者简介

String author\_info=html

.regex("<div class=\"son5\" style=\"overflow:auto;\">"

+ ".\*</a> ("+author+".\*?)<a.\*?>\\.\\.\\.</a>")

.toString();

//System.out.println(author\_info);

article.setAuthor\_info(author\_info);

}

//标题

String title=html.xpath("div[@class='son1']/h1/text()")

.toString();

//System.out.println(title);

article.setTitle(title);

//原文

String content=html

.xpath("div[@class='son2']")

.regex("<span>原文：</span></p>(.\*?)</div>")

.toString();

//System.out.println(content);

article.setContent(content);

//译文链接

String fanyi\_url=html

.xpath("div[@class='son5']").links()

.regex(***URL\_TRANSLATION***).toString();

//赏析链接

String shangxi\_url=html

.xpath("div[@class='son5']").links()

.regex(***URL\_APPRECIATION***).toString();

//System.out.println("翻译："+fanyi\_url);

//System.out.println("赏析："+shangxi\_url);

**if**(fanyi\_url==**null**&&shangxi\_url==**null**){

//如果没有译文和赏析，则直接保存该Article对象

saveArticle(article,page);

}

**else**{

//否则，则把Article存在articleMap，等待信息被补齐才保存

**if**(fanyi\_url!=**null**){

article.setTranslation(fanyi\_url);

page.addTargetRequest(fanyi\_url);

}

**if**(shangxi\_url!=**null**){

article.setAppreciation(shangxi\_url);

page.addTargetRequest(shangxi\_url);

}

*articleMap*.put(page.getUrl().toString(), article);

page.setSkip(**true**);//跳过这个页面

}

}

**else** **if**(page.getUrl().regex(***URL\_TRANSLATION***).match()){

Html html=page.getHtml();

String article\_url=html

.xpath("//div[@class='sontitle']/span/a/@href")

.toString();

//System.out.println("诗词的链接"+article\_url);

//翻译标题

String tran\_title=html

.xpath("//div[@class='shileft']/div[@class='son1']/h1/text()")

.toString();

//System.out.println(tran\_title);

String translation=**null**;

String comment=**null**;

**if**(tran\_title.endsWith("译文及注释")){

translation=html

.xpath("//div[@class='shangxicont']")

.regex("<p><strong>译文.\*?</strong>(.\*?)</p>")

.toString();

**if**(translation!=**null**)//去掉无关内容

translation=translation.replaceAll("</?a.\*?>", "");

comment=html

.xpath("//div[@class='shangxicont']")

.regex("<p><strong>注释.\*?</strong>(.\*?)</p>")

.toString();

**if**(comment!=**null**)

comment=comment.replaceAll("</?a.\*?>", "");

**if**(translation==**null**&&comment==**null**){

//译文和注释被合并在了一起

translation=html

.xpath("//div[@class='shangxicont']")

.regex("<p>作者：佚名</p>(.\*?)<p style=")

.toString();

**if**(translation!=**null**)

translation=translation.replaceAll("</?a.\*?>", "");

}

}**else**{

//只有译文

**if**(tran\_title.endsWith("译文")){

translation=html

.xpath("//div[@class='shangxicont']")

.regex("<p>作者：佚名</p>(.\*?)<p style=")

.toString();

**if**(translation!=**null**)//去掉无关内容

translation=translation.replaceAll("</?a.\*?>", "");

}

//只有注释

**if**(tran\_title.endsWith("注释")){

comment=html

.xpath("//div[@class='shangxicont']")

.regex("<p>作者：佚名</p>(.\*?)<p style=")

.toString();

**if**(comment!=**null**)

comment=comment.replaceAll("</?a.\*?>", "");

}

}

//System.out.println(translation);

//System.out.println(comment);

Article article=*articleMap*.get(article\_url);

String appreciation=article.getAppreciation();

**if**(appreciation!=**null**&&appreciation.startsWith("http")){

*articleMap*.get(article\_url).setTranslation(translation);

*articleMap*.get(article\_url).setComment(comment);;

page.setSkip(**true**);//跳过这个页面

}**else**{

article.setTranslation(translation);

article.setComment(comment);

saveArticle(article,page);

*articleMap*.remove(article\_url);

}

}

**else** **if**(page.getUrl().regex(***URL\_APPRECIATION***).match()){

Html html=page.getHtml();

String article\_url=html

.xpath("//div[@class='sontitle']/span/a/@href")

.toString();

//System.out.println("诗词的链接"+article\_url);

String appre\_title=html

.xpath("//div[@class='shileft']/div[@class='son1']/h1")

.toString();

//System.out.println(appre\_title);

String appreciation=html

.xpath("//div[@class='shangxicont']")

.regex("<p>作者：佚名</p>(.\*?)<p style=")

.toString();

**if**(appreciation!=**null**)

appreciation=appreciation.replaceAll("</?a.\*?>", "");

//System.out.println(appreciation);

Article article=*articleMap*.get(article\_url);

String translation=article.getTranslation();

**if**(translation!=**null**&&translation.startsWith("http")){

*articleMap*.get(article\_url).setAppreciation(appre\_title+appreciation);

page.setSkip(**true**);//跳过这个页面

}**else**{

article.setAppreciation(appre\_title+appreciation);

saveArticle(article,page);

*articleMap*.remove(article\_url);

}

}

}

**public** Site getSite() {

**return** site;

}

}

#### 运行多线程爬虫

因为elasticsearch的数据类型是JSON格式的，所以我们把获取到的古诗文数据都保存成JSON格式。当然，要保存到数据库MySQL中，然后再从MySQL中将数据导入elastic search也不是不行，但是因为我们只是做个简单的搜索系统，所以就不弄数据库了。

**import** us.codecraft.webmagic.Spider;

**import** us.codecraft.webmagic.pipeline.JsonFilePipeline;

**public** **class** GuShiCiCrawler {

/\*\*网址头部\*\*/

**private** **final** **static** String ***headUrl*** =

"http://www.haoshiwen.org/type.php?";

/\*\*初始化开始爬取的链接\*\*/

**private** **static** String[] intiUrls(){

String[] urls=**new** String[55];

**int** count=0;

**for**(**int** i=1;i<=11;i++){

**for**(**int** j=1;j<=5;j++){

urls[count++]=***headUrl***+"c="+i+"&x="+j;

}

}

**return** urls;

}

**public** **static** **void** main(String[] args){

Spider.*create*(**new** GuShiCiPageProcessor())//指定PageProcessor页面处理器

.addUrl(*intiUrls*())//添加爬取链接

//指定Pipeline结果处理对象，这里把结果保存成JSON文件

.addPipeline(**new** JsonFilePipeline())

.thread(5)//指定线程数

.run();//开始爬虫

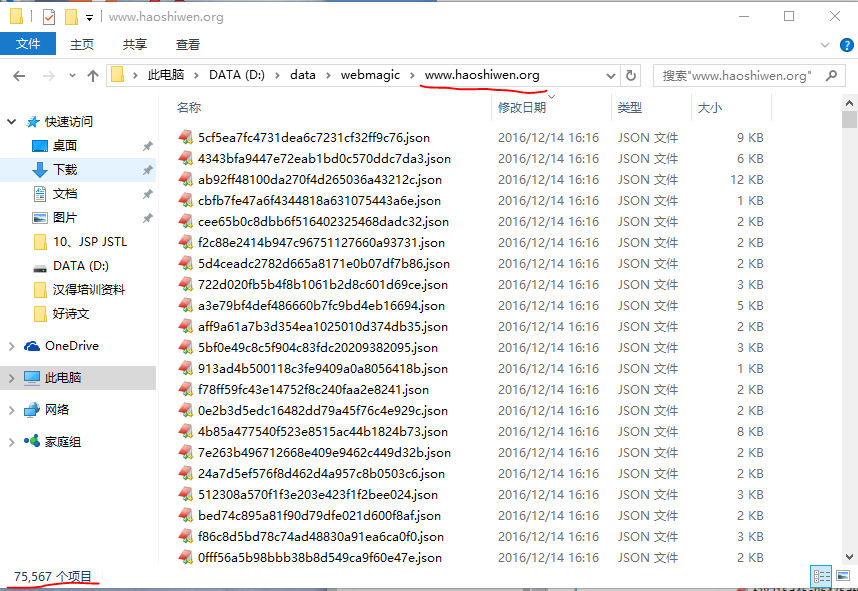
System.***out***.println("运行结束");

}

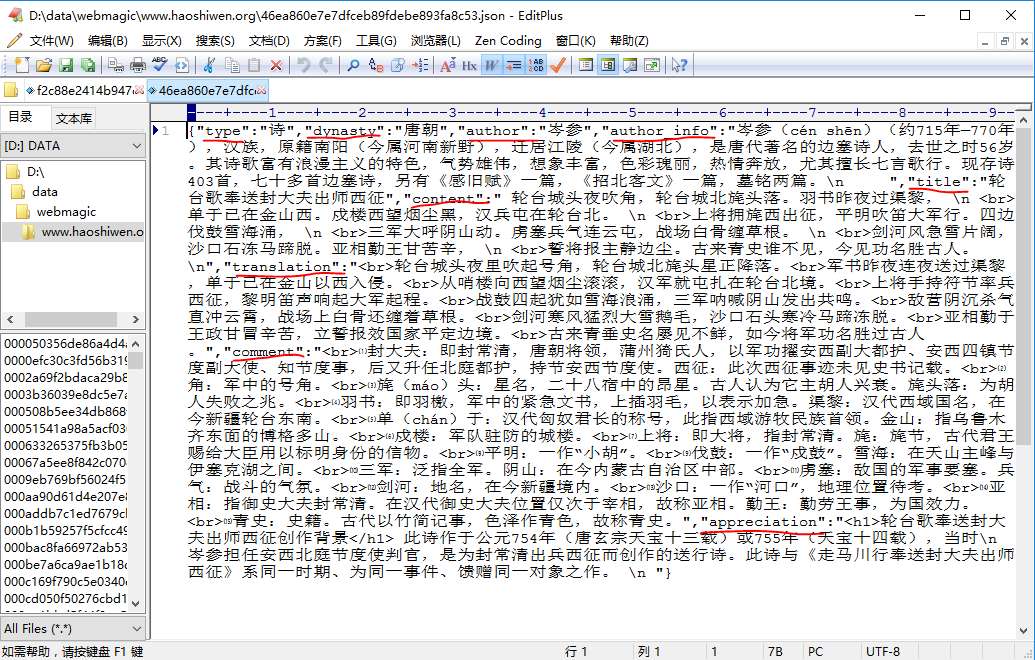
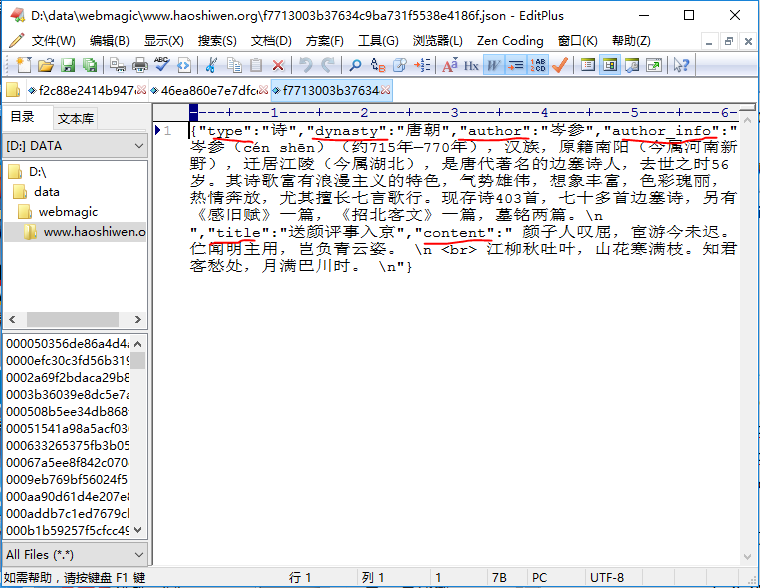
}

点击运行这个类，我们的爬虫就开始爬取并保存“好诗文网”中我们想要的古诗文数据了，数据被保存在项目路径/data/webmagic/网址下。

“好诗文网”共有诗文75837篇，我们的爬虫运行了大概7小时，总共爬到了75567篇（有点误差，但数据量实在太大，也查不出是哪里的问题）。



打开某个JSON文件，查看其内容：



可以看到，Article的各个属性基本上都被获取到了。

## 使用elasticsearch建立搜索引擎

elasticsearch官方网址：<https://www.elastic.co/products/elasticsearch>

### 1、选择Elasticsearch的原因

（1）Elasticsearch是一个建立在全文搜索引擎 Apache Lucene™ 基础上的开源的实时分布式搜索和分析引擎，功能强大：

* 支持全文搜索；
* 分布式实时文件存储，并将每一个字段都编入索引，使其可以被搜索；
* 实时分析的分布式搜索引擎；
* 可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。

所有的这些功能被集成到一个服务里面，你的应用可以通过简单的RESTful API、各种语言的客户端甚至命令行与之交互。

1. 简单易学，文档齐全

搜索引擎选择： Elasticsearch与Solr：<http://www.cnblogs.com/chowmin/articles/4629220.html>

### 2、安装并配置Elasticsearch

因为我们要使用ansj分词工具进行分词，最新的ansj与elasticsearch结合的工具包对应的elastic search的版本是5.0.1，所以我们下载5.0.1版本的elasticsearch。

**（1）下载并解压**

wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.0.1.tar.gz

sha1sum elasticsearch-5.0.1.tar.gz

tar -xzf elasticsearch-5.0.1.tar.gz

cd elasticsearch-5.0.1/

1. 启动ES

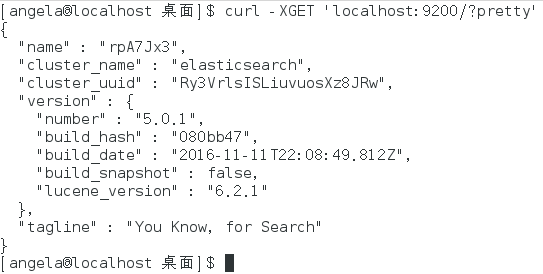
./bin/elasticsearch

16-12-11T17:28:33,912][INFO ][o.e.n.Node ] [rpA7Jx3] started

看到类似这一句的，则说明启动ES了

新开一个终端，查看是否运行成功

curl -XGET 'localhost:9200/?pretty'



出现如上形式内容，则说明ES运行成功。

可以按Ctrl-C关闭ES

### 3、安装并配置ansj分词器

进入es目录执行如下命令

./bin/elasticsearch-plugin install http://maven.nlpcn.org/org/ansj/elasticsearch-analysis-ansj/5.0.1.0/elasticsearch-analysis-ansj-5.0.1.0-release.zip

### 4、elasticsearch启动出现的错误解决

1. Java HotSpot(TM) 64-Bit Server VM warning: INFO:

os::commit\_memory(0x0000000085330000, 2060255232, 0) failed; error='Cannot allocate memory' (errno=12)

由于elasticsearch5.0默认分配jvm空间大小为2g，修改jvm空间分配

# vim config/jvm.options

-Xms2g

-Xmx2g

修改为

-Xms512m

-Xmx512m

（2）max number of threads [1024] for user [elasticsearch] is too low, increase to at least [2048]

修改 /etc/security/limits.d/90-nproc.conf

\* soft nproc 1024

\* soft nproc 2048

（3）max virtual memory areas vm.max\_map\_count [65530] is too low, increase to at least [262144]

修改/etc/sysctl.conf配置文件，

cat /etc/sysctl.conf | grep vm.max\_map\_count

vm.max\_map\_count=262144

如果不存在则添加

echo "vm.max\_map\_count=262144" >>/etc/sysctl.conf

（4）max file descriptors [65535] for elasticsearch process likely too low, increase to at least [65536]

ulimit -n 65536

（5）[root@localhost elasticsearch-5.0.1]# ./bin/elasticsearch

[WARN ][o.e.b.ElasticsearchUncaughtExceptionHandler] [] uncaught exception in thread [main]

org.elasticsearch.bootstrap.StartupException: java.lang.RuntimeException: **can not run elasticsearch as root**

注意：ES不能用root管理员身份启动

### 5、配置elasticsearch Java API

在pom.xml添加如下依赖：

<!-- elasticsearch Java API -->

<dependency>

<groupId>org.elasticsearch.client</groupId>

<artifactId>transport</artifactId>

<version>5.0.1</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.8.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.8.0</version>

</dependency>

### elasticsearch教程

官方教程：<https://www.elastic.co/guide/en/elasticsearch/reference/current/zip-targz.html>

Elasticsearch基础教程：<http://blog.csdn.net/cnweike/article/details/33736429>

Elasticsearch JAVA API教程：<http://www.07net01.com/2016/07/1603264.html>

**1.Java API批量导出**

Settings settings = ImmutableSettings.settingsBuilder().put("cluster.name", "elasticsearch-bigdata").build();

Client client = new TransportClient(settings)

.addTransportAddress(new InetSocketTransportAddress("10.58.71.6", 9300));

SearchResponse response = client.prepareSearch("bigdata").setTypes("student")

.setQuery(QueryBuilders.matchAllQuery()).setSize(10000).setScroll(new TimeValue(6000 00))

.setSearchType(SearchType.SCAN).execute().actionGet();//setSearchType(SearchType.Scan) 告诉ES不需要排序只要结果返回即可 setScroll(new TimeValue(600000)) 设置滚动的时间

String scrollid = response.getScrollId();

try {

//把导出的结果以JSON的格式写到文件里

BufferedWriter out = new BufferedWriter(new FileWriter("es", true));

//每次返回数据10000条。一直循环查询直到所有的数据都查询出来

while (true) {

SearchResponse response2 = client.prepareSearchScroll(scrollid).setScroll(new TimeValue(1000000))

.execute().actionGet();

SearchHits searchHit = response2.getHits();

//再次查询不到数据时跳出循环

if (searchHit.getHits().length == 0) {

break;

}

System.out.println("查询数量 ：" + searchHit.getHits().length);

for (int i = 0; i < searchHit.getHits().length; i++) {

String json = searchHit.getHits()[i].getSourceAsString();

out.write(json);

out.write("\r\n");

}

}

System.out.println("查询结束");

out.close();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

**2.Java API 批量导入**

Settings settings = ImmutableSettings.settingsBuilder().put("cluster.name", "elasticsearch-bigdata").build();

Client client = new TransportClient(settings)

.addTransportAddress(new InetSocketTransportAddress("10.58.71.6", 9300));

try {

//读取刚才导出的ES数据

BufferedReader br = new BufferedReader(new FileReader("es"));

String json = null;

int count = 0;

//开启批量插入

BulkRequestBuilder bulkRequest = client.prepareBulk();

while ((json = br.readLine()) != null) {

bulkRequest.add(client.prepareIndex("bigdata", "student").setSource(json));

//每一千条提交一次

if (count% 1000==0) {

bulkRequest.execute().actionGet();

System.out.println("提交了：" + count);

}

count++;

}

bulkRequest.execute().actionGet();

System.out.println("插入完毕");

br.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

**elasticsearch导入数据的两种方法**

第一种方法：手动导入

1、cat test.json

{"index":{"\_index":"stuff\_orders","\_type":"order\_list","\_id":903713}}

{"real\_name":"刘备","user\_id":48430,"address\_province":"上海","address\_city":"浦东新区","address\_district":null,"address\_street":"上海市浦东新区广兰路1弄2号345室","price":30.0,"carriage":6.0,"state":"canceled","created\_at":"2013-10-24T09:09:28.000Z","payed\_at":null,"goods":["营养早餐：火腿麦满分"],"position":[121.53,31.22],"weight":70.0,"height":172.0,"sex\_type":"female","birthday":"1988-01-01"}

2、导入elasticsearch

[root@ELKServer opt]# curl -XPOST ‘localhost:9200/stuff\_orders/\_bulk?pretty‘ --data-binary @test.json

{

  "took" : 600,

  "errors" : false,

  "items" : [ {

    "index" : {

      "\_index" : "stuff\_orders",

      "\_type" : "order\_list",

      "\_id" : "903713",

      "\_version" : 1,

      "\_shards" : {

        "total" : 2,

        "successful" : 1,

        "failed" : 0

      },

      "status" : 201

    }

  } ]

}

3、查看elasticsearch是否存在数据

[root@ELKServer opt]# curl localhost:9200/stuff\_orders/order\_list/903713?pretty

{

  "\_index" : "stuff\_orders",

  "\_type" : "order\_list",

  "\_id" : "903713",

  "\_version" : 1,

  "found" : true,

  "\_source" : {

    "real\_name" : "刘备",

    "user\_id" : 48430,

    "address\_province" : "上海",

    "address\_city" : "浦东新区",

    "address\_district" : null,

    "address\_street" : "上海市浦东新区广兰路1弄2号345室",

    "price" : 30.0,

    "carriage" : 6.0,

    "state" : "canceled",

    "created\_at" : "2013-10-24T09:09:28.000Z",

    "payed\_at" : null,

    "goods" : [ "营养早餐：火腿麦满分" ],

    "position" : [ 121.53, 31.22 ],

    "weight" : 70.0,

    "height" : 172.0,

    "sex\_type" : "female",

    "birthday" : "1988-01-01"

  }

}

第二种方法：从数据库中导入

1、下载安装插件elasticsearch-jdbc-2.3.4.0

wget http://xbib.org/repository/org/xbib/elasticsearch/importer/elasticsearch-jdbc/2.3.4.0/elasticsearch-jdbc-2.3.4.0-dist.zip

elasticsearch-jdbc-2.3.4.0-dist.zip的版本要和你安装的elasticsearch对应。

unzip elasticsearch-jdbc-2.3.4.0-dist.zip

mv elasticsearch-jdbc-2.3.4.0 /usr/local/

cd /usr/local/elasticsearch-jdbc-2.3.4.0/

2、配置脚本

vim import.sh

#!/bin/sh

JDBC\_IMPORTER\_HOME=/usr/local/elasticsearch-jdbc-2.3.4.0

bin=$JDBC\_IMPORTER\_HOME/bin

lib=$JDBC\_IMPORTER\_HOME/lib

echo ‘{

"type" : "jdbc",

"jdbc": {

"elasticsearch.autodiscover":true,

"elasticsearch.cluster":"my-application", #簇名 详见：/usr/local/elasticsearch/config/elasticsearch.yml

"url":"jdbc:mysql://localhost:3306/test",  #mysql数据库地址

"user":"test",  #mysql用户名

"password":"1234",  #mysql密码

"sql":"select \*,id as \_id from workers\_info",

"elasticsearch" : {

  "host" : "192.168.10.49",

  "port" : 9300

},

"index" : "myindex",  #新的index

"type" : "mytype"  #新的type

}

}‘| java   -cp "${lib}/\*"   -Dlog4j.configurationFile=${bin}/log4j2.xml   org.xbib.tools.Runner   org.xbib.tools.JDBCImporter

chmod + import.sh

sh import.sh

3、查看数据是否导入elasticsearch

[root@ELKServer bin]# curl -XGET ‘http://localhost:9200/myindex/mytype/\_search?pretty‘

{

  "took" : 15,

  "timed\_out" : false,

  "\_shards" : {

    "total" : 5,

    "successful" : 5,

    "failed" : 0

  },

  "hits" : {

    "total" : 1,

    "max\_score" : 1.0,

    "hits" : [ {

      "\_index" : "myindex",

      "\_type" : "mytype",

      "\_id" : "AVZyXCReGHjmX33dpJi3",

      "\_score" : 1.0,

      "\_source" : {

        "id" : 1,

        "workername" : "xing",

        "salary" : 10000,

        "tel" : "1598232123",

        "mailbox" : "xing@qq.com",

        "department" : "yanfa",

        "sex" : "F",

        "qq" : 736019646,

        "EmployedDates" : "2012-12-21T00:00:00.000+08:00"

      }

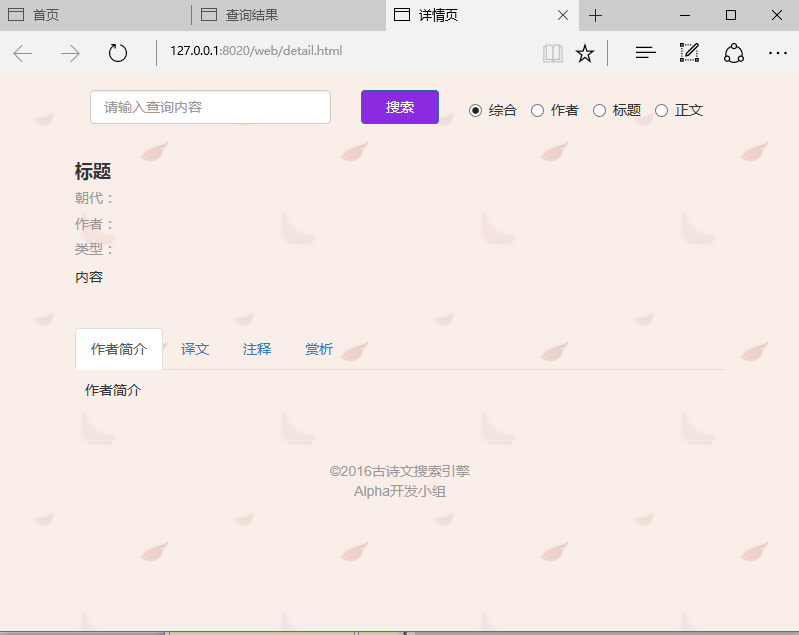
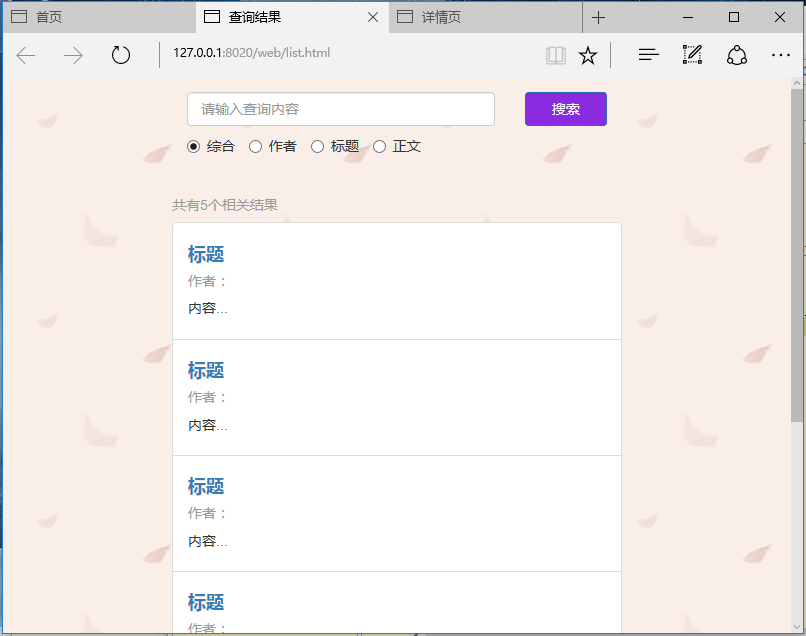
    } ]

  }

}

[](http://s2.51cto.com/wyfs02/M00/86/21/wKioL1e1dCCB5tn7AABBoLN-zp4909.png" \t "http://www.mamicode.com/_blank)

## 设计前端展示页面

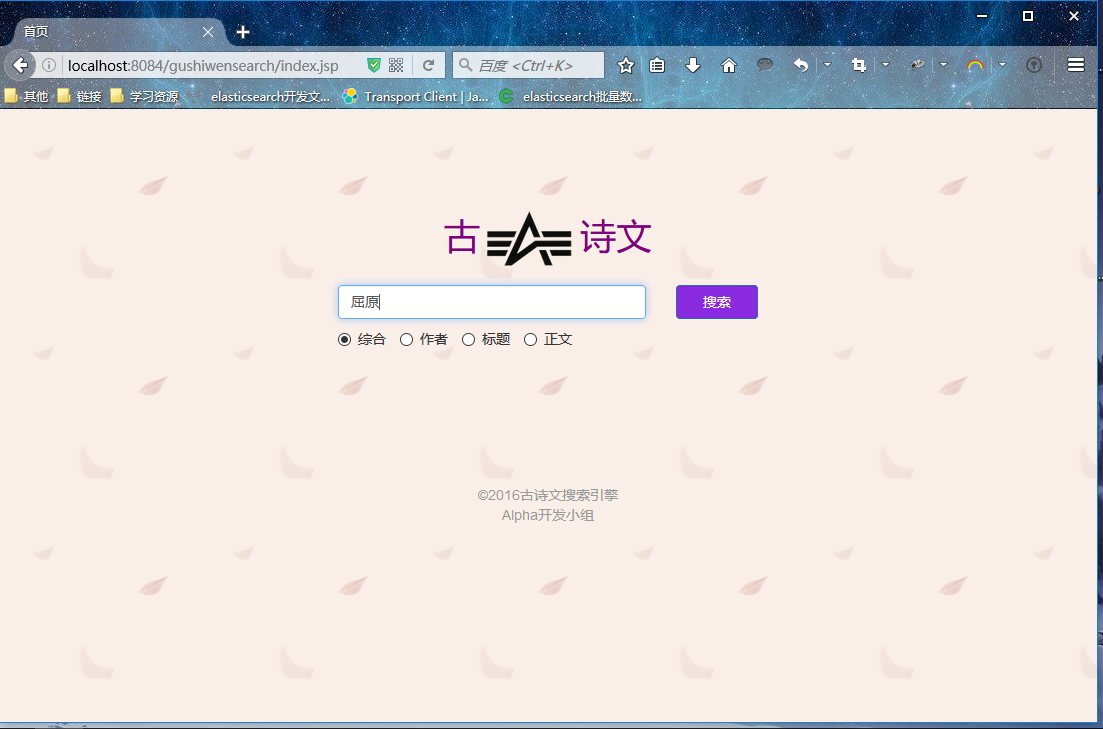


## 系统使用教程

### 启动ES

Windows系统，在ES目录的bin目录中打开命令行窗口，输入命令：elasticsearch.bat，回车，如果看到ES集群显示started并且状态为Green，则说明启动成功

### 使用浏览器打开首页



输入要查询的内容，并选择搜索范围，点击搜索按钮



可看到匹配的结果数和各个结果的部分信息。

点击某篇诗文的链接，就可以查看诗文的详细信息了。

