## 启动nginx和fastDfs来当做图片服务器。

## nginx的启动

cd /usr/local/nginx/sbin

./nginx //启动

,/nginx -s stop //停止

./nginx -s reloadt //刷新配置

## FastDFS的启动

1. 启动tracker服务

/usr/bin/fdfs\_trackerd /etc/fdfs/tracker.conf

重启

/usr/bin/fdfs\_trackerd /etc/fdfs/tracker.conf

1. 启动storage服务。

/usr/bin/fdfs\_storaged /etc/fdfs/storage.conf

重启

/usr/bin/fdfs\_storaged /etc/fdfs/storage.conf restart

# 根据进程名来查看进程是否启动的命令

ps aux|grep name

# spring和springMVC的父子容器的概念

1. 子容器可以访问父容器的对象，父容器无法访问子容器的对象;即spring是父容器，springMvc是子容器。在spring的对象中无法装配spirngmvc所扫描装配的controller对象。
2. 当spring和springmvc所扫描的范围是一样时。Spring和springMvc的容器中都会装载这些对象。但是会各不相同。有可能导致spring配置的事务在springmvc中失效的情况。

Springmvc（前端控制器）（子容器）

Controller对象

Spring容器（父容器）

1. Mapper代理对象
2. Service对象
3. 注意：在spring和springmvc中配置的**属性**不能跨容器调用.

# Spring中获取资源文件中的值

使用@value注解

|  |
| --- |
| @value(“${属性名}”) |

注意:

前提是在spring的配置文件中必须要有加载配置文件的配置。

|  |
| --- |
| <context:property-placeholder location=”classpath:properties/\*.poreties /> |

# Linux 复制文件

|  |
| --- |
| cp 源文件名 目标路径 |

# Linux vim快速查询和取消查询

查询

|  |
| --- |
| :查询的关键字 |

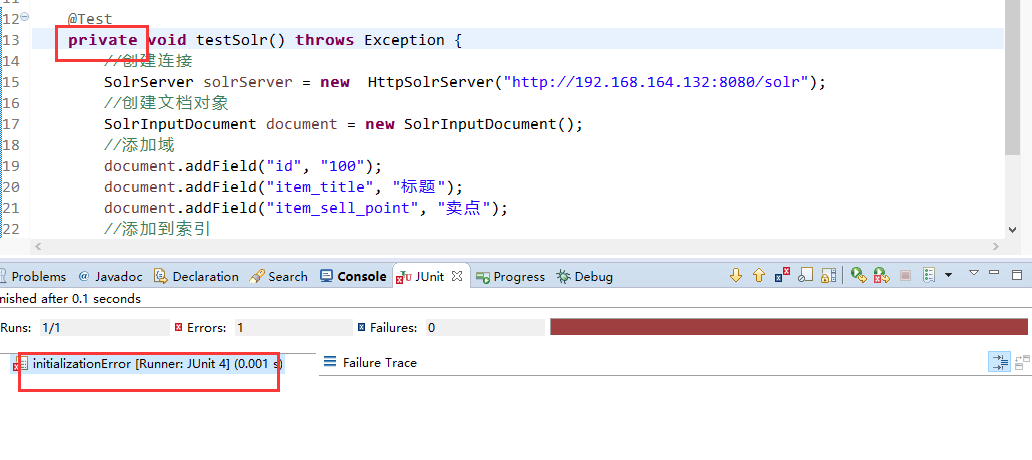
取消查询 即取消查询的高亮

|  |
| --- |
| :noh |

# 第八天笔记

## Junit笔记

如果执行的方法是private类型 将会执行不了，并会报initializationError



## Solr安装

### Solr安装 以centos 6.5 为例

第一步：安装jdk、安装tomcat

第二步：解压solr压缩包。

命令为 tar -zxf solr-4.10.3.tgz.tgz

第三步：把dist/solr-4.10.3.war部署到tomcat下。

|  |
| --- |
| cp solr-4.10.3.war /usr/local/solr/tomcat/webapps/solr.war -r |

第四步：解压缩war包。启动tomcat解压。

即启动Tomcat再关闭

|  |
| --- |
| [root@localhost tomcat]# ./bin/startup.sh |

再关闭tomcat 就解压完成

第五步：需要把/root/solr-4.10.3/example/lib/ext目录下的所有的jar包添加到solr工程中。

|  |
| --- |
| [root@localhost ext]# cp \* /usr/local/solr/tomcat/webapps/solr/WEB-INF/lib/ |

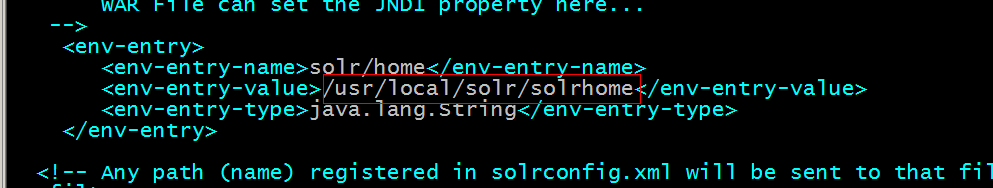
第六步：创建solrhome。把/root/solr-4.10.3/example/solr文件夹复制一份作为solrhome。

|  |
| --- |
| [root@localhost solr]# cp -r solr /usr/local/solr/solrhome |

第七步：告诉solr服务solrhome的位置。需要修改web.xml

web.xml在/usr/local/solr/tomcat/webapps/solr/WEB-INF/web.xml

修改结果为

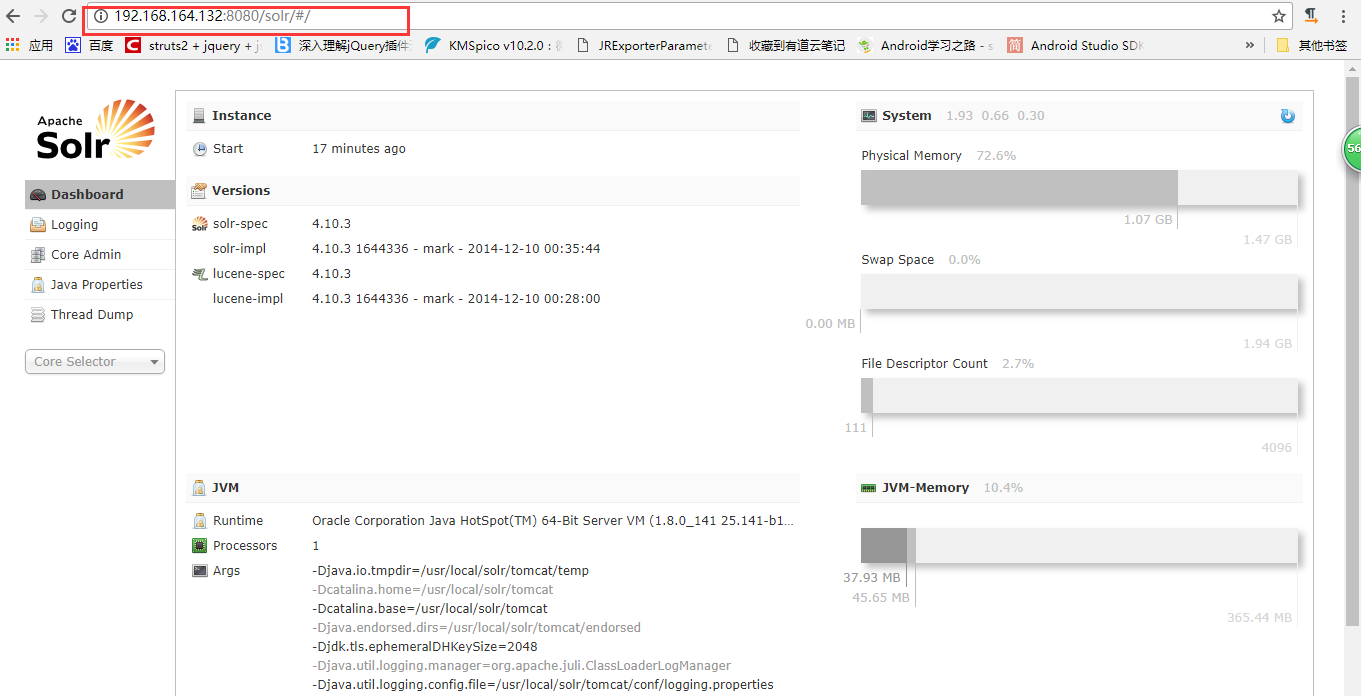


第八步：启动tomcat。

查看tomcat输出日志命令

|  |
| --- |
| tail -f logs/catalina.out |

最终结果：即安装成功



### solr配置中文分析器、自定义业务域

分析器使用IKAnalyzer。

使用方法：

第一步：把IKAnalyzer依赖的jar包添加到solr工程中。把分析器使用的扩展词典添加到classpath中。

|  |
| --- |
| [root@localhost IK Analyzer 2012FF\_hf1]# cp IKAnalyzer2012FF\_u1.jar /usr/local/solr/tomcat/webapps/solr/WEB-INF/lib/ |
| [root@localhost IK Analyzer 2012FF\_hf1]# cp ext\_stopword.dic IKAnalyzer.cfg.xml mydict.dic /usr/local/solr/tomcat/webapps/solr/WEB-INF/classes |

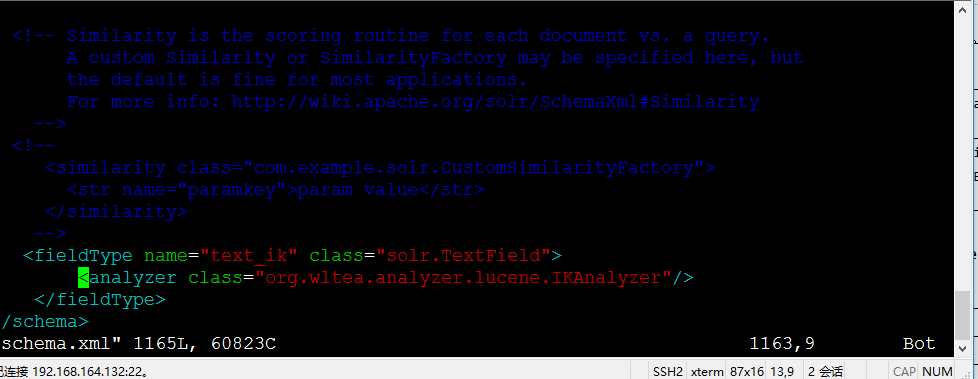
第二步：需要自定义一个FieldType。Schema.xml中定义。可以在FieldType中指定中文分析器。

地址：

|  |
| --- |
| /usr/local/solr/solrhome/collection1/conf/schema.xml |

在schema.xml的底部添加如下配置

|  |
| --- |
| <fieldType name="text\_ik" class="solr.TextField">  <analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>  </fieldType> |



即完成中文分析器的配置

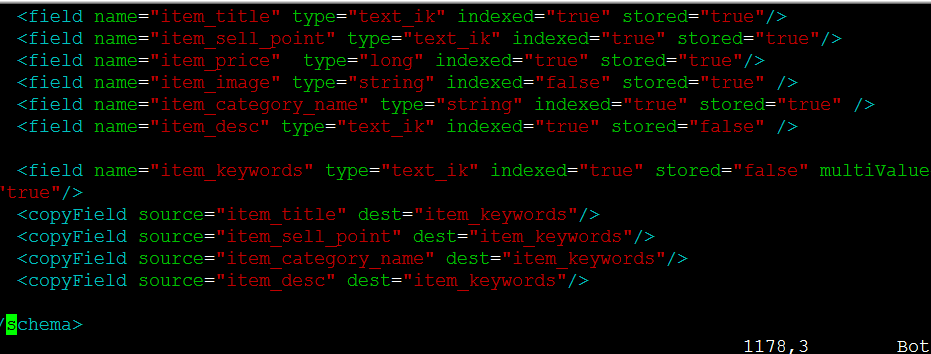
第三步：自定义域。指定域的类型为自定义的FieldType。

Sql语句：

|  |
| --- |
| SELECT  a.id,  a.title,  a.sell\_point,  a.price,  a.image,  b.`name` category\_name,  c.item\_desc  FROM  tb\_item a  LEFT JOIN tb\_item\_cat b ON a.cid = b.id  LEFT JOIN tb\_item\_desc c ON a.id = c.item\_id  WHERE  a.`status` = 1 |

将如下配置配置到schema.xml中完成solr的自定义域的配置

|  |
| --- |
| <field name="item\_title" type="text\_ik" indexed="true" stored="true"/>  <field name="item\_sell\_point" type="text\_ik" indexed="true" stored="true"/>  <field name="item\_price" type="long" indexed="true" stored="true"/>  <field name="item\_image" type="string" indexed="false" stored="true" />  <field name="item\_category\_name" type="string" indexed="true" stored="true" />  <field name="item\_desc" type="text\_ik" indexed="true" stored="false" />  <field name="item\_keywords" type="text\_ik" indexed="true" stored="false" multiValued="true"/>  <copyField source="item\_title" dest="item\_keywords"/>  <copyField source="item\_sell\_point" dest="item\_keywords"/>  <copyField source="item\_category\_name" dest="item\_keywords"/>  <copyField source="item\_desc" dest="item\_keywords"/> |



第四步：重新启动tomcat

## 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

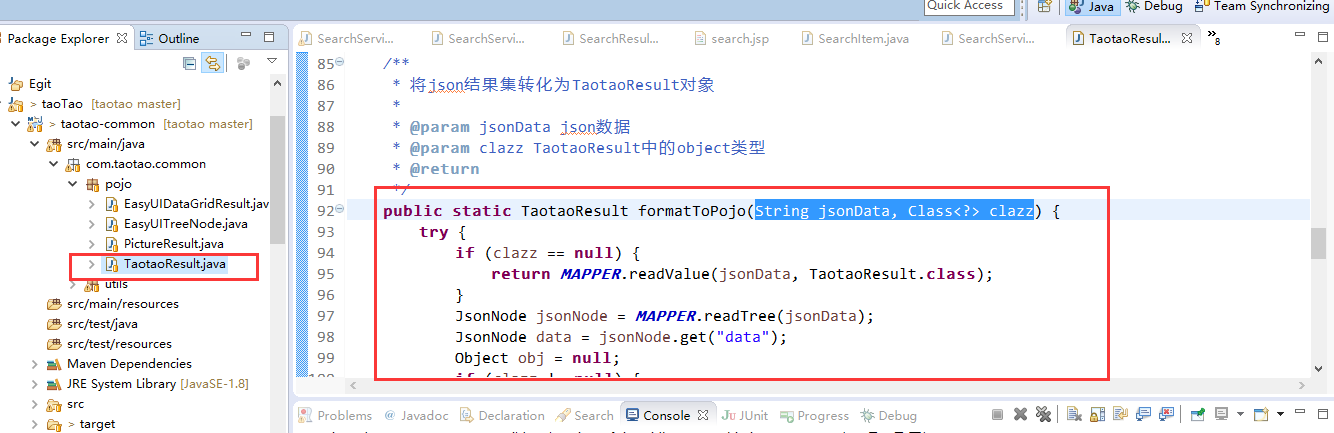
2、查询结果总记录数

3、查询结果的总页数

4、当前页码

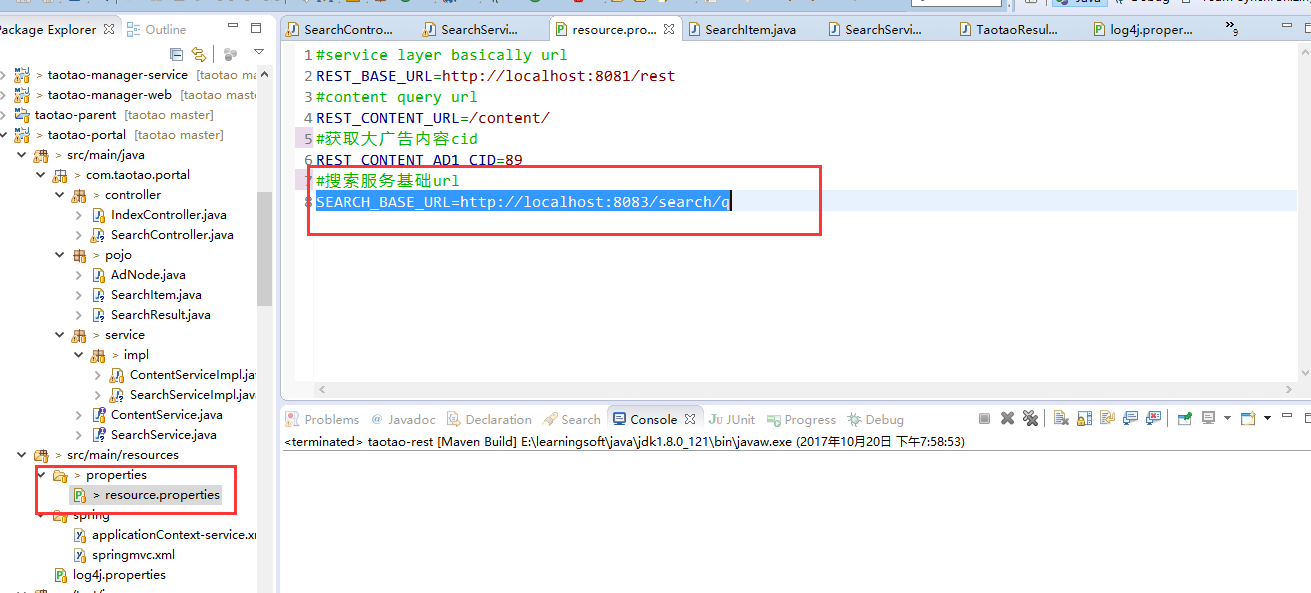
### 注意点

1. json转taotaoResult使用TaotaoResult.formatToPojo(String jsonData, Class<?> clazz;



1. 使用HttpClient向search服务请求时注意url的正确性。

SEARCH\_BASE\_URL=http://localhost:8083/search/q

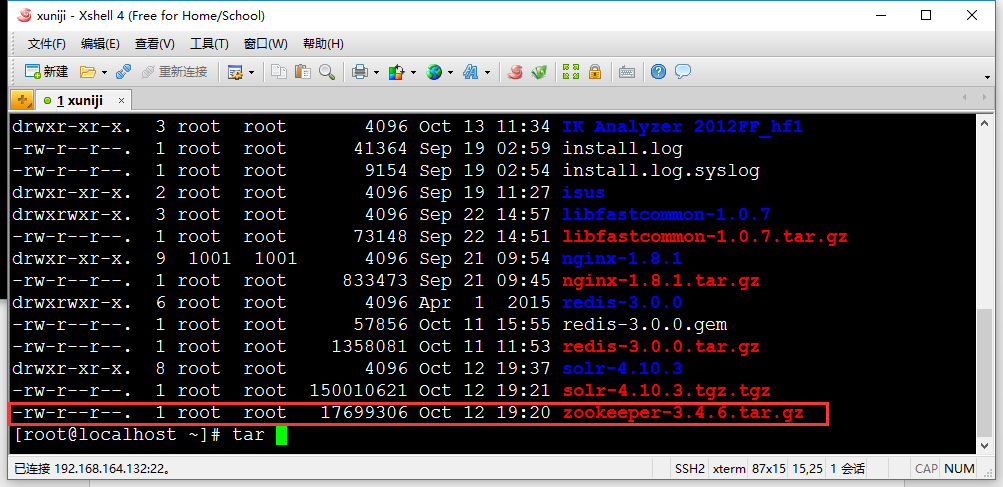


4.HttpClient请求带参数的方式为

|  |
| --- |
| //使用HttpClientUtil 向search工程请求数据 返回类型为json字符串  Map<String, String> param = **new** HashMap<>();  param.put("keyword", keyword);  param.put("page", page + "");  param.put("rows", rows + "");  String json = HttpClientUtil.*doGet*(SEARCH\_BASE\_URL,param); |

## Zookeeper安装手册

第一步：需要把zookeeper的安装包上传到服务器。



第二步：把zookeeper解压。

|  |
| --- |
| tar -zxf zookeeper-3.4.6.tar.gz |

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

|  |
| --- |
| [root@localhost ~]# mkdir /usr/local/solr-cloud  [root@localhost ~]# cp -r zookeeper-3.4.6 /usr/local/solr-cloud/zookeeper01  [root@localhost ~]# cp -r zookeeper-3.4.6 /usr/local/solr-cloud/zookeeper02  [root@localhost ~]# cp -r zookeeper-3.4.6 /usr/local/solr-cloud/zookeeper03 |

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”）

|  |
| --- |
| echo 1 >> data/myid |

4、Zookeeper02、03以此类推。

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

|  |
| --- |
| cp -r zoo\_sample.cfg zoo.cfg cp -r zoo\_sample.cfg zoo.cfg |

6、修改zoo.cfg，把dataDir=属性指定为刚创建的data文件夹。

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

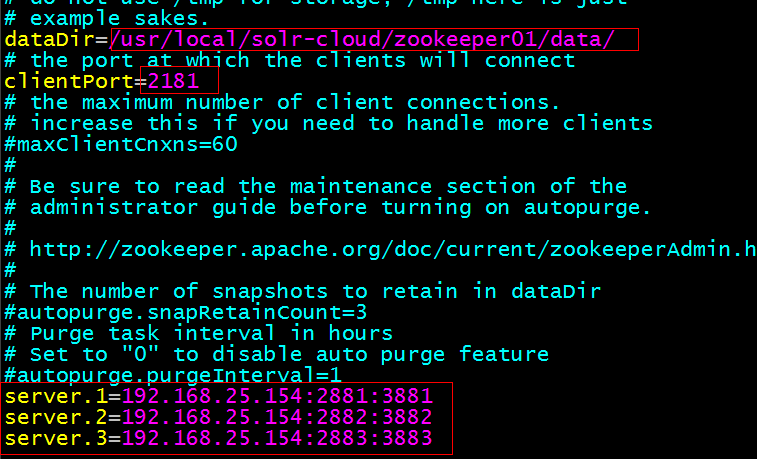
7、修改zoo.cfg，把clientPort指定为不冲突的端口号（01:2181、02:2182、03:2183）

8、在zoo.cfg中添加如下内容：

server.1=192.168.164.132:2881:3881

server.2=192.168.164.132:2882:3882

server.3=192.168.164.132:2883:3883



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

|  |
| --- |
| [root@localhost solr-cloud]# zookeeper01/bin/zkServer.sh start  [root@localhost solr-cloud]# zookeeper02/bin/zkServer.sh start  [root@localhost solr-cloud]# zookeeper03/bin/zkServer.sh start |

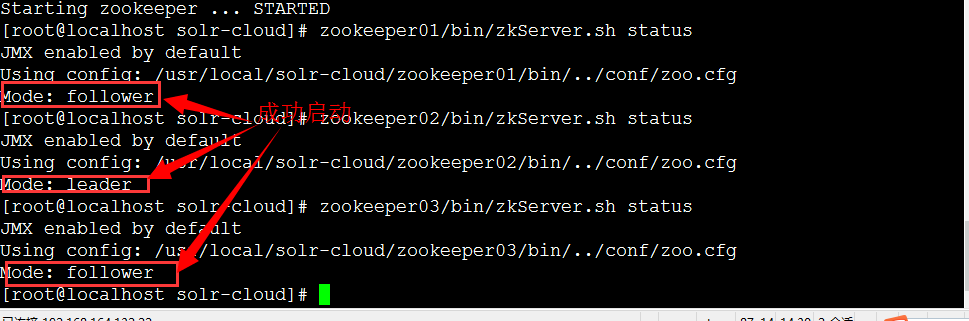
。

启动：./zkServer.sh start

关闭：./zkServer.sh stop

查看服务状态：./zkServer.sh status

|  |
| --- |
| [root@localhost solr-cloud]# zookeeper01/bin/zkServer.sh status  [root@localhost solr-cloud]# zookeeper02/bin/zkServer.sh status  [root@localhost solr-cloud]# zookeeper03/bin/zkServer.sh status |



安装zookeeper成功

## Solr集群的安装

第二部分：搭建solr集群

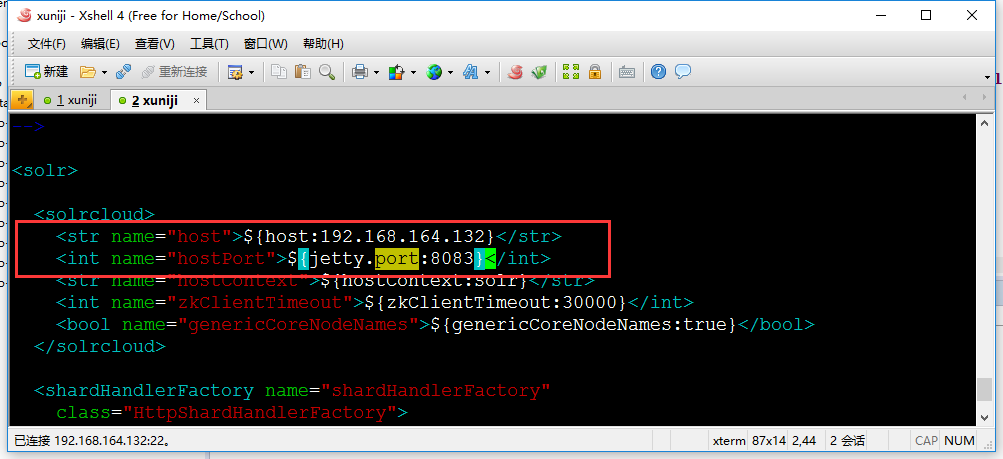
第一步：安装四个tomcat，修改其端口号不能冲突。8080~8083

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。

在/root/solr-4.10.3/example/scripts/cloud-scripts下执行如下命令

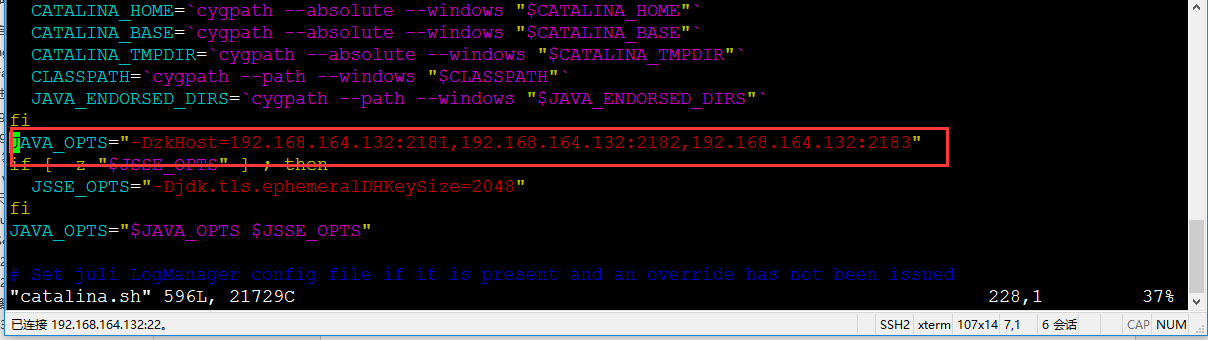
|  |
| --- |
| ./zkcli.sh -zkhost 192.168.164.132:2181,192.168.164.132:2182,192.168.164.132:2183 -cmd upconfig -confdir /usr/local/solr-cloud/solrhome01/collection1/conf -confname myconf |

第七步：查看是否上传成功。

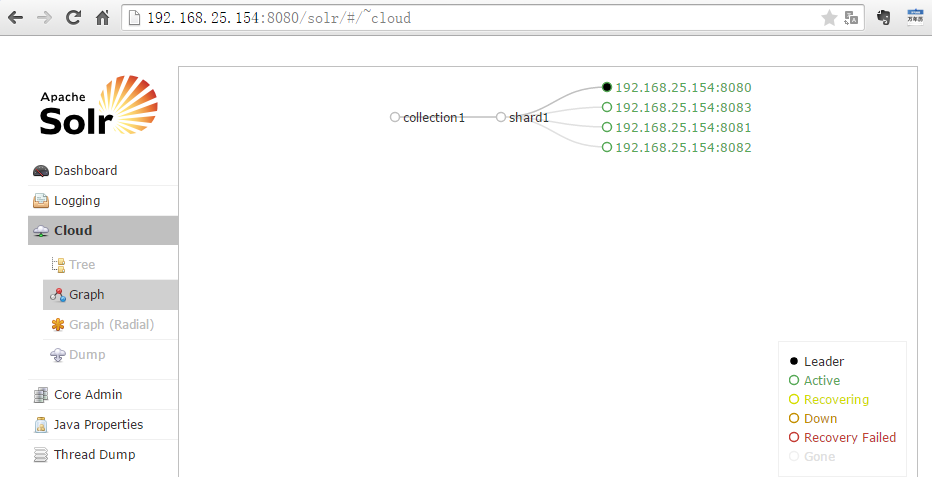
使用zookeeper的zkCli.sh命令。

第八步：告诉solr实例zookeeper的位置。需要修改tomcat的catalina.sh添加，每个节点都需要添加。即需要修改每个tomcat的catalina.sh

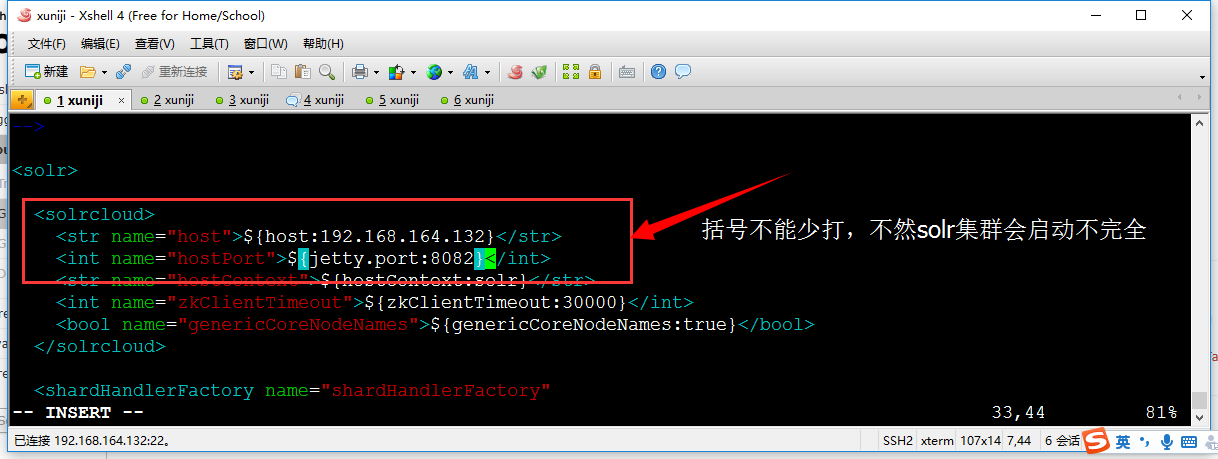
|  |
| --- |
| JAVA\_OPTS="-DzkHost=192.168.164.132:2181,192.168.164.132:2182,192.168.164.132:2183" |



第九步：启动每个solr实例。



注意：如果发现部分solr节点的tomcat启动成功，但solrAdmin界面出不来，极有可能是Solrhome下的solr.xml的配置出错了，如下图就是一种情况：



修改好后重新启动这个solr节点的Tomcat。

第十步：集群分片。

将集群分为两片，每片两个副本。

<http://192.168.164.132:8080/solr/admin/collections?action=CREATE&name=collection2&numShards=2&replicationFactor=2>

注意：似乎只能在主节点执行这个分片命令，在从节点执行这个命令会报错但是也将集群分为两片。我出现这个情况是直接删掉这个collection并重新在主节点执行。这样就和教案上一致了。





第十一步：删除不用collection1

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

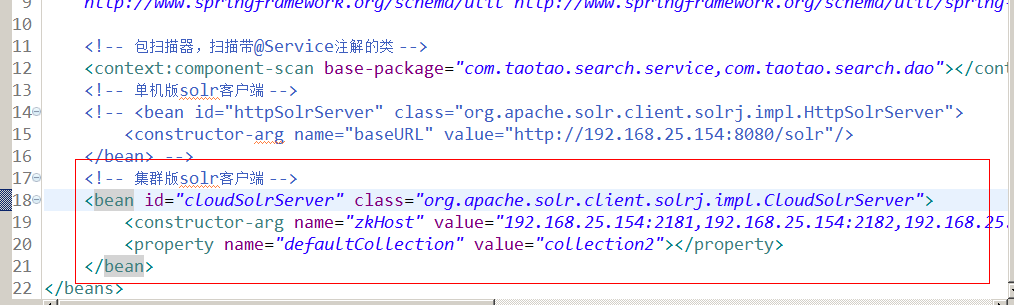


## 使用solrJ连接集群

|  |
| --- |
| @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。



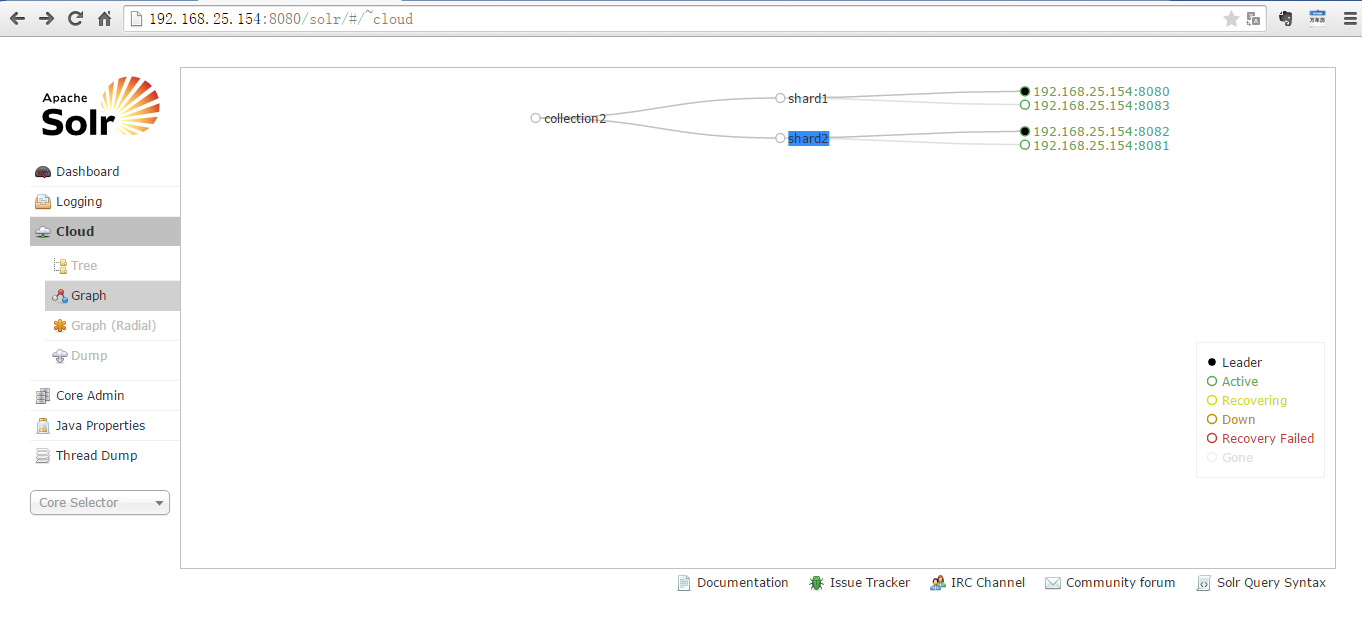
注意：所有在spring容器中装配的solrSever对象必须是

org.apache.solr.client.solrj.SolrServer。

因为SolrServer是org.apache.solr.client.solrj.impl.CloudSolrServer和org.apache.solr.client.solrj.impl.HttpSolrServer的父类。如果之前写的solrSever对象是HttpSolrServer类型，那么会在启动项目taotao-search时报错。

|  |
| --- |
| @Autowired  **private** SolrServer solrServer ; |

在solrAdmin上删除某条记录。Xml形式



# 第九天笔记

## 启动redis

两种启动方式，前端启动、后台启动。

前端启动：./redis-server

后台启动：

1、复制redis.conf到redis的安装目录

2、修改redis.conf。修改daemonize yes

3、[root@bogon redis]# ./redis-server redis.conf

## Freemarker的使用

### 什么是freemarker

FreeMarker是一个用Java语言编写的模板引擎，它基于模板来生成文本输出。FreeMarker与Web容器无关，即在Web运行时，它并不知道Servlet或HTTP。它不仅可以用作表现层的实现技术，而且还可以用于生成XML，JSP或Java 等。

### Freemarker的使用方法

第一步：把freemarker的jar包添加到工程中

第二步：freemarker的运行不依赖web容器，可以在java工程中运行。创建一个测试方法进行测试。

第三步：创建一个Configration对象

第四步：告诉config对象模板文件存放的路径。

第五步：设置config的默认字符集。一般是utf-8

第六步：从config对象中获得模板对象。需要制定一个模板文件的名字。

第七步：创建模板需要的数据集。可以是一个map对象也可以是一个pojo，把模板需要的数据都放入数据集。

第八步：创建一个Writer对象，指定生成的文件保存的路径及文件名。

第九步：调用模板对象的process方法生成静态文件。需要两个参数数据集和writer对象。

第十步：关闭writer对象。

### 代码实现

|  |
| --- |
| **public** **class** FreeMarkerTest {  @Test  **public** **void** testFreeMarker() **throws** Exception {  // 第一步：把freemarker的jar包添加到工程中  // 第二步：freemarker的运行不依赖web容器，可以在java工程中运行。创建一个测试方法进行测试。  // 第三步：创建一个Configuration对象  Configuration configuration = **new** Configuration(Configuration.*getVersion*());  // 第四步：告诉config对象模板文件存放的路径。  configuration.setDirectoryForTemplateLoading(**new** File("D:\\workspaces-itcast\\JaveEE18\\taotao-portal\\src\\main\\webapp\\WEB-INF\\ftl"));  // 第五步：设置config的默认字符集。一般是utf-8  configuration.setDefaultEncoding("utf-8");  // 第六步：从config对象中获得模板对象。需要制定一个模板文件的名字。  Template template = configuration.getTemplate("first.ftl");  // 第七步：创建模板需要的数据集。可以是一个map对象也可以是一个pojo，把模板需要的数据都放入数据集。  Map root = **new** HashMap<>();  root.put("hello", "hello freemarker");  // 第八步：创建一个Writer对象，指定生成的文件保存的路径及文件名。  Writer out = **new** FileWriter(**new** File("D:\\temp\\html\\hello.html"));  // 第九步：调用模板对象的process方法生成静态文件。需要两个参数数据集和writer对象。  template.process(root, out);  // 第十步：关闭writer对象。  out.flush();  out.close();  }  } |

模板：

${hello}

### Freemarker模板的写法

#### 取简单数据类型数据

使用EL表达式。

${hello}

#### 包装数据类型

模板：

|  |
| --- |
| <html>  <head>  <title>${title}</title>  </head>  <body>  <label>学号：</label>${student.id}<br>  <label>姓名：</label>${student.name}<br>  <label>住址：</label>${student.address}<br>  </body>  </html> |

#### 历遍集合/数组

List<Person> persons = new ArrayList<Person>();

省略….

页面中内容

<#list persons as p>

${p.id}/${p.name}

</#list>

#### 获得当前迭代的索引

List<Person> list = new ArrayList<Person>();

获取当前选代的索引:<br/>

<#list persons as p>

${p\_index}

</#list>

#### 模板中判断条件

<#if 判断条件>

<#else>

</#if>

逻辑运算符（== != || &&）

#### 日期类型格式化

默认格式

1：date

${cur\_time?date}

2：datetime

${cur\_time?datetime}

3：time

${cur\_time?time}

自定义格式

${cur\_time?string("yyyy-MM-dd HH:mm:ss")}

#### 处理null值

root.put(“val”,null);

解决办法

1:null 变 空串

${val!} ${val!"这里是空"}

2:为Null时给默认值

${val!“我是默认值"}

3、<#if curdate ??>

当前日期:${curdate?string("yyyy/MM/dd HH:mm:ss")}

<#else>

curdate属性为null

</#if>

#### Include

将另一个页面引入本页面时可用以下命令完成

<#include "/include/head.html">

# 第十天笔记

## \MANIFEST.MF (系统找不到指定的路径）的问题

Maven项目pom.xml文件报xxx\target\classes\META-INF\MANIFEST.MF (系统找不到指定的路径）的问题解决：

|  |
| --- |
| Description Resource Path Location Type  E:\learningsoft\eclipseWorkspaces\personalWorkspace\taotao\taotao-rest\target\m2e-wtp\web-resources\META-INF\MANIFEST.MF (系统找不到指定的路径。) pom.xml /taotao-rest line 1 Maven Configuration Problem |

解决方法：

|  |
| --- |
| 昨天关闭eclipse之前项目一切正常，今天打开的时候maven的pom.xml文件报错...\taotao-manager\taotao-manager-web\target\m2e-wtp\web-resources\META-INF\MANIFEST.MF (系统找不到指定的路径。)  经过查找解决方法如下  eclipse->project->clean->clean all projects.  .等着progress视图里的build完成就可以。 |

## 什么是单点登录系统

SSO英文全称Single Sign On，单点登录。SSO是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。它包括可以将这次主要的登录映射到其他应用中用于同一个用户的登录的机制。它是目前比较流行的企业业务整合的解决方案之一。

### 解决Session共享的问题的两种思路：

在传统的登录系统中，只有一个Tomcat服务，用户登录时直接使用javaee 的session可以满足系统判定用户是否登录的需求。

但在单点登录系统中由于有多个tomcat服务器，判断用户登录时session要做到在多个tomcat中共享。所以有如下解决方式：

1、tomcat做集群配置session复制。如果集群中节点很多，会形成网络风暴。推荐节点数量不要超过5个。（这个是Tomcat的一个功能，我没有实际操作过）

2、分布式架构。拆分成多个子系统。（推荐使用这种方式）

### 分布式架构

Redis中

使用redis模拟session

反向代理服务器

Tomcat集群

单点登录系统

Tomcat n

Tomcat2

Tomcat1

用户系统

订单系统

Tomcat1

Tomcat2

Tomcat n

Tomcat n

Tomcat2

Tomcat1

商品系统

## 单点登录系统处理流程



## 加上单点登录系统后淘淘商城的系统架构

