**淘淘商城**

**第九天**

# 课程计划

1. solr单机版安装
2. solrj回顾
3. solr集群
4. 创建搜索系统
5. 数据导入

# solr服务安装

我们需要实现商品的查询功能。

可以使用MySQL进行查询，但是MySQL的模糊查询（like）速度很慢，

而且数据量越大，查询速度就越慢。

一般不会使用MySQL实现查询功能，我们这里使用solr实现查询功能。

## 安装环境

solr是java开发的，安装solr需要有jdk，jdk1.7

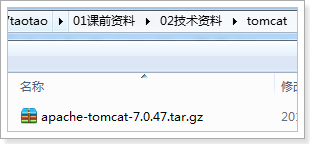
使用Tomcat部署，需要安装Tomcat

### 安装jdk

jdk已安装，安装步骤请参考Linux课程

### 安装Tomcat

Tomcat准备



1. 上传

按alt+p进入SFTP界面，进行上传



1. 解压

[root@itcast-01 ~]# tar -zxvf apache-tomcat-7.0.47.tar.gz

1. 移动

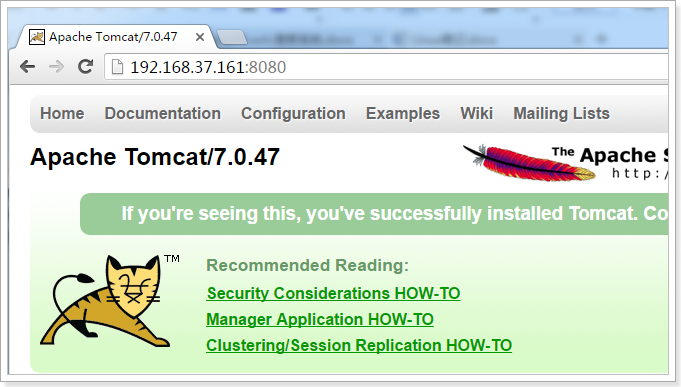
[root@itcast-01 ~]# mkdir /usr/local/solr

[root@itcast-01 ~]# mv apache-tomcat-7.0.47 /usr/local/solr/tomcat

1. 启动测试

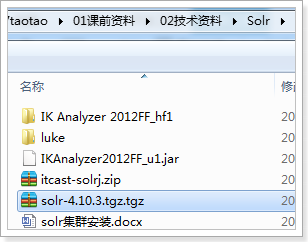
[root@itcast-01 ~]# /usr/local/solr/tomcat/bin/startup.sh

测试http://192.168.37.161:8080/：



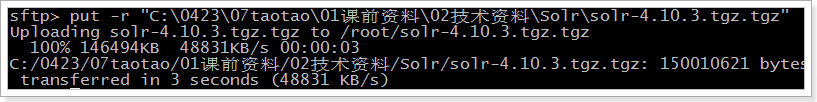
## 安装solr

solr准备



### 上传solr

按alt+p进入SFTP界面，进行上传



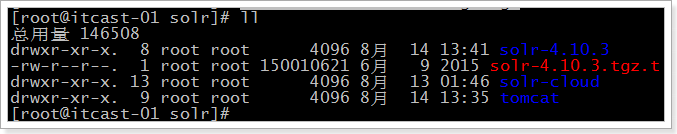
### 移动solr

[root@itcast-01 ~]# mv solr-4.10.3.tgz.tgz /usr/local/solr/

### 解压solr

[root@itcast-01 ~]# cd /usr/local/solr/

[root@itcast-01 solr]# tar -zxf solr-4.10.3.tgz.tgz



PS：进入solr-4.10.3/example目录有start.jar，可以使用java -jar start.jar启动内置的jetty服务器，从而启动solr服务

### 把solr部署到Tomcat

1. 停止之前运行的Tomcat

[root@itcast-01 solr]# /usr/local/solr/tomcat/bin/shutdown.sh

2. 删除Tomcat的webapps文件夹所有文件，并创建solr文件夹

[root@itcast-01 solr]# cd tomcat/webapps/

[root@itcast-01 webapps]# rm -rf \*

[root@itcast-01 webapps]# mkdir solr

3. 复制solr解压目录的solr.war到Tomcat中

[root@itcast-01 webapps]# cd /usr/local/solr/solr-4.10.3/example/webapps/

[root@itcast-01 webapps]# cp solr.war /usr/local/solr/tomcat/webapps/solr/

1. 解压solr.war

[root@itcast-01 webapps]# cd /usr/local/solr/tomcat/webapps/solr/

[root@itcast-01 solr]# jar -xvf solr.war

5. 删除war包

[root@itcast-01 solr]# rm -f solr.war

6. 把solr-4.10.3/example/lib/ext目录下的所有的jar包，添加到solr工程中

[root@itcast-01 ROOT]# cd /usr/local/solr/solr-4.10.3/example/lib/ext/

[root@itcast-01 ext]# cp \* /usr/local/solr/tomcat/webapps/solr/WEB-INF/lib/

1. 加入配置文件到solr工程中

[root@itcast-01 ext]# cd /usr/local/solr/solr-4.10.3/example/resources/

[root@itcast-01 resources]# cp \* /usr/local/solr/tomcat/webapps/solr/WEB-INF/classes/

### 创建solrhome

/example/solr目录就是一个solrhome。复制此目录到/usr/local/solr/solrhome

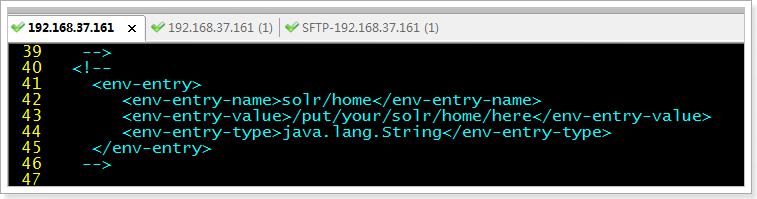
[root@itcast-01 ext]# cd /usr/local/solr/solr-4.10.3/example/

[root@itcast-01 example]# cp -r solr /usr/local/solr/solrhome

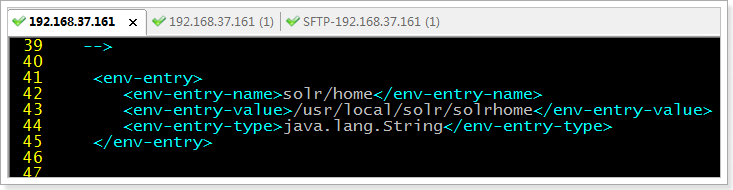
### 关联solr及solrhome

需要修改solr工程的web.xml文件

[root@itcast-01 example]# vim /usr/local/solr/tomcat/webapps/solr/WEB-INF/web.xml



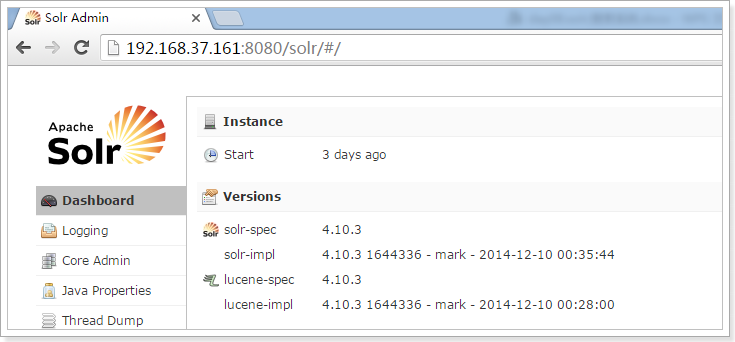
1. 删除注释
2. 修改第43行，指定solrhome地址(42行的solr/home是固定写法，不能修改)



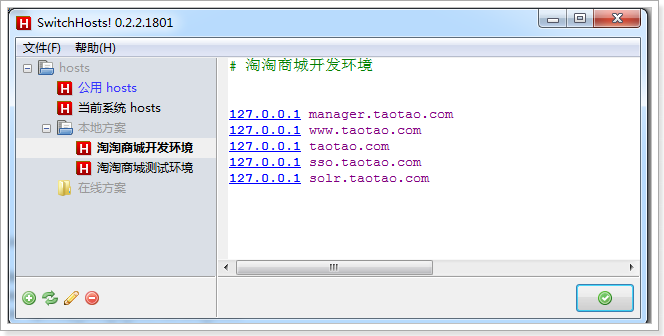
### 启动Tomcat

[root@itcast-01 example]# /usr/local/solr/tomcat/bin/startup.sh

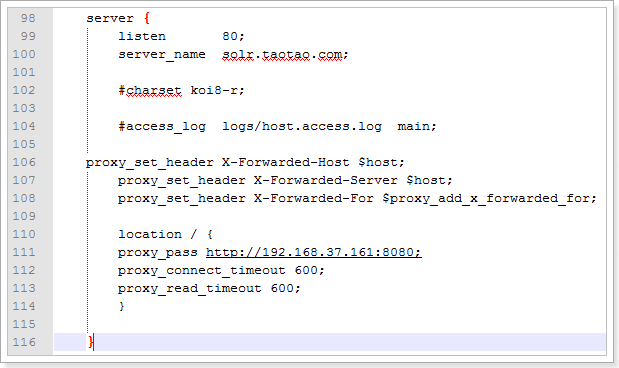
访问地址 http://192.168.37.161:8080/solr/#/



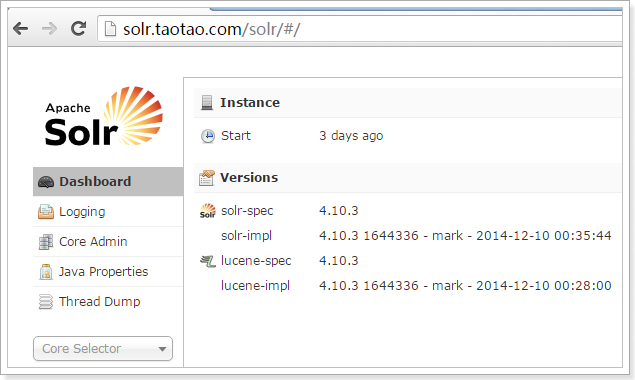
### 设置使用域名访问



### 设置Nginx反向代理服务器



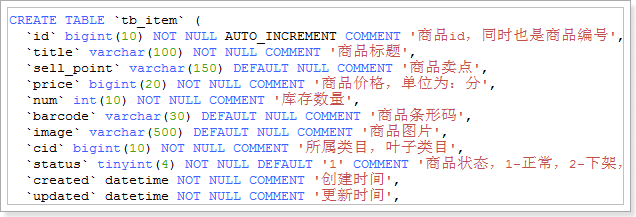
### 最终效果



# Solr配置和使用

## 业务域的配置

要实现商品搜索的功能，需要配置商品的业务域，只需要商品数据，不需要商品描述。商品的数据库表：



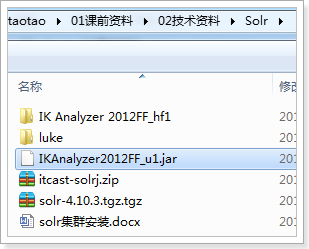
### 配置中文分词器

搜索商品标题需要中文分词，需要配置中文分词器IK

停止Tomcat

[root@itcast-01 example]# /usr/local/solr/tomcat/bin/shutdown.sh

上传中文分词器jar包



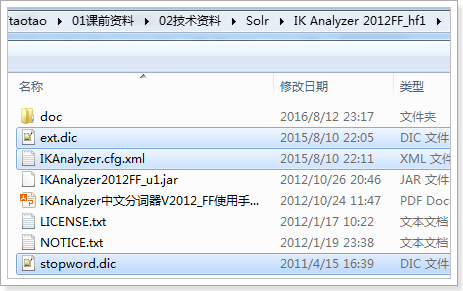
在SFTP窗口上传，alt+p

把IKAnalyzer2012FF\_u1.jar添加到solr/WEB-INF/lib目录下

[root@itcast-01 example]# cd /usr/local/solr/tomcat/webapps/solr/WEB-INF/lib/

[root@itcast-01 lib]# mv /root/IKAnalyzer2012FF\_u1.jar ./

复制IKAnalyzer的配置文件和自定义词典和停用词词典到Tomcat服务



在SFTP窗口上传，alt+p

把IKAnalyzer2012FF\_u1.jar添加到solr/WEB-INF/classes目录下

[root@itcast-01 lib]# cd ../classes

[root@itcast-01 classes]# mv /root/ext.dic ./

[root@itcast-01 classes]# mv /root/IKAnalyzer.cfg.xml ./

[root@itcast-01 classes]# mv /root/stopword.dic ./

在schema.xml配置中文分词器

[root@itcast-01 WEB-INF]# cd /usr/local/solr/solrhome/collection1/conf/

[root@itcast-01 conf]# vim schema.xml

在最后面添加如下：

<!-- IKAnalyzer-->

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

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

</fieldType>

<!--IKAnalyzer Field-->

<field name="content\_ik" type="text\_ik" indexed="true" stored="true" />

重启Tomcat生效

### 配置业务域

根据商品数据库表配置业务域

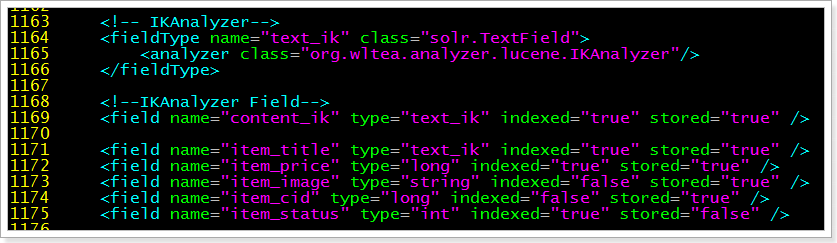
<field name="item\_title" type="text\_ik" indexed="true" stored="true" />

<field name="item\_price" type="long" indexed="true" stored="true" />

<field name="item\_image" type="string" indexed="false" stored="true" />

<field name="item\_cid" type="long" indexed="false" stored="true" />

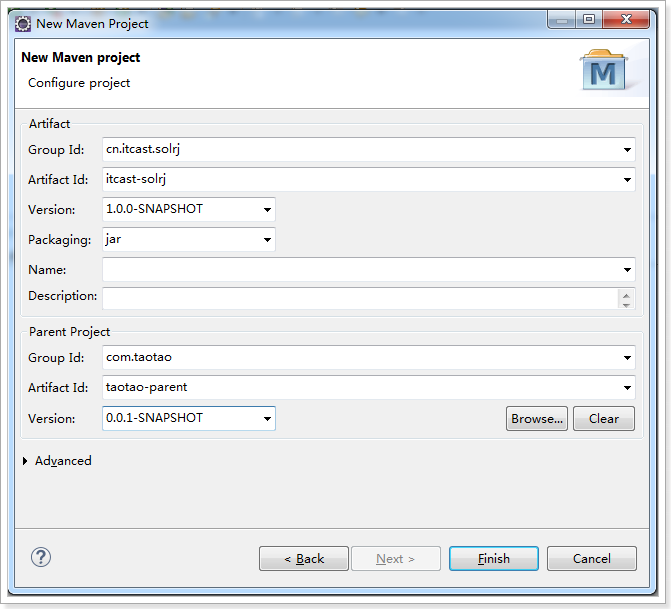
<field name="item\_status" type="int" indexed="true" stored="false" />



重启Tomcat使其生效

## 回顾solrj

### 创建itcast-solrj



### 加入依赖

直接使用笔记的依赖，solrj必须使用commons-logging日志包

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>cn.itcast.solrj</groupId>

<artifactId>itcast-solrj</artifactId>

<version>1.0.0-SNAPSHOT</version>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.1.1</version>

</dependency>

<!-- solr客户端 -->

<dependency>

<groupId>org.apache.solr</groupId>

<artifactId>solr-solrj</artifactId>

</dependency>

<!-- Apache工具包 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

</dependencies>

</project>

### 创建httpSolrServer

**public** **class** SolrjTest {

**private** HttpSolrServer httpSolrServer;

@Before

**public** **void** init() {

// 声明接口地址

String baseURL = "http://solr.taotao.com/solr/collection1";

// 创建HttpSolrServer

**this**.httpSolrServer = **new** HttpSolrServer(baseURL);

}

}

### 实现新增/修改索引库

solrj没有修改方法，是使用add方法进行修改

在业务域中配置了id，如果保存时，如果id存在则更新，id不存在则新增

@Test

**public** **void** testCreateAndUpdateIndex() **throws** Exception {

// 创建SolrInputDocument对象

SolrInputDocument document = **new** SolrInputDocument();

document.addField("id", "c1001");

document.addField("item\_title", "java编程思想");

// 把SolrInputDocument对象添加到索引库中

**this**.httpSolrServer.add(document);

// 4. 提交

**this**.httpSolrServer.commit();

}

### 实现删除索引库

两种方式删除

@Test

**public** **void** testDeleteIndex() **throws** Exception {

// 根据id删除索引数据

// this.httpSolrServer.deleteById("c1001");

// 根据条件删除（如果是\*:\*就表示全部删除，慎用）

**this**.httpSolrServer.deleteByQuery("\*:\*");

// 提交

**this**.httpSolrServer.commit();

}

### 实现查询索引库

@Test

**public** **void** testSearchIndex() **throws** Exception {

// 创建搜索对象

SolrQuery solrQuery = **new** SolrQuery();

// 设置查询条件

solrQuery.setQuery("item\_title:java");

// 设置分页

solrQuery.setStart(0);

solrQuery.setRows(10);

// 设置高亮

solrQuery.setHighlight(**true**);

solrQuery.addHighlightField("item\_title");

solrQuery.setHighlightSimplePre("<font color=\"red\">");

solrQuery.setHighlightSimplePost("</font>");

// 查询数据

QueryResponse response = **this**.httpSolrServer.query(solrQuery);

SolrDocumentList results = response.getResults();

System.***out***.println("搜索到的数据总条数：" + results.getNumFound());

Map<String, Map<String, List<String>>> map = response.getHighlighting();

// 解析查询结果

**for** (SolrDocument solrDocument : results) {

System.***out***.println("----------------------------------------------------");

// 获取高亮数据

List<String> list = map.get(solrDocument.get("id")).get("item\_title");

System.***out***.println("商品ID：" + solrDocument.get("id"));

// 显示高亮

**if** (list != **null** && list.size() > 0) {

System.***out***.println("商品名称：" + list.get(0));

} **else** {

System.***out***.println("商品名称：" + solrDocument.get("item\_title"));

}

}

}

# solr集群版

## 什么是SolrCloud

SolrCloud(solr 云)是Solr提供的分布式搜索方案，当你需要大规模，容错，分布式索引和检索能力时使用 SolrCloud。当一个系统的索引数据量少的时候是不需要使用SolrCloud的，当索引量很大，搜索请求并发很高，这时需要使用SolrCloud来满足这些需求。

SolrCloud是基于Solr和Zookeeper的分布式搜索方案，它的主要思想是使用zookeeper作为集群的配置信息中心。

它有几个特色

1）集中式的配置信息

2）自动容错

3）近实时搜索

4）查询时自动负载均衡

## zookeeper介绍

顾名思义zookeeper就是动物园管理员，他是用来管hadoop（大象）、Hive(蜜蜂)、pig(小猪)的管理员， Apache Hbase和 Apache Solr 的分布式集群都用到了zookeeper；Zookeeper:是一个分布式的、开源的程序协调服务，是hadoop项目下的一个子项目。

## Zookeeper使用场景

### 配置管理

在我们的应用中除了代码外，还有一些就是各种配置。比如数据库连接等。一般我们都是使用配置文件的方式，在代码中引入这些配置文件。但是当我们只有一种配置，只有一台服务器，并且不经常修改的时候，使用配置文件是一个很好的做法，但是如果我们配置非常多，有很多服务器都需要这个配置，而且还可能是动态的话使用配置文件就不是个好主意了。这个时候往往需要寻找一种集中管理配置的方法，我们在这个集中的地方修改了配置，所有对这个配置感兴趣的都可以获得变更。比如我们可以把配置放在数据库里，然后所有需要配置的服务都去这个数据库读取配置。但是，因为很多服务的正常运行都非常依赖这个配置，所以需要这个集中提供配置服务的服务具备很高的可靠性。一般我们可以用一个集群来提供这个配置服务，但是用集群提升可靠性，那如何保证配置在集群中的一致性呢？ 这个时候就需要使用一种实现了一致性协议的服务了。Zookeeper就是这种服务，它使用Zab这种一致性协议来提供一致性。现在有很多开源项目使用Zookeeper来维护配置，比如在HBase中，客户端就是连接一个Zookeeper，获得必要的HBase集群的配置信息，然后才可以进一步操作。还有在开源的消息队列Kafka中，也使用Zookeeper来维护broker的信息。在Alibaba开源的SOA框架Dubbo中也广泛的使用Zookeeper管理一些配置来实现服务治理。

### 名字服务

名字服务这个就很好理解了。比如为了通过网络访问一个系统，我们得知道对方的IP地址，但是IP地址对人非常不友好，这个时候我们就需要使用域名来访问。但是计算机是不能是别域名的。怎么办呢？如果我们每台机器里都备有一份域名到IP地址的映射，这个倒是能解决一部分问题，但是如果域名对应的IP发生变化了又该怎么办呢？于是我们有了DNS这个东西。我们只需要访问一个大家熟知的(known)的点，它就会告诉你这个域名对应的IP是什么。在我们的应用中也会存在很多这类问题，特别是在我们的服务特别多的时候，如果我们在本地保存服务的地址的时候将非常不方便，但是如果我们只需要访问一个大家都熟知的访问点，这里提供统一的入口，那么维护起来将方便得多了。

### 分布式锁

其实在第一篇文章中已经介绍了Zookeeper是一个分布式协调服务。这样我们就可以利用Zookeeper来协调多个分布式进程之间的活动。比如在一个分布式环境中，为了提高可靠性，我们的集群的每台服务器上都部署着同样的服务。但是，一件事情如果集群中的每个服务器都进行的话，那相互之间就要协调，编程起来将非常复杂。而如果我们只让一个服务进行操作，那又存在单点。通常还有一种做法就是使用分布式锁，在某个时刻只让一个服务去干活，当这台服务出问题的时候锁释放，立即fail over到另外的服务。这在很多分布式系统中都是这么做，这种设计有一个更好听的名字叫Leader Election(leader选举)。比如HBase的Master就是采用这种机制。但要注意的是分布式锁跟同一个进程的锁还是有区别的，所以使用的时候要比同一个进程里的锁更谨慎的使用。

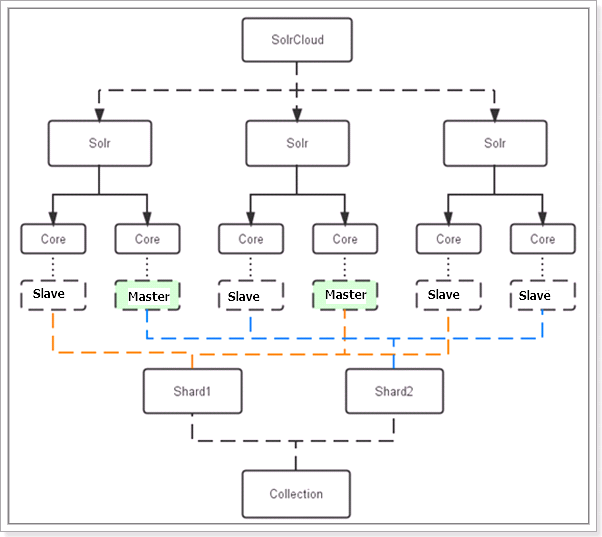
### 集群管理

在分布式的集群中，经常会由于各种原因，比如硬件故障，软件故障，网络问题，有些节点会进进出出。有新的节点加入进来，也有老的节点退出集群。这个时候，集群中其他机器需要感知到这种变化，然后根据这种变化做出对应的决策。比如我们是一个分布式存储系统，有一个中央控制节点负责存储的分配，当有新的存储进来的时候我们要根据现在集群目前的状态来分配存储节点。这个时候我们就需要动态感知到集群目前的状态。还有，比如一个分布式的SOA架构中，服务是一个集群提供的，当消费者访问某个服务时，就需要采用某种机制发现现在有哪些节点可以提供该服务(这也称之为服务发现，比如Alibaba开源的SOA框架Dubbo就采用了Zookeeper作为服务发现的底层机制)。还有开源的Kafka队列就采用了Zookeeper作为Cosnumer的上下线管理。

## SolrCloud结构

SolrCloud为了降低单机的处理压力，需要由多台服务器共同来完成索引和搜索任务。实现的思路是将索引数据进行Shard（分片）拆分，每个分片由多台的服务器共同完成，当一个索引或搜索请求过来时会分别从不同的Shard的服务器中操作索引。

SolrCloud需要Solr基于Zookeeper部署，Zookeeper是一个集群管理软件，由于SolrCloud需要由多台服务器组成，由Zookeeper来进行协调管理。



对上图进行图解，如下：



### 物理结构

三个Solr实例（ 每个实例包括两个Core），组成一个SolrCloud。

### 逻辑结构

索引集合包括两个Shard（shard1和shard2），shard1和shard2分别由三个Core组成，其中一个Leader两个Replication，Leader是由zookeeper选举产生，zookeeper控制每个shard上三个Core的索引数据一致，解决高可用问题。

用户发起索引请求分别从shard1和shard2上获取，解决高并发问题。

#### collection

Collection在SolrCloud集群中是一个逻辑意义上的完整的索引结构。它常常被划分为一个或多个Shard（分片），它们使用相同的配置信息。

比如：针对商品信息搜索可以创建一个collection。

collection=shard1+shard2+....+shardX

#### Shard

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

#### Core

每个Core是Solr中一个独立运行单位，提供 索引和搜索服务。一个shard需要由一个Core或多个Core组成。由于collection由多个shard组成所以collection一般由多个core组成。

#### Master或Slave

Master是master-slave结构中的主结点（通常说主服务器），Slave是master-slave结构中的从结点（通常说从服务器或备服务器）。同一个Shard下master和slave存储的数据是一致的，这是为了达到高可用目的。

## 架构分析

SolrCloud结构图如下：



### 集群分成两大部分：

1. zookeeper集群
   1. zookeeper需要保证高可用，需要搭建集群
   2. zookeeper需要进行投票和选举机制，最少需要三个节点
2. Solr的集群
   1. 需要四个节点
   2. 整个collection分成两个分片，每个分片都由两个服务器组成，master和slave
   3. 每个Tomcat有一个solr服务

集群安装：

参考课后资料《虚拟机安装.docx》solr集群部分

## 启动集群

在虚拟机中依次执行以下命令：

启动zookeeper集群：

/usr/local/solr/solr-cloud/zookeeper01/bin/zkServer.sh start

/usr/local/solr/solr-cloud/zookeeper02/bin/zkServer.sh start

/usr/local/solr/solr-cloud/zookeeper03/bin/zkServer.sh start

zookeeper有3个实例，实例的端口号是3181~3183

启动solr集群

/usr/local/solr/solr-cloud/tomcat-solr01/bin/startup.sh

/usr/local/solr/solr-cloud/tomcat-solr02/bin/startup.sh

/usr/local/solr/solr-cloud/tomcat-solr03/bin/startup.sh

/usr/local/solr/solr-cloud/tomcat-solr04/bin/startup.sh

访问集群：

集群有4个节点，只需要访问任意的一个节点即可

Tomcat的端口号四8081~8084

## 集群测试

在itcast-solrj测试，和单机版一样，仅仅是创建连接对象不同

测试代码一样

**public** **class** SolrjCloudTest {

**private** CloudSolrServer cloudSolrServer;

@Before

**public** **void** init() {

// 声明接口地址

String zkHost = "192.168.37.161:3181,192.168.37.161:3182,192.168.37.161:3183";

// 创建HttpSolrServer

**this**.cloudSolrServer = **new** CloudSolrServer(zkHost);

// 设置collection

**this**.cloudSolrServer.setDefaultCollection("collection2");

}

@Test

**public** **void** testCreateAndUpdateIndex() **throws** Exception {

// 创建SolrInputDocument对象

SolrInputDocument document = **new** SolrInputDocument();

document.addField("id", "c1001");

document.addField("item\_title", "java编程思想");

// 把SolrInputDocument对象添加到索引库中

**this**.cloudSolrServer.add(document);

// 4. 提交

**this**.cloudSolrServer.commit();

}

@Test

**public** **void** testDeleteIndex() **throws** Exception {

// 根据id删除索引数据

// this.httpSolrServer.deleteById("c1001");

// 根据条件删除（如果是\*:\*就表示全部删除，慎用）

**this**.cloudSolrServer.deleteByQuery("\*:\*");

// 提交

**this**.cloudSolrServer.commit();

}

@Test

**public** **void** testSearchIndex() **throws** Exception {

// 创建搜索对象

SolrQuery solrQuery = **new** SolrQuery();

// 设置查询条件

solrQuery.setQuery("item\_title:java");

// 设置分页

solrQuery.setStart(0);

solrQuery.setRows(10);

// 设置高亮

solrQuery.setHighlight(**true**);

solrQuery.addHighlightField("item\_title");

solrQuery.setHighlightSimplePre("<font color=\"red\">");

solrQuery.setHighlightSimplePost("</font>");

// 查询数据

QueryResponse response = **this**.cloudSolrServer.query(solrQuery);

SolrDocumentList results = response.getResults();

System.***out***.println("搜索到的数据总条数：" + results.getNumFound());

Map<String, Map<String, List<String>>> map = response.getHighlighting();

// 解析查询结果

**for** (SolrDocument solrDocument : results) {

System.***out***.println("----------------------------------------------------");

// 获取高亮数据

List<String> list = map.get(solrDocument.get("id")).get("item\_title");

System.***out***.println("商品ID：" + solrDocument.get("id"));

// 显示高亮

**if** (list != **null** && list.size() > 0) {

System.***out***.println("商品名称：" + list.get(0));

} **else** {

System.***out***.println("商品名称：" + solrDocument.get("item\_title"));

}

}

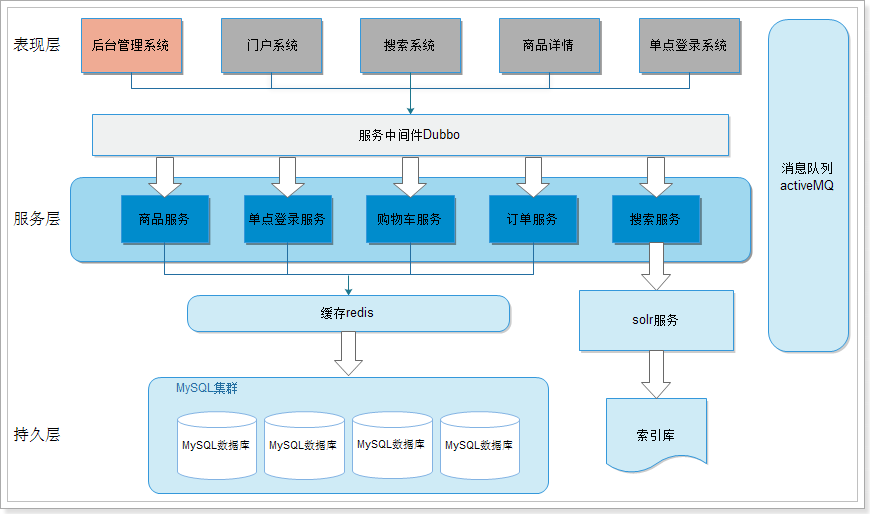
}

}

# 创建搜索系统

## 架构分析

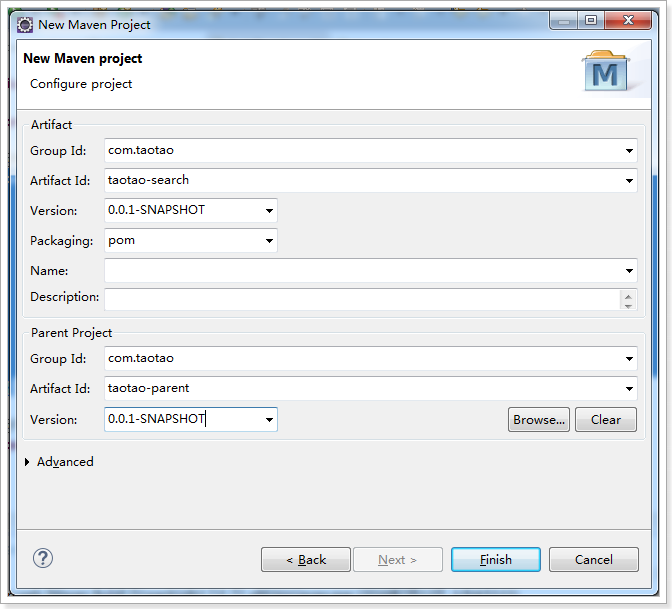
需要创建搜索表现层和服务层



## 创建服务层

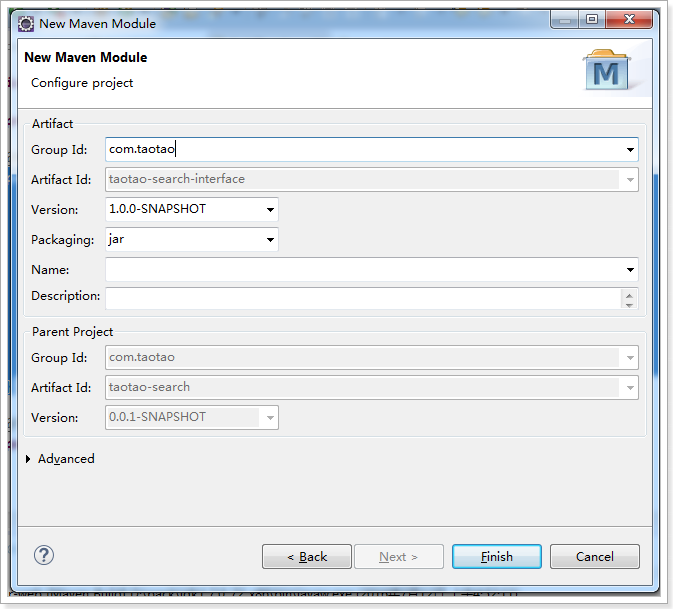
### 创建taotao-search工程

聚合工程父工程



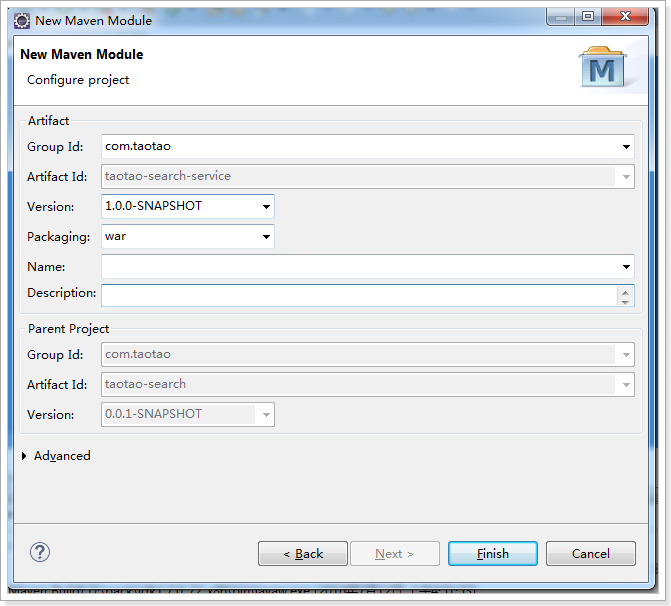
### 创建taotao-search-interface

聚合工程子工程



### 创建taotao-search-service

聚合工程子工程



### 加入依赖

服务层跟taotao-manager很像，可以参考

#### taotao-search

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-search</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<modules>

<module>taotao-search-interface</module>

<module>taotao-search-service</module>

</modules>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- Apache通用工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8084</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

</build>

</project>

#### taotao-search-interface

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-search</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>taotao-search-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

<dependencies>

<!-- 加入taotao-manager-pojo依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-manager-pojo</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

</project>

#### taotao-search-service

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-search</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>taotao-search-service</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<!-- 加入taotao-manager-mapper依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-manager-mapper</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- 加入taotao-search-interface依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-search-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- dubbo相关 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

</dependency>

<!-- solr客户端 -->

<dependency>

<groupId>org.apache.solr</groupId>

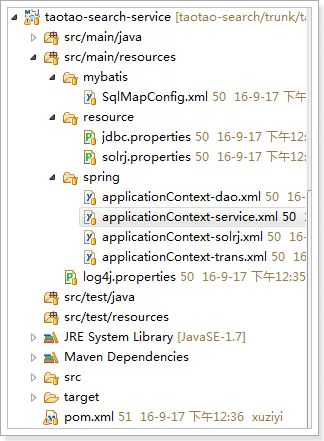
<artifactId>solr-solrj</artifactId>

</dependency>

</dependencies>

</project>

### 加入配置文件



#### applicationContext-dao.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 配置 读取properties文件 jdbc.properties -->

<context:property-placeholder location=*"classpath:resource/\*.properties"* />

<!-- 配置数据源 -->

<bean id=*"dataSource"* class=*"com.jolbox.bonecp.BoneCPDataSource"*

destroy-method=*"close"*>

<!-- 数据库驱动 -->

<property name=*"driverClass"* value=*"${jdbc.driver}"* />

<!-- 相应驱动的jdbcUrl -->

<property name=*"jdbcUrl"* value=*"${jdbc.url}"* />

<!-- 数据库的用户名 -->

<property name=*"username"* value=*"${jdbc.username}"* />

<!-- 数据库的密码 -->

<property name=*"password"* value=*"${jdbc.password}"* />

<!-- 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：240，如果要取消则设置为0 -->

<property name=*"idleConnectionTestPeriod"* value=*"60"* />

<!-- 连接池中未使用的链接最大存活时间，单位是分，默认值：60，如果要永远存活设置为0 -->

<property name=*"idleMaxAge"* value=*"30"* />

<!-- 每个分区最大的连接数 -->

<property name=*"maxConnectionsPerPartition"* value=*"150"* />

<!-- 每个分区最小的连接数 -->

<property name=*"minConnectionsPerPartition"* value=*"5"* />

</bean>

<!-- 配置SqlSessionFactory -->

<bean class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 配置Mybatis核心配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"* />

<!-- 配置数据源 -->

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

<!-- 配置Mapper扫描 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 配置Mapper扫描包 -->

<property name=*"basePackage"* value=*"com.taotao.manager.mapper"* />

</bean>

</beans>

#### applicationContext-trans.xml

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>

<!-- 定义事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

<!-- 定义事务策略 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<!-- 传播行为 -->

<tx:method name=*"save\*"* propagation=*"REQUIRED"* />

<tx:method name=*"insert\*"* propagation=*"REQUIRED"* />

<tx:method name=*"add\*"* propagation=*"REQUIRED"* />

<tx:method name=*"create\*"* propagation=*"REQUIRED"* />

<tx:method name=*"delete\*"* propagation=*"REQUIRED"* />

<tx:method name=*"update\*"* propagation=*"REQUIRED"* />

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"query\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

</tx:attributes>

</tx:advice>

<aop:config>

<!--pointcut元素定义一个切入点，execution中的第一个星号 用以匹配方法的返回类型， 这里星号表明匹配所有返回类型。 com.abc.dao.\*.\*(..)表明匹配cn.itcast.mybatis.service包下的所有类的所有

方法 -->

<aop:pointcut id=*"myPointcut"*

expression=*"execution(\* com.taotao.search.service.\*.\*(..))"* />

<!--将定义好的事务处理策略应用到上述的切入点 -->

<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"myPointcut"* />

</aop:config>

</beans>

#### applicationContext-service.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 配置service扫描 -->

<context:component-scan base-package=*"com.taotao.search.service"* />

<!-- 配置dubbo -->

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"taotao-search-service"* />

<!-- 使用multicast广播注册中心暴露服务地址 -->

<!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->

<!-- <dubbo:registry address="N/A" /> -->

<!-- 使用zookeeper注册中心 -->

<dubbo:registry protocol=*"zookeeper"* address=*"192.168.37.161:2181"* />

<!-- 用dubbo协议在20880端口暴露服务 -->

<dubbo:protocol name=*"dubbo"* port=*"20882"* />

<!-- <bean id="testServiceImpl" class="com.taotao.manager.service.impl.TestServiceImpl"

/> -->

<!-- 声明需要暴露的服务接口 -->

<!-- <dubbo:service interface="com.taotao.content.service.ContentCategoryService" -->

<!-- ref="contentCategoryServiceImpl" timeout="1000000" /> -->

</beans>

#### applicationContext-solrj.xml

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>

<!-- 配置CloudSolrServer -->

<bean class=*"org.apache.solr.client.solrj.impl.CloudSolrServer"*>

<!-- 配置zookeeper连接地址 -->

<constructor-arg name=*"zkHost"* value=*"${solr.zkhost}"* />

<!-- 设置collection的名字 -->

<property name=*"defaultCollection"* value=*"${solr.collectionName}"* />

</bean>

</beans>

#### solrj.properties

#配置solr集群的zookeeper连接地址

solr.zkhost=192.168.37.161:3181,192.168.37.161:3182,192.168.37.161:3183

#solr集群的索引库collection的名字

solr.collectionName=collection2

#### web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"MyWebApp"* version=*"2.5"*>

<display-name>taotao-search</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<!-- spring -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/applicationContext\*.xml</param-value>

</context-param>

<!-- spring监听器 -->

<listener>

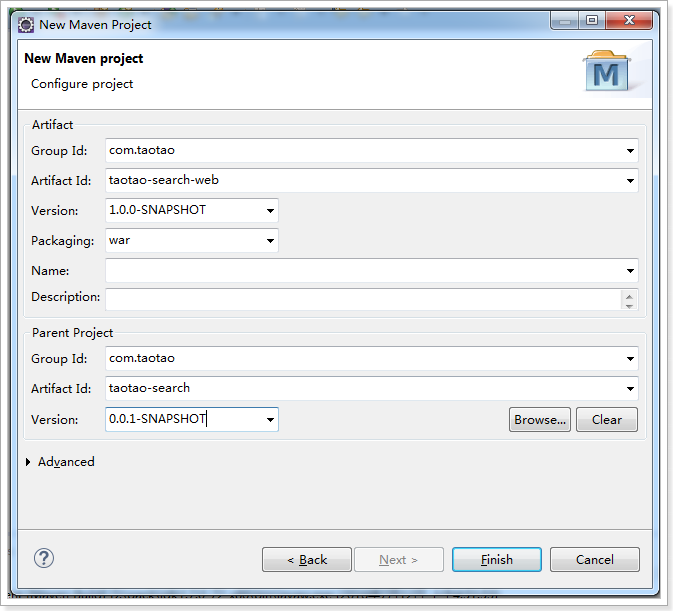
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

</web-app>

## 创建表现层

### 创建taotao-search-web



### 加入依赖

和taotao-portal很像，可以参考

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-search-web</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<!-- 加入taotao-search-interface依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-search-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- JSP相关 -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<scope>provided</scope>

</dependency>

<!-- Apache通用工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

<!-- dubbo相关 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8087</port>

<path>/</path>

</configuration>

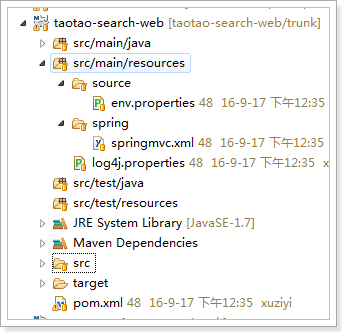
</plugin>

</plugins>

</build>

</project>

### 加入配置文件



#### springmvc.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 加载配置文件 -->

<context:property-placeholder location=*"classpath:env.properties"* />

<!-- 配置controller扫描 -->

<context:component-scan base-package=*"com.taotao.search.controller"* />

<!-- 配置注解驱动 -->

<mvc:annotation-driven />

<!-- 配置视图解析器 -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 配置前缀 -->

<property name=*"prefix"* value=*"/WEB-INF/views/"* />

<!-- 配置后缀 -->

<property name=*"suffix"* value=*".jsp"* />

</bean>

<!-- 配置dubbo服务 -->

<dubbo:application name=*"taotao-search-web"* />

<!-- 使用广播 -->

<!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->

<dubbo:registry protocol=*"zookeeper"* address=*"192.168.37.161:2181"* />

<!-- 声明要调用的服务，timeout是设置连接超时最长时间,如果不设置，超时时间默认是3秒 -->

<!-- <dubbo:reference interface="com.taotao.content.service.ContentService" -->

<!-- id="contentService" timeout="1000000" /> -->

</beans>

#### web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"MyWebApp"* version=*"2.5"*>

<display-name>taotao-search-web</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<!-- 配置解决post提交乱码的问题 -->

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 配置springMVC -->

<servlet>

<servlet-name>taotao-search-web</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 设置springMVC配置文件 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- 配置springMVC映射 -->

<servlet-mapping>

<servlet-name>taotao-search-web</servlet-name>

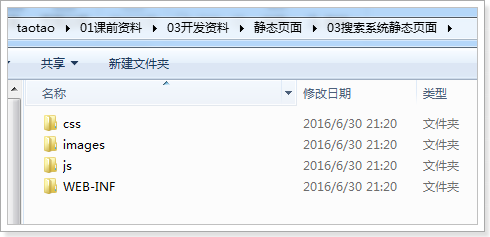
<!-- 所有以html结尾的请求进入springMVC -->

<url-pattern>\*.html</url-pattern>

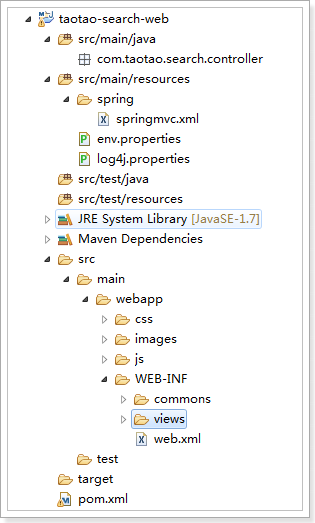
</servlet-mapping>

</web-app>

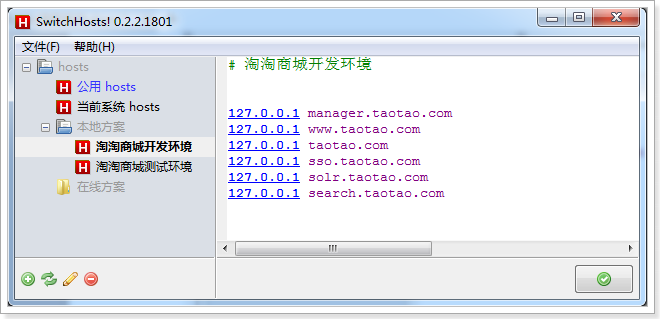
## 加入静态资源



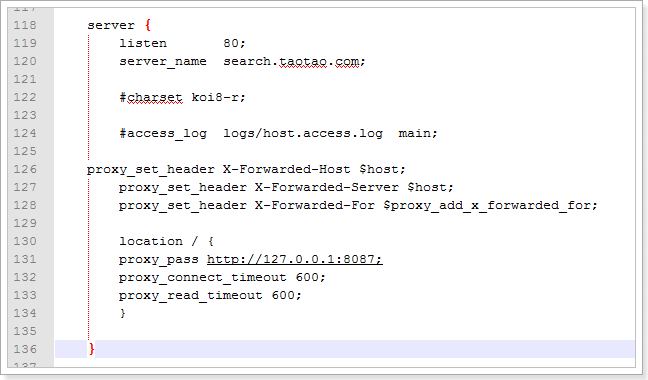
效果



## 修改域名访问

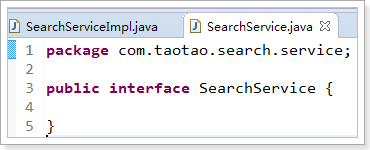


## 修改Nginx配置

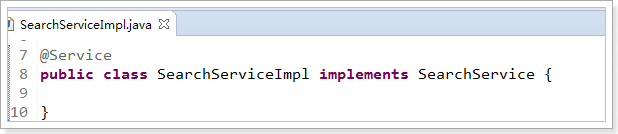


## 开发准备

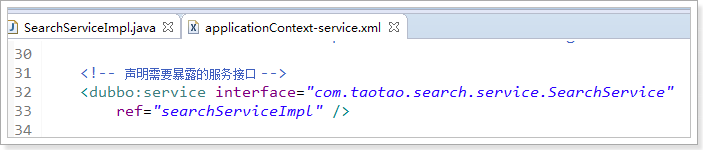
在taotao-search-interface创建接口



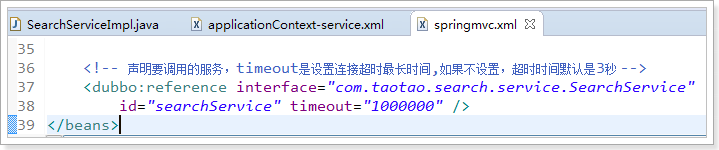
在taotao-search-service创建接口实现类



声明服务发布



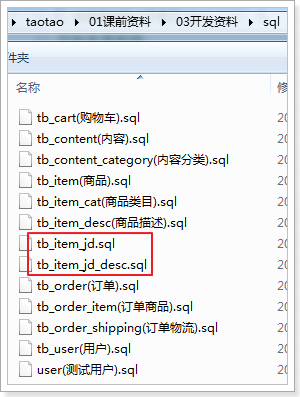
声明服务调用



# 数据准备

## 把商品数据导入到MySQL中

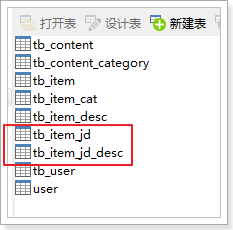
### 数据sql脚本准备



### 执行sql脚本

在MySQL执行sql脚本，

效果

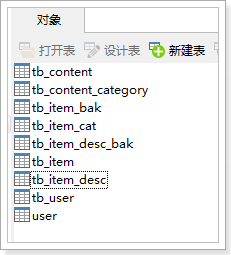


### 修改数据库名称

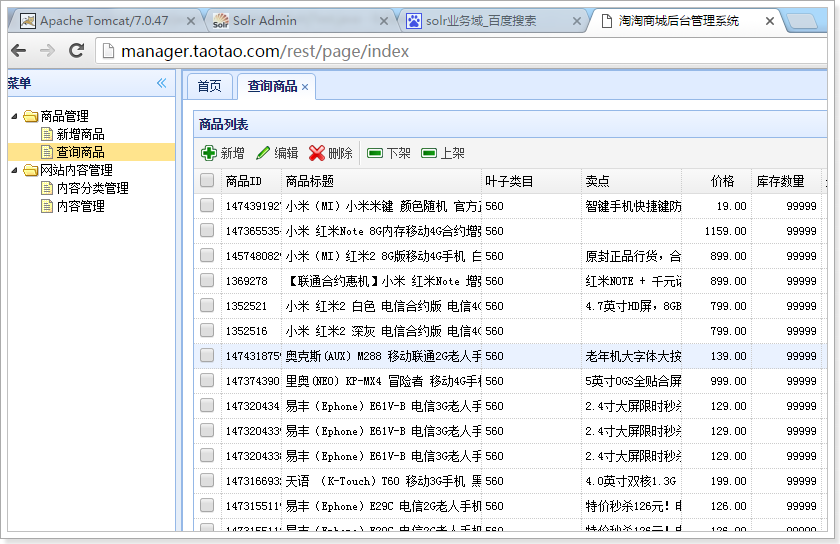
把原来的tb\_item和tb\_item\_desc的表名后面加tb\_item\_bak和tb\_item\_desc\_bak

把tb\_item\_jd和tb\_item\_jd\_desc的\_jd去掉。

效果



启动后台系统，查看数据导入成功：



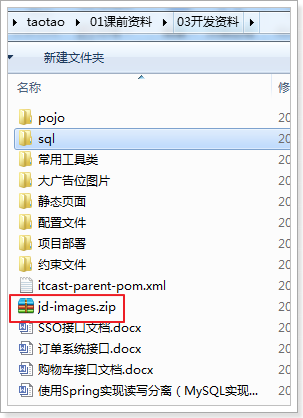
## 设置Nginx图片服务器

之前介绍过，企业搭建图片服务器主要有两种方式

1. 搭建FastDFS分布式文件系统
2. 使用web服务器（Nginx、Apache）搭建

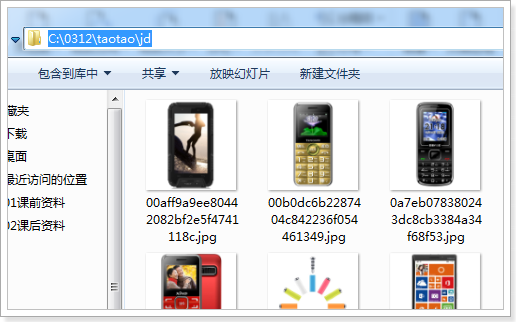
第一种已经使用了，我们再使用第二种方式搭建图片服务器使商品图片正常显示

### 商品图片



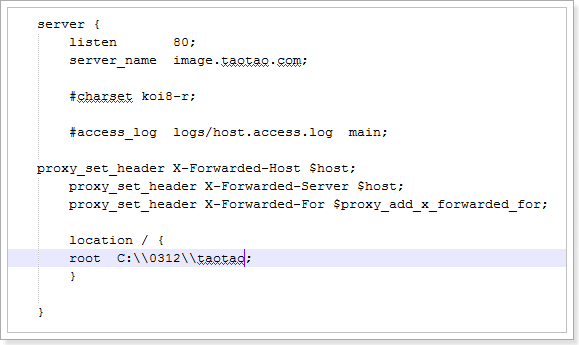
### 解压图片

解压图片到taotao目录下

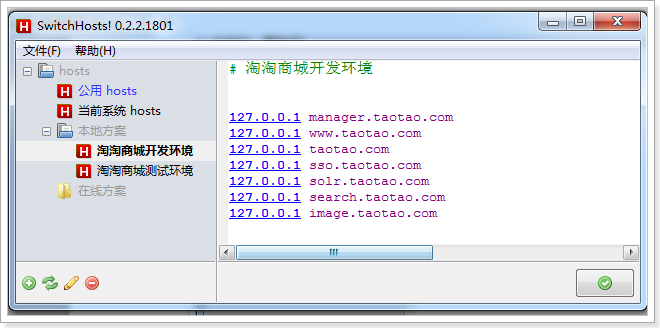


### 使用Nginx配置图片服务器

修改Nginx配置文件



修改图片服务器使用域名访问



### 查看效果

使用后台管理系统，选择编辑商品，查看图片回显



## 把商品导入到索引库中

### 导入分析

最终我们要使用solr查询商品数据，需要把商品数据导入到solr索引库中，之前使用插件导入，这次使用java代码导入。

编写测试方法，实现数据导入到索引库。

### 导入数据到索引库

1. 获取数据，使用Mapper来获取
2. 导入数据到solr索引库中，使用CloudSolrServer导入数据

实现数据导入

**public** **class** ItemData {

**private** CloudSolrServer cloudSolrServer;

**private** ItemMapper itemMapper;

@Before

**public** **void** init() {

ApplicationContext context = **new** ClassPathXmlApplicationContext("spring/applicationContext-\*.xml");

// 从容器中获取声明的bean

**this**.cloudSolrServer = context.getBean(CloudSolrServer.**class**);

**this**.itemMapper = context.getBean(ItemMapper.**class**);

}

@Test

**public** **void** testItemData() **throws** Exception {

// 可以使用循环来获取所有的商品数据

**int** i = 1, pagesize = 500;

**do** {

// 分页查询商品数据

PageHelper.*startPage*(i, 500);

List<Item> list = **this**.itemMapper.select(**null**);

// 遍历结果，把商品放到索引库中

List<SolrInputDocument> docs = **new** ArrayList<>();

**for** (Item item : list) {

// 把获取的商品数据，批量导入到solr索引库中

SolrInputDocument document = **new** SolrInputDocument();

// 商品id

document.setField("id", item.getId().toString());

// 商品名称

document.setField("item\_title", item.getTitle());

// 商品价格

document.setField("item\_price", item.getPrice());

// 商品图片

**if** (StringUtils.*isNotBlank*(item.getImage())) {

document.setField("item\_image", StringUtils.*split*(item.getImage(), ",")[0]);

} **else** {

document.setField("item\_image", "");

}

// 商品类目id

document.setField("item\_cid", item.getCid());

// 商品状态

document.setField("item\_status", item.getStatus());

// 把Document放到集合中，统一提交

docs.add(document);

}

// 把数据保存在solr索引库中

**this**.cloudSolrServer.add(docs);

**this**.cloudSolrServer.commit();

i++;

pagesize = list.size();

} **while** (pagesize == 500);

}