# 淘淘商城-redis缓存、搜索系统搭建

## 内容复习

轮播图的展示

1. 内容管理的后台：CMS系统
   1. 内容分类的管理
   2. 内容管理，需要指定内容的分类
2. 前台展示内容
   1. 需要服务层发布服务。
   2. Portal调用服务获得数据。
   3. HttpClient调用服务获得数据。
   4. 获得数据后，把数据转换成需要的数据格式传递给jsp。

## 课程计划

1. 在业务逻辑中添加缓存
   1. 缓存可以使用redis作为缓存。
   2. Redis集群
   3. 在java代码中使用redis单机版、集群版
   4. 在业务逻辑中添加缓存
2. 搜索系统的实现
   1. 创建一个搜索的工程
   2. 使用solr实现搜索
   3. Solr集群搭建

## 缓存的添加

### Redis 的单机版

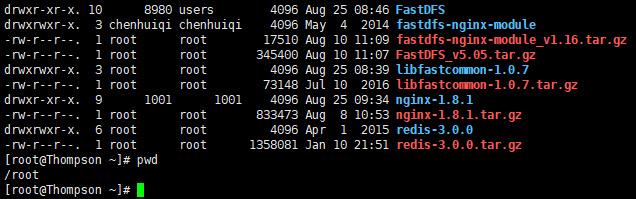
#### 安装步骤：

第一步：安装gcc编译环境

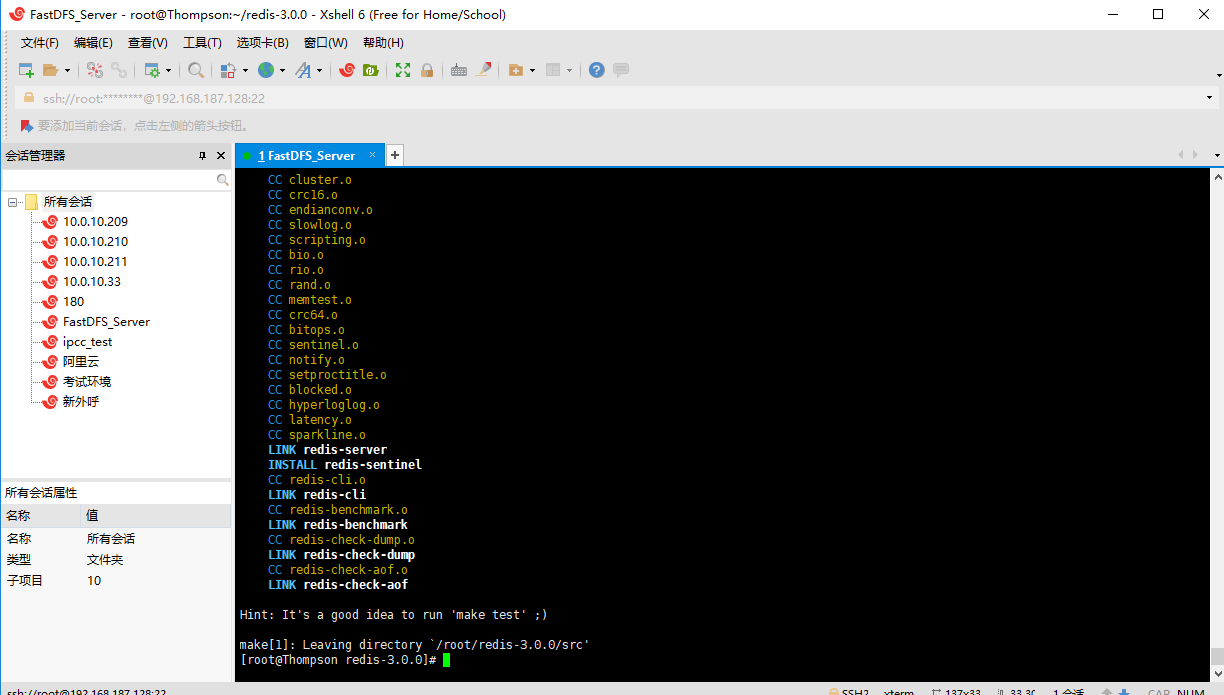
yum install gcc-c++

第二步：把redis的源码上传到linux服务器（cd /root）

第三步：解压缩(tar -zxvf redis-3.0.0.tar.gz)

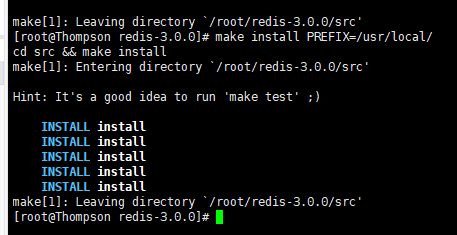


第四步：进入到解压后的redis文件夹中执行make指令

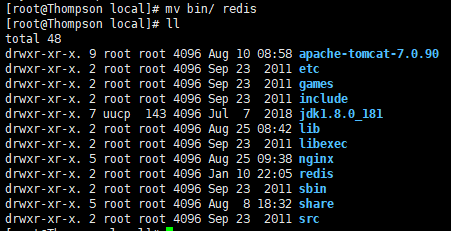


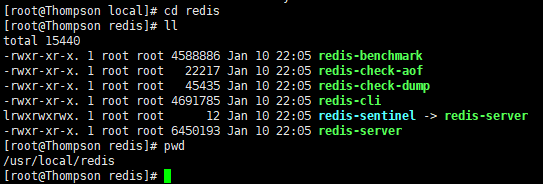
（如果执行失败则可能是gcc环境没有弄好，删除相关文件夹之后重新安装配置）

第五步：指定安装目录：make install PREFIX=/usr/local/redis



（如果执行目录是/usr/local，则默认安装到其对应bin目录下，进入/usr/local/bin目录查看信息。此处指定安装目录为/usr/local/redis）完成安装，（如果换路径，通过移动相关文件夹即可：“mv bin/ redis”）

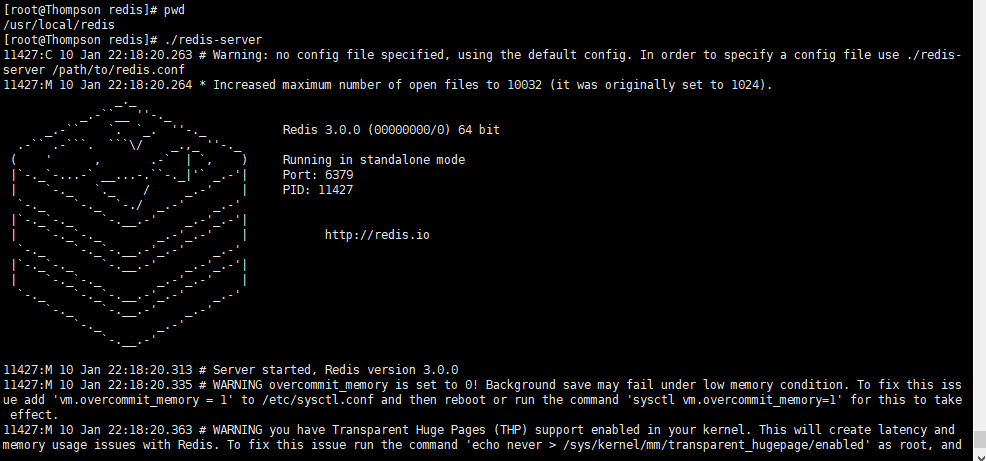




#### 启动redis

两种启动方式，前端启动、后台启动。

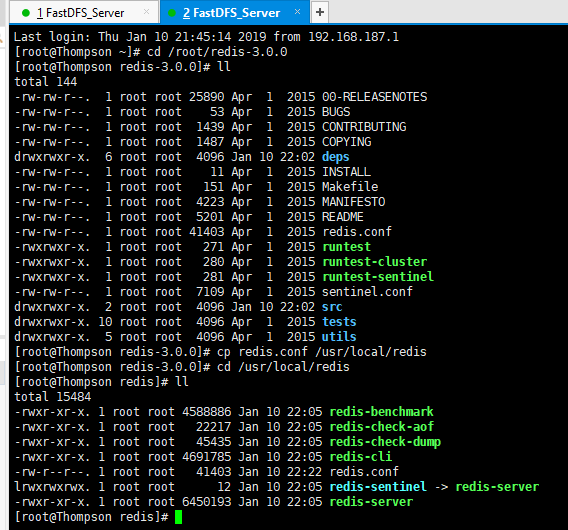
前端启动：./redis-server（进入到redis安装目录，执行指令）



后台启动：

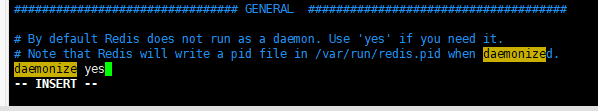
1. 复制redis.conf到redis的安装目录

进入redis解压包将redis.conf文件复制到redis安装目录

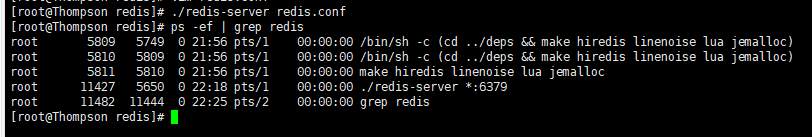


cp redis.conf /usr/local/redis/

1. 修改redis.conf。修改daemonize yes



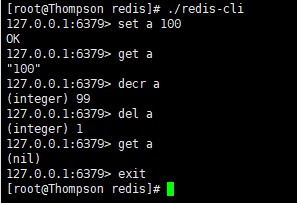
1. [root@bogon redis]# ./redis-server redis.conf



#### 客户端

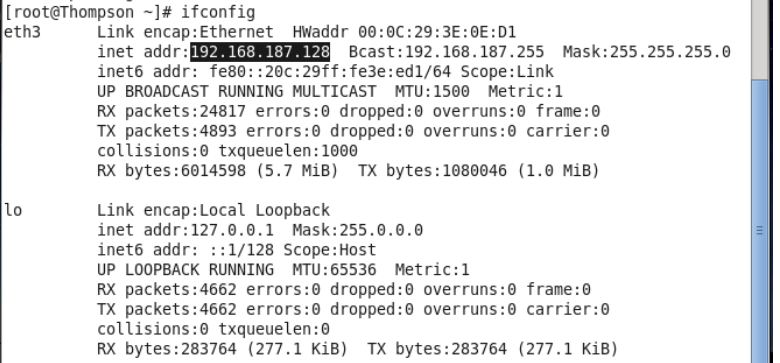
redis-cli -p 端口 -h ip地址 <-c>连接集群时使用此参数

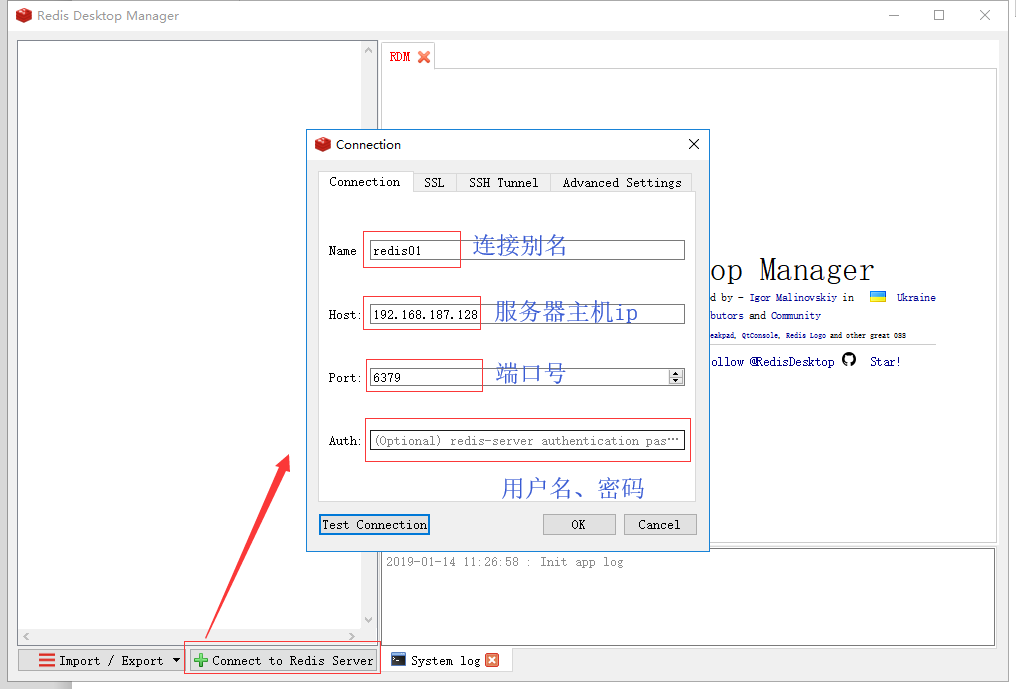
**默认端口：6379 Ip：localhost（127.0.0.1）**



RedisDesktopManager：

虚拟机配置，虚拟ip为192.168.187.128，如果连接失败，则需要查看出错原因，例如防火墙是否关闭？redis.conf配置文件“#bind 127.0.0.1是否已注释”、“是否配置requirepass密码”等，一一进行排除





但此工具只能在单机版环境使用，不支持redis集群。

### Redis集群

#### redis-cluster架构图



架构细节:

(1)所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽.

(2)节点的fail是通过集群中超过半数的节点检测失效时才生效.

(3)客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可

(4)redis-cluster把所有的物理节点映射到[0-16383]slot上,cluster 负责维护node<->slot<->value

Redis 集群中内置了 16384 个哈希槽，当需要在 Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点

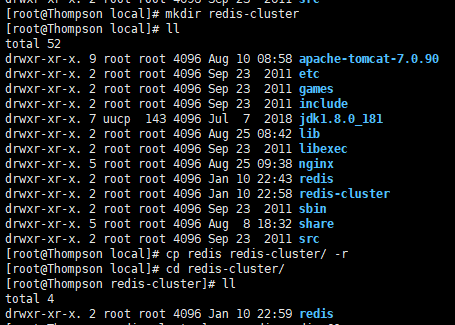
#### 集群搭建

集群中应该至少有三个节点，每个节点有一备份节点。需要6台服务器。

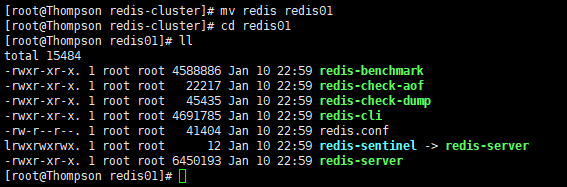
搭建**伪分布式**，需要6个redis实例。

搭建集群的步骤：

第一步：创建6个redis实例指定端口从7001到7006



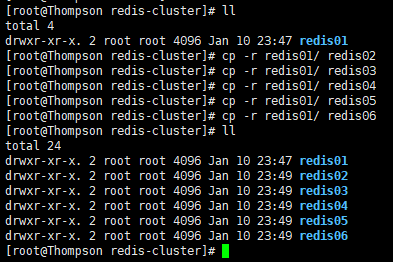
修改redis名称，进入redis01文件夹下，如果存在dump.rdb文件（快照文件）,删除该文件。



第二步：修改redis.conf 配置文件，修改端口号，打开Cluster-enable yes前面的注释。

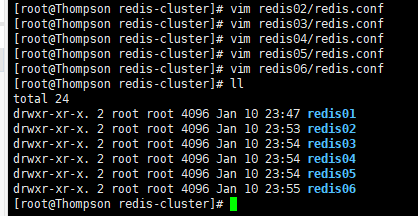
修改redis01的端口号为7001，设置cluster-enable为yes

以此类推，复制多个redis



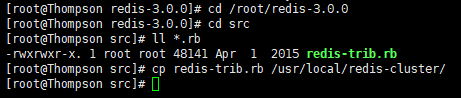
进入到每个redis文件夹中，修改相应的端口号：依次对应

redis01-7001、redis02-7002、redis03-7003、redis04-7004、redis05-7005、redis06-7006



第三步：需要一个ruby脚本。在redis源码文件夹下的src目录下。redis-trib.rb

第四步：把redis-trib.rb文件复制到到redis-cluster目录下。

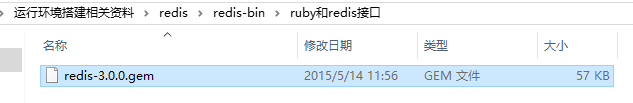


第五步：执行ruby脚本之前，需要安装ruby环境。

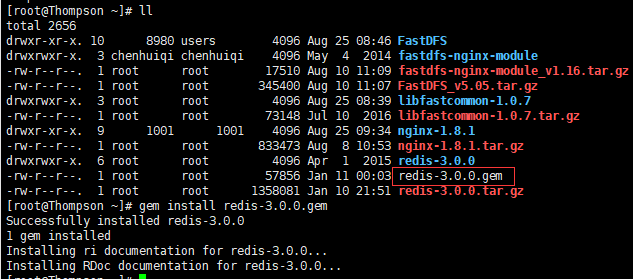
1、yum install ruby

2、yum install rubygems

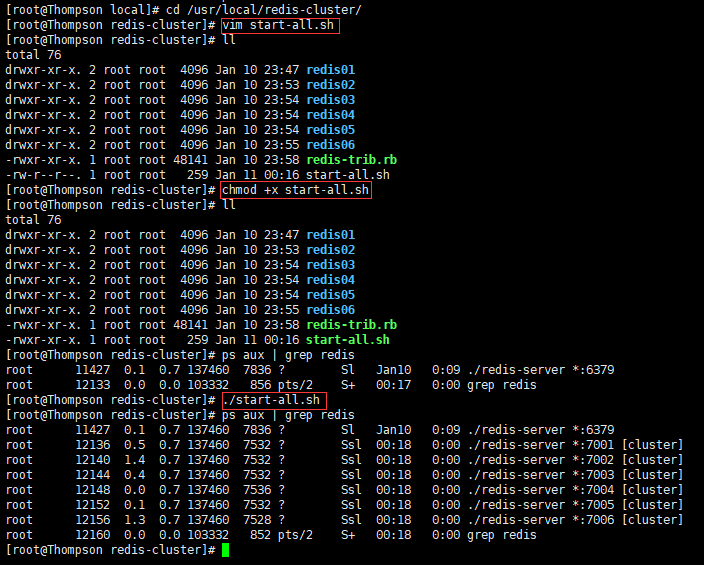
3、安装redis-trib.rb运行依赖的ruby的包（将相关包上传到服务器）

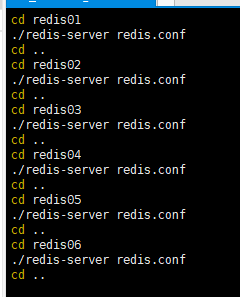


[root@bogon ~]# gem install redis-3.0.0.gem



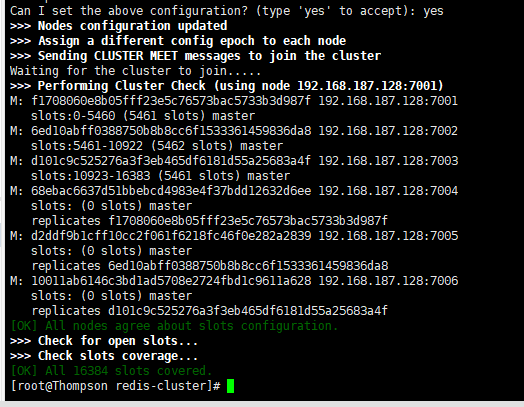
第六步：启动所有的redis实例



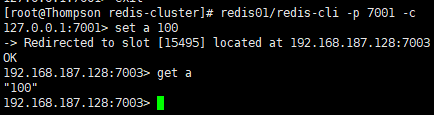


第七步：使用redis-trib.rb创建集群（将对应ip修改为指定的服务器ip）

|  |
| --- |
| ./redis-trib.rb create --replicas 1 192.168.187.128:7001 192.168.187.128:7002 192.168.187.128:7003 192.168.187.128:7004 192.168.187.128:7005 192.168.187.128:7006 |
|  |



使用客户端连接集群： **redis01/redis-cli -p 7001 -c**



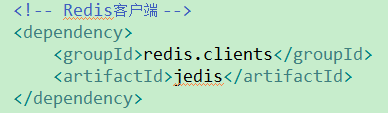
如果没有-c后缀则无法跳转

### 如何使用redis的java客户端

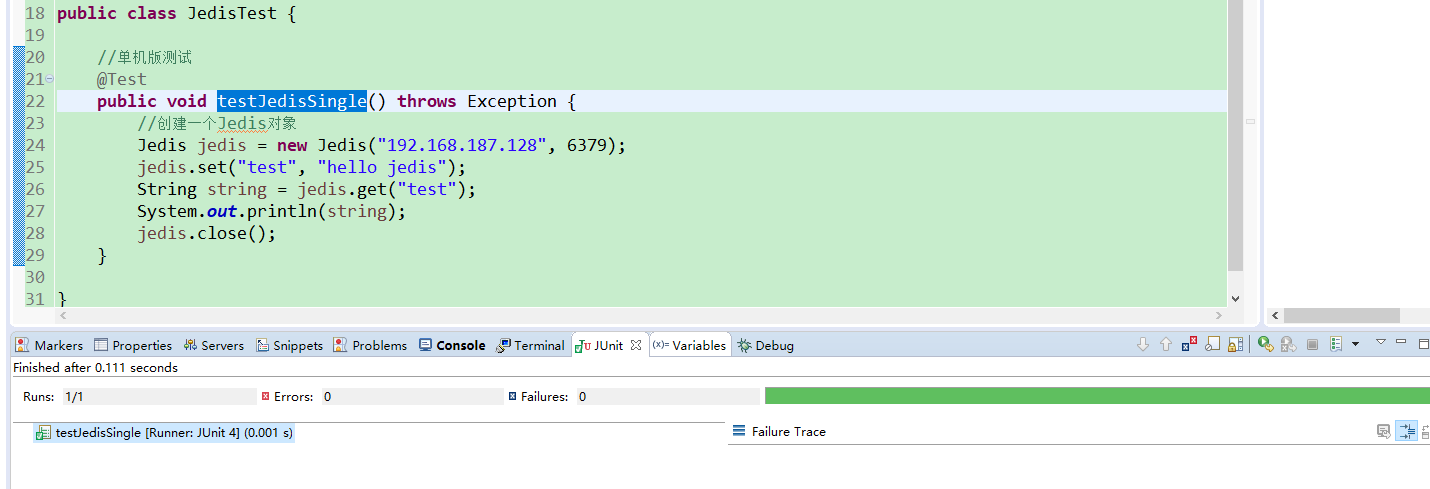
需要使用Jedis连接redis服务器。

#### 连接单机版

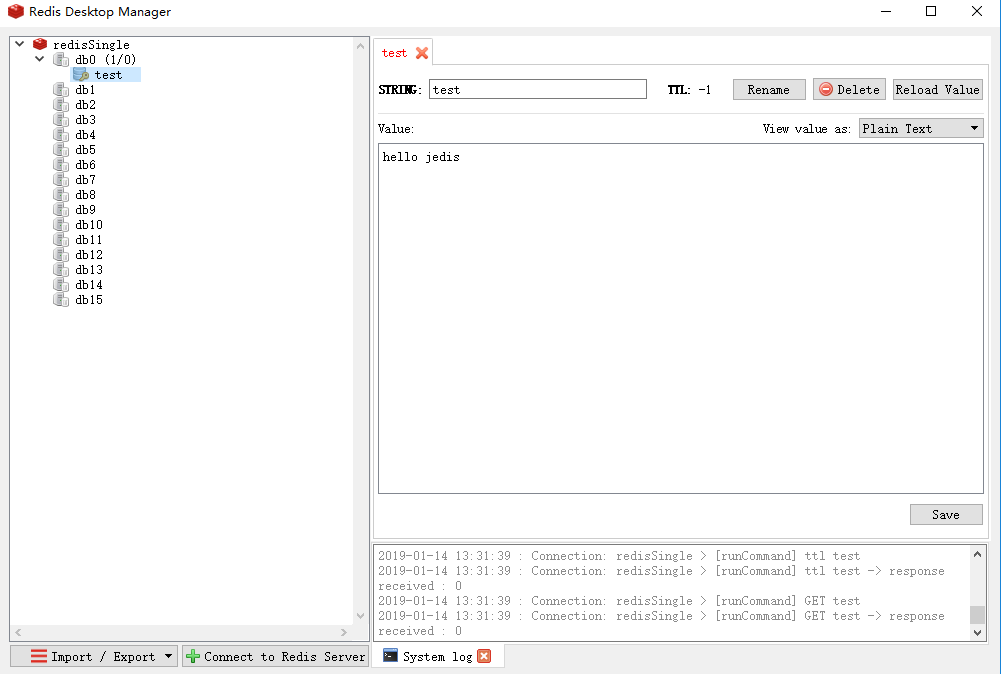
先把jedis依赖的jar包添加到工程中。在taotao-rest工程下的pom.xml中添加jedis依赖



|  |
| --- |
| //单机版测试  @Test  **public** **void** testJedisSingle() **throws** Exception {  //创建一个Jedis对象  Jedis jedis = **new** Jedis("192.168.25.153", 6379);  jedis.set("test", "hello jedis");  String string = jedis.get("test");  System.***out***.println(string);  jedis.close();  } |



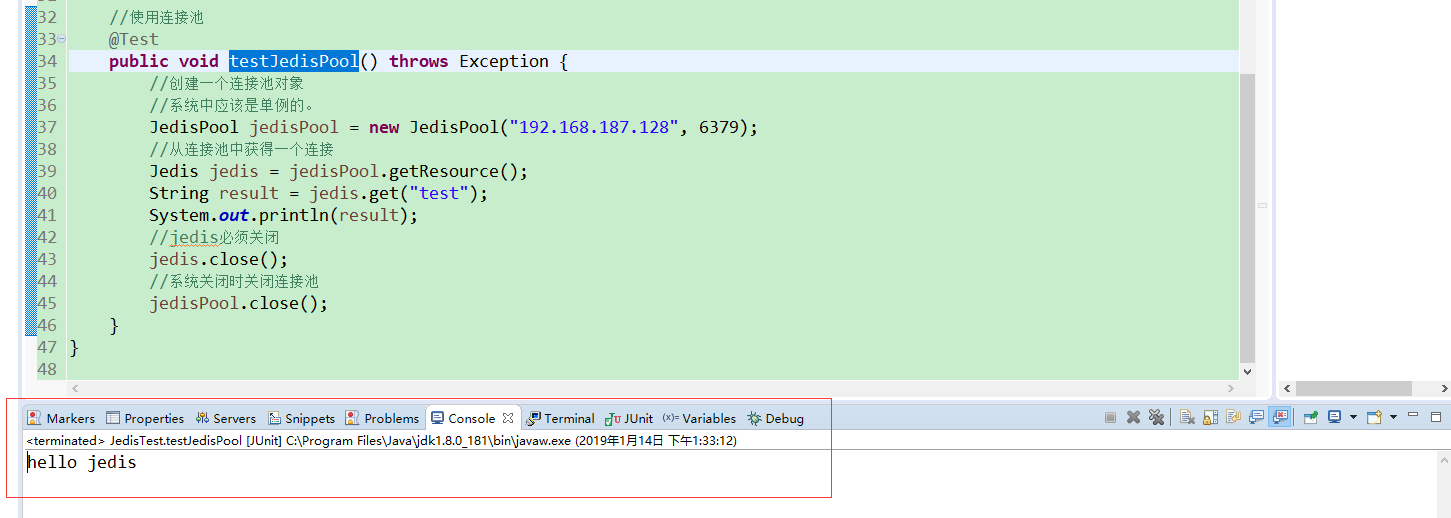
查看测试结果:



使用连接池：

|  |
| --- |
| //使用连接池  @Test  **public** **void** testJedisPool() **throws** Exception {  //创建一个连接池对象  //系统中应该是单例的。  JedisPool jedisPool = **new** JedisPool("192.168.25.153", 6379);  //从连接池中获得一个连接  Jedis jedis = jedisPool.getResource();  String result = jedis.get("test");  System.***out***.println(result);  //jedis必须关闭  jedis.close();    //系统关闭时关闭连接池  jedisPool.close();    } |

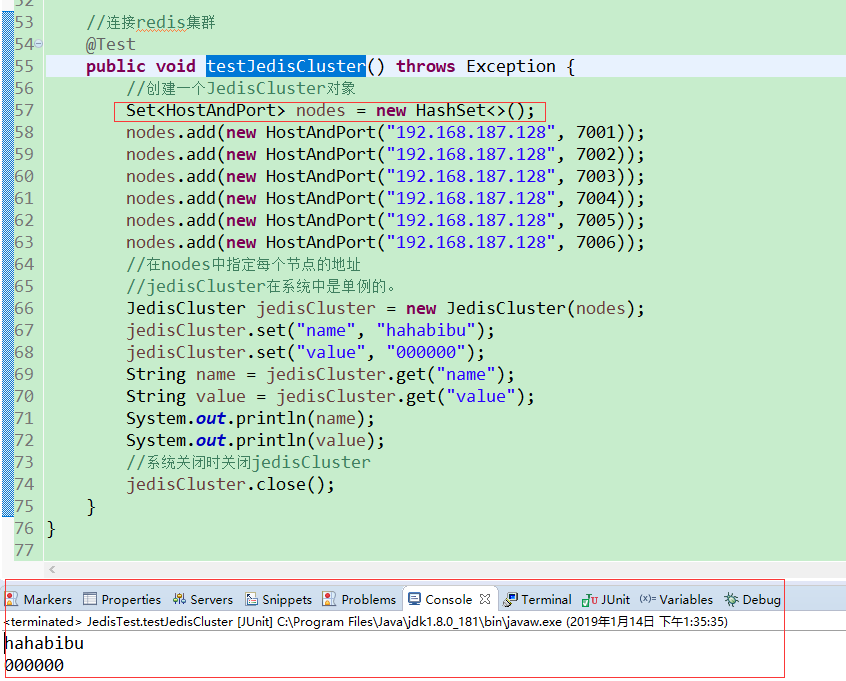
测试结果：



#### 集群版使用Jedis

|  |
| --- |
| //连接redis集群  @Test  **public** **void** testJedisCluster() **throws** Exception {  //创建一个JedisCluster对象  Set<HostAndPort> nodes = **new** HashSet<>();  nodes.add(**new** HostAndPort("192.168.25.153", 7001));  nodes.add(**new** HostAndPort("192.168.25.153", 7002));  nodes.add(**new** HostAndPort("192.168.25.153", 7003));  nodes.add(**new** HostAndPort("192.168.25.153", 7004));  nodes.add(**new** HostAndPort("192.168.25.153", 7005));  nodes.add(**new** HostAndPort("192.168.25.153", 7006));  //在nodes中指定每个节点的地址  //jedisCluster在系统中是单例的。  JedisCluster jedisCluster = **new** JedisCluster(nodes);  jedisCluster.set("name", "zhangsan");  jedisCluster.set("value", "100");  String name = jedisCluster.get("name");  String value = jedisCluster.get("value");  System.***out***.println(name);  System.***out***.println(value);      //系统关闭时关闭jedisCluster  jedisCluster.close();  } |

测试结果：



### 项目中使用jedis

思路：创建一个redis操作的接口。分别创建两个实现类对应redis 的单机版和集群版。当使用单机版redis时，配置单机版的实现类，当使用集群版本的时候，配置集群版的实现类

接口：

**public** **interface** JedisClient {

**public** String set(String key,String value);

**public** String get(String key);

**public** Long hset(String key,String item,String value);

**public** String hget(String key,String item);

**public** Long incr(String key);

**public** Long decr(String key);

**public** Long expire(String key,**int** second);

**public** Long ttl(String key);

**public** Long hdel(String key,String item);

}

#### 单机版实现类

|  |
| --- |
| **package** com.taotao.rest.component.impl;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** redis.clients.jedis.Jedis;  **import** redis.clients.jedis.JedisPool;  **public** **class** JedisClientSingle **implements** JedisClient{  @Autowired  **private** JedisPool jedisPool;  @Override  **public** String set(String key, String value) {  Jedis jedis = jedisPool.getResource();  String result = jedis.set(key, value);  jedis.close();  **return** result;  }  @Override  **public** String get(String key) {  Jedis jedis = jedisPool.getResource();  String result = jedis.get(key);  jedis.close();  **return** result;  }  @Override  **public** Long hset(String key, String item, String value) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.hset(key, item, value);  jedis.close();  **return** result;  }  @Override  **public** String hget(String key, String item) {  Jedis jedis = jedisPool.getResource();  String result = jedis.hget(key, item);  jedis.close();  **return** result;  }  @Override  **public** Long incr(String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.incr(key);  jedis.close();  **return** result;  }  @Override  **public** Long decr(String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.decr(key);  jedis.close();  **return** result;  }  @Override  **public** Long expire(String key, **int** second) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.expire(key, second);  jedis.close();  **return** result;  }  @Override  **public** Long ttl(String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.ttl(key);  jedis.close();  **return** result;  }  @Override  **public** Long hdel(String key, String item) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.hdel(key, item);  jedis.close();  **return** result;  }  } |

#### 集群版

|  |
| --- |
| **package** com.taotao.rest.component.impl;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** com.taotao.rest.component.JedisClient;  **import** redis.clients.jedis.JedisCluster;  **public** **class** JedisClientCluster **implements** JedisClient {  @Autowired  **private** JedisCluster jedisCluster;  @Override  **public** String set(String key, String value) {  **return** jedisCluster.set(key, value);  }  @Override  **public** String get(String key) {  **return** jedisCluster.get(key);  }  @Override  **public** Long hset(String key, String item, String value) {  **return** jedisCluster.hset(key, item, value);  }  @Override  **public** String hget(String key, String item) {  **return** jedisCluster.hget(key, item);  }  @Override  **public** Long incr(String key) {  **return** jedisCluster.incr(key);  }  @Override  **public** Long decr(String key) {  **return** jedisCluster.decr(key);  }  @Override  **public** Long expire(String key, **int** second) {  **return** jedisCluster.expire(key, second);  }  @Override  **public** Long ttl(String key) {  **return** jedisCluster.ttl(key);  }  @Override  **public** Long hdel(String key, String item) {  **return** jedisCluster.hdel(key, item);  }  } |

#### Spring的配置（ip：192.168.187.128）

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <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"*>  <!-- 包扫描器，扫描带@Service注解的类 -->  <context:component-scan base-package=*"com.taotao.rest.service"*></context:component-scan>  <!-- 配置redis客户端单机版 -->  <bean id=*"jedisPool"* class=*"redis.clients.jedis.JedisPool"*>  <constructor-arg name=*"host"* value=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"6379"*></constructor-arg>  </bean>  <!-- 配置redis客户端实现类 -->  <bean id=*"jedisClientSingle"* class=*"com.taotao.rest.component.impl.JedisClientSingle"*/>    <!-- 配置redis客户端集群版 -->  <!-- <bean id="jedisCluster" class="redis.clients.jedis.JedisCluster">  <constructor-arg>  <set>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7001"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7002"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7003"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7004"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7005"/>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"/>  <constructor-arg name="port" value="7006"/>  </bean>  </set>  </constructor-arg>  </bean>  <bean id="jedisClientCluster" class="com.taotao.rest.component.impl.JedisClientCluster"/> -->  </beans> |

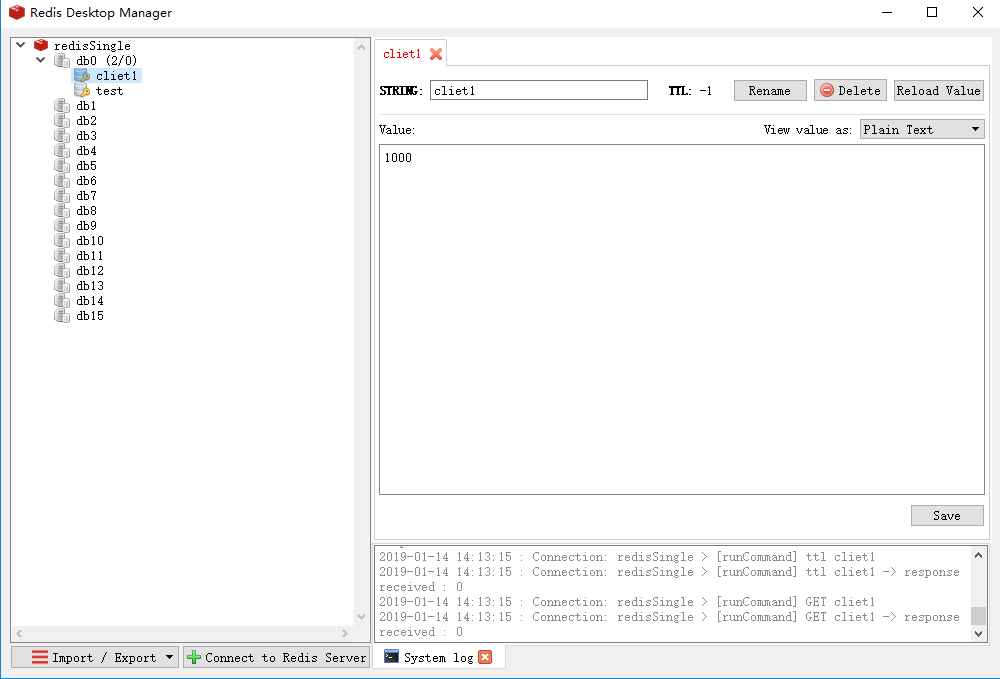
测试：

|  |
| --- |
| @Test  **public** **void** testJedisClientSpring() **throws** Exception {  //创建一个spring容器  ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");  //从容器中获得JedisClient对象  JedisClient jedisClient = applicationContext.getBean(JedisClient.**class**);  //jedisClient操作redis  jedisClient.set("cliet1", "1000");  String string = jedisClient.get("cliet1");  System.***out***.println(string);  } |

测试结果：



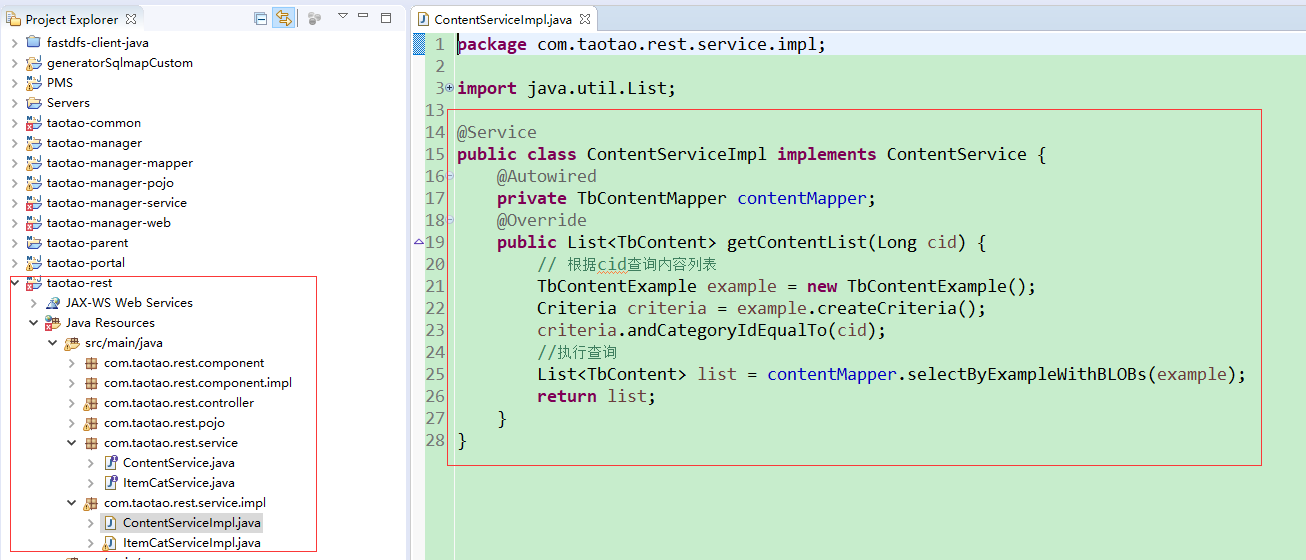
在可视化工具中查看数据：

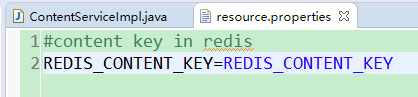


集群版切换，修改applicationContext-service.xml文件，将单机版配置注释，打开集群注释，再次测试即可！

### 业务逻辑中添加缓存

注意：添加缓存时不要影响正常的业务逻辑。





|  |
| --- |
| **package** com.taotao.rest.service.impl;  **import** java.util.List;  **import** org.apache.commons.lang3.StringUtils;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** com.taotao.common.utils.JsonUtils;  **import** com.taotao.mapper.TbContentMapper;  **import** com.taotao.pojo.TbContent;  **import** com.taotao.pojo.TbContentExample;  **import** com.taotao.pojo.TbContentExample.Criteria;  **import** com.taotao.rest.component.JedisClient;  **import** com.taotao.rest.service.ContentService;  @Service  **public** **class** ContentServiceImpl **implements** ContentService {  @Autowired  **private** JedisClient jedisClient;  @Autowired  **private** TbContentMapper contentMapper;  @Value("${REDIS\_CONTENT\_KEY}")  **private** String REDIS\_CONTENT\_KEY;  @Override  **public** List<TbContent> getContentList(Long cid) {  // 根据cid查询内容列表  //查询数据库之前先查询缓存，如果有直接返回  **try** {  //从redis中取缓存数据  String json = jedisClient.hget(REDIS\_CONTENT\_KEY, cid+"");  **if** (!StringUtils.*isBlank*(json)) {  //把json转换成List  List<TbContent> list = JsonUtils.*jsonToList*(json, TbContent.**class**);  **return** list;  }  } **catch** (Exception e) {  e.printStackTrace();  }  TbContentExample example = **new** TbContentExample();  Criteria criteria = example.createCriteria();  criteria.andCategoryIdEqualTo(cid);  //执行查询  List<TbContent> list = contentMapper.selectByExampleWithBLOBs(example);  //返回结果之前，向缓存中添加数据  **try** {  //为了规范key可以使用hash  //定义一个保存内容的key，hash中每个项就是cid  //value是list，需要把list转换成json数据。  jedisClient.hset(REDIS\_CONTENT\_KEY, cid+"", JsonUtils.*objectToJson*(list));  } **catch** (Exception e) {  e.printStackTrace();  }  **return** list;  }  } |

### 缓存同步

当后台修改内容信息后，只需要把redis中缓存的数据删除即可。

后台系统不直接操作redis数据库。

可以在taotao-rest中发布一个服务，当后台对内容信息修改后，调用服务即可。

服务的功能就是根据cid删除redis中缓存数据。

#### 发布服务

##### Dao层

使用JedisClient。

##### Service层

接收cid，根据cid调用JedisClient删除redis中缓存的数据。返回结果TaotaoResult。

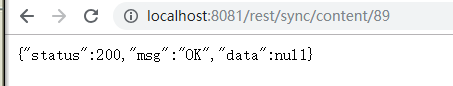
|  |
| --- |
| @Override  **public** TaotaoResult syncContent(Long cid) {  jedisClient.hdel(REDIS\_CONTENT\_KEY, cid + "");  **return** TaotaoResult.*ok*();  } |

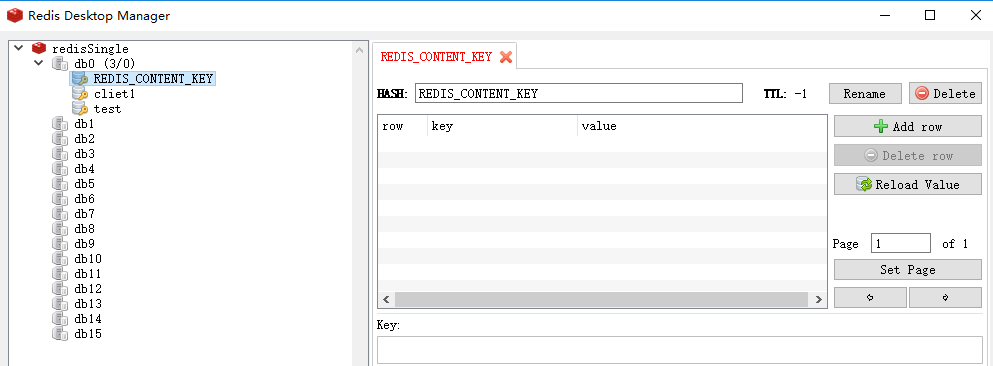
##### Controller层

发布服务，接收参数cid，返回结果TaotaoResult。

|  |
| --- |
| @RequestMapping("/sync/content/{cid}")  @ResponseBody  **public** TaotaoResult sysncContent(@PathVariable Long cid) {  **try** {  TaotaoResult result = contentService.syncContent(cid);  **return** result;  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  } |

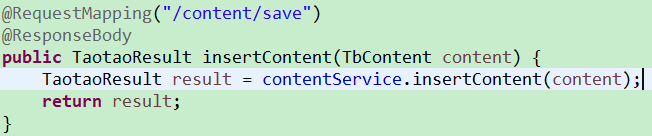
测试：<http://localhost:8081/rest/sync/content/89>

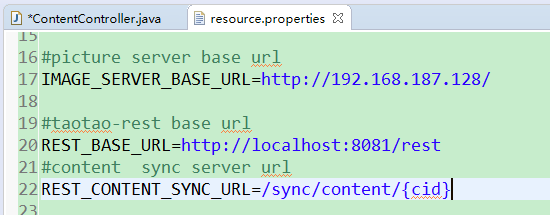




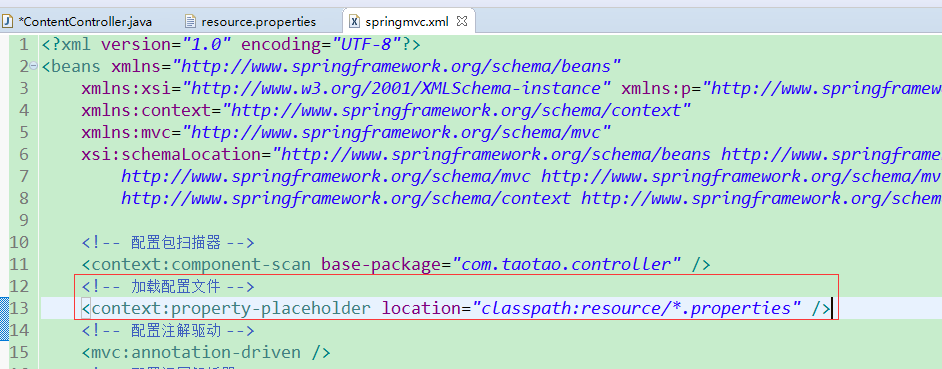
### 调用缓存同步

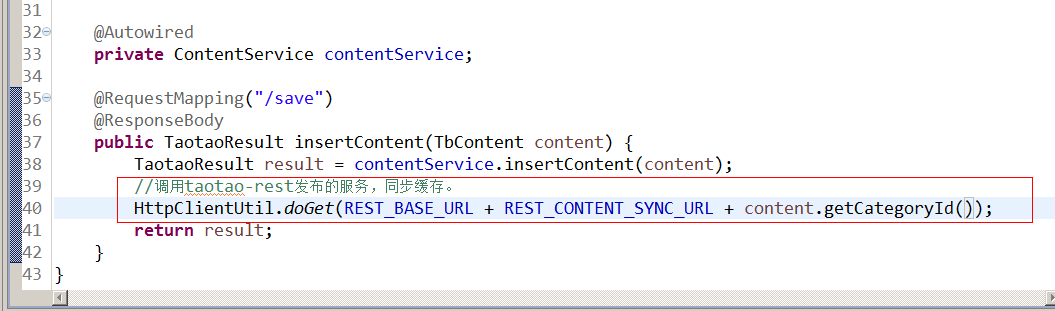
taotao-manager-web后台系统中，只要内容发生变化（增删改操作）均需要调用缓存同步的服务。





此处是在controller获取配置文件的相关属性，因此可以在springmvc中扫描相关配置文件：（或者是在service层提供相应的get方法）





启动taotao-rest、taotao-manager、taotao-portal进行测试（taotao-rest没开则导致redis数据访问失败）

测试：如果redis数据没有及时更新，则首页访问获取的数据是脏数据！

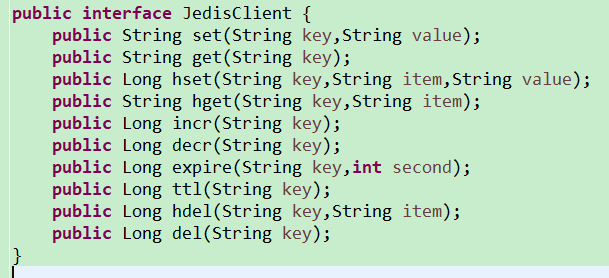
先将REDIS\_CONTENT\_KEY数据删除，随后访问taotao商城首页刷新数据，可以查看到REDIS\_CONTENT\_KEY数据添加到redis缓存中，随后在taotao后台管理系统中添加大广告图片，之后再查看redis的GUI工具可以看到REDIS\_CONTENT\_KEY被清楚，重新访问taotao首页则可看到REDIS\_CONTENT\_KEY又重新被加载到redis缓存！

## 商品类目展示添加缓存-待开发

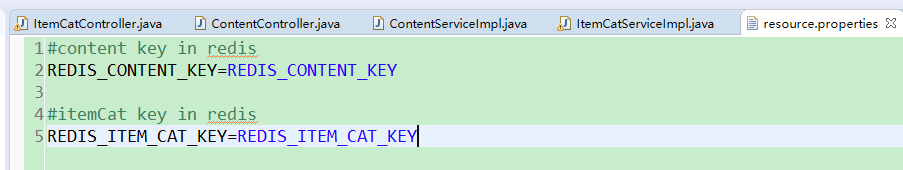
结合上述内容，仿写代码。

### taotao-rest修改

结合需求修改JedisClient接口，封装方法



resource.properties:



商品类目Service接口：

**public** **interface** ItemCatService {

// 获取分类列表

**public** ItemCatResult getItemCatList();

// 同步内容信息

**public** TaotaoResult syncItemCat();

}

商品类目Service实现类：

**package** com.taotao.rest.service.impl;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.apache.commons.lang3.StringUtils;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.stereotype.Service;

**import** com.taotao.common.result.TaotaoResult;

**import** com.taotao.common.utils.JsonUtils;

**import** com.taotao.mapper.TbItemCatMapper;

**import** com.taotao.pojo.TbItemCat;

**import** com.taotao.pojo.TbItemCatExample;

**import** com.taotao.pojo.TbItemCatExample.Criteria;

**import** com.taotao.rest.component.JedisClient;

**import** com.taotao.rest.pojo.CatNode;

**import** com.taotao.rest.pojo.ItemCatResult;

**import** com.taotao.rest.service.ItemCatService;

@Service

**public** **class** ItemCatServiceImpl **implements** ItemCatService {

@Autowired

**private** TbItemCatMapper itemCatMapper;

@Autowired

**private** JedisClient jedisClient;

@Value("${REDIS\_ITEM\_CAT\_KEY}")

**private** String REDIS\_ITEM\_CAT\_KEY;

@Override

**public** ItemCatResult getItemCatList() {

//调用递归方法查询商品分类列表

//查询数据库之前先查询缓存，如果有直接返回

**try** {

//从redis中取商品分类缓存数据

String json = jedisClient.get(REDIS\_ITEM\_CAT\_KEY);

**if** (!StringUtils.*isBlank*(json)) {

//把json转换成List

List list = JsonUtils.*jsonToList*(json, CatNode.**class**);

ItemCatResult result = **new** ItemCatResult();

result.setData(list);

**return** result;

}

} **catch** (Exception e) {

e.printStackTrace();

}

List catList = getItemCatList(0l);

//返回结果之前，向缓存中添加数据

**try** {

//为了规范key可以使用hash

//定义一个保存内容的key，hash中每个项就是cid

//value是list，需要把list转换成json数据。

jedisClient.set(REDIS\_ITEM\_CAT\_KEY, JsonUtils.*objectToJson*(catList));

} **catch** (Exception e) {

e.printStackTrace();

}

ItemCatResult result = **new** ItemCatResult();

result.setData(catList);

**return** result;

}

**private** List getItemCatList(Long parentId) {

//根据parentId查询列表

TbItemCatExample example = **new** TbItemCatExample();

Criteria criteria = example.createCriteria();

criteria.andParentIdEqualTo(parentId);

//执行查询

List<TbItemCat> list = itemCatMapper.selectByExample(example);

List resultList = **new** ArrayList<>();

// 定义计数器

**int** index = 0;

**for** (TbItemCat tbItemCat : list) {

// 判断index是否到达临界值

**if**(index>=14) {

**break**;

}

//如果是父节点

**if** (tbItemCat.getIsParent()) {

CatNode node = **new** CatNode();

node.setUrl("/products/"+tbItemCat.getId()+".html");

//如果当前节点为第一级节点

**if** (tbItemCat.getParentId() == 0) {

node.setName("<a href='/products/"+tbItemCat.getId()+".html'>"+tbItemCat.getName()+"</a>");

// 第一级节点设置不能超过14

index++;

} **else** {

node.setName(tbItemCat.getName());

}

node.setItems(getItemCatList(tbItemCat.getId()));

//把node添加到列表

resultList.add(node);

} **else** {

//如果是叶子节点

String item = "/products/"+tbItemCat.getId()+".html|" + tbItemCat.getName();

resultList.add(item);

}

}

**return** resultList;

}

@Override

**public** TaotaoResult syncItemCat() {

jedisClient.del(REDIS\_ITEM\_CAT\_KEY);

**return** TaotaoResult.*ok*();

}

}

Controller层修改：

**package** com.taotao.rest.controller;

**import** org.apache.commons.lang3.StringUtils;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.converter.json.MappingJacksonValue;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.ResponseBody;

**import** com.taotao.common.result.TaotaoResult;

**import** com.taotao.common.utils.ExceptionUtil;

**import** com.taotao.rest.pojo.ItemCatResult;

**import** com.taotao.rest.service.ItemCatService;

@Controller

@RequestMapping("/item/cat")

**public** **class** ItemCatController {

@Autowired

**private** ItemCatService itemCatService;

// @RequestMapping(value="/list")

// @ResponseBody

// public ItemCatResult testGetItemCatList() {

// return itemCatService.getItemCatList();

// }

// 方法1

// @RequestMapping(value="/list",produces=MediaType.APPLICATION\_JSON\_VALUE+";charset=utf-8")

// @ResponseBody

// public String getItemCatList(String callback) {

// ItemCatResult result = itemCatService.getItemCatList();

// if (StringUtils.isBlank(callback)) {

// //需要把result转换成字符串

// String json = JsonUtils.objectToJson(result);

// return json;

// }

// //如果字符串不为空，需要支持jsonp调用

// //需要把result转换成字符串

// String json = JsonUtils.objectToJson(result);

// return callback + "(" + json + ");";

// }

// 方法2

@RequestMapping(value="/list")

@ResponseBody

**public** Object getItemCatList(String callback) {

ItemCatResult result = itemCatService.getItemCatList();

**if** (StringUtils.*isBlank*(callback)) {

//需要把result转换成字符串

**return** result;

}

//如果字符串不为空，需要支持jsonp调用

MappingJacksonValue mappingJacksonValue = **new** MappingJacksonValue(result);

mappingJacksonValue.setJsonpFunction(callback);

**return** mappingJacksonValue;

}

@RequestMapping(value="/sync")

@ResponseBody

**public** TaotaoResult sysncItemCat() {

**try** {

TaotaoResult result = itemCatService.syncItemCat();

**return** result;

} **catch** (Exception e) {

e.printStackTrace();

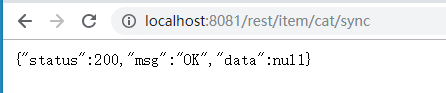
**return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));

}

}

}

测试同步：<http://localhost:8081/rest/item/cat/sync>



### Taotao-manager-web修改

此处还没有设计商品分类内容的调整，此处待定！

## 搜索系统搭建

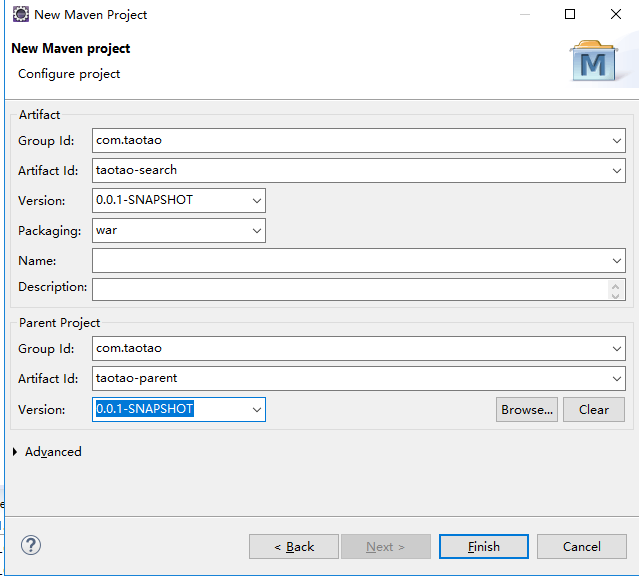
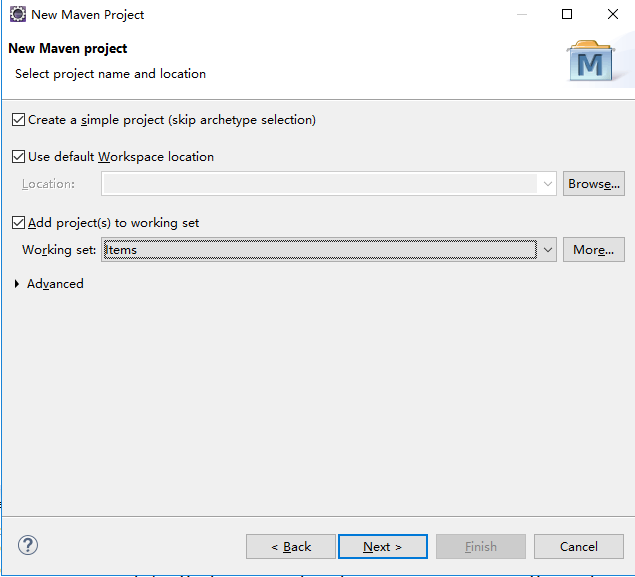
搜索功能需要发布服务共pc端、移动端使用。根据关键词搜索，得到json格式的搜索结果。创建一个搜索系统，发布搜索服务。

系统架构：  


* 创建一个搜索工程：taotao-search
* 搭建solr服务

### 创建搜索系统

可以参考taotao-rest创建



使用的技术：

* Mybatis
* Spring
* Springmvc（发布服务）
* SolrJ（solr服务的客户端）

### Pom文件参考

|  |
| --- |
| <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</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <dependencies>  <dependency>  <groupId>com.taotao</groupId>  <artifactId>taotao-manager-dao</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <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>  <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>  <!-- solr客户端 -->  <dependency>  <groupId>org.apache.solr</groupId>  <artifactId>solr-solrj</artifactId>  </dependency>  </dependencies>  <!-- 添加tomcat插件 -->  <build>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8083</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

### POM文件：

<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>war</packaging>

<dependencies>

<dependency>

<groupId>com.taotao</groupId>

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

<version>0.0.1-SNAPSHOT</version>

</dependency>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

</dependency>

<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>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<scope>provided</scope>

</dependency>

<!-- solr客户端 -->

<dependency>

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

<artifactId>solr-solrj</artifactId>

</dependency>

</dependencies>

<!-- 添加tomcat插件 -->

<build>

<pluginManagement>

<plugins>

<plugin>

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

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

<configuration>

<port>8083</port>

<path>/</path>

</configuration>

</plugin>

<plugin>

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

<artifactId>maven-war-plugin</artifactId>

<configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

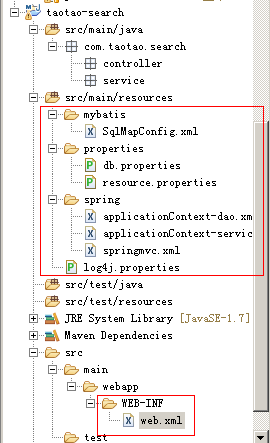
</pluginManagement>

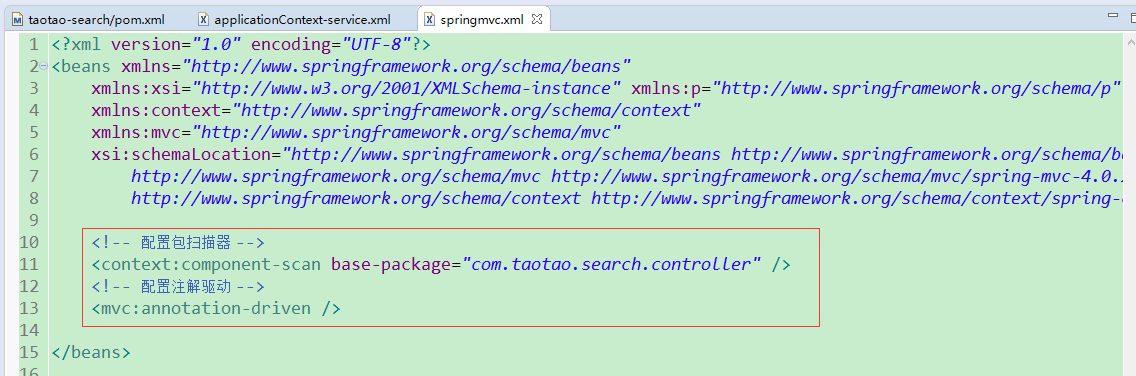
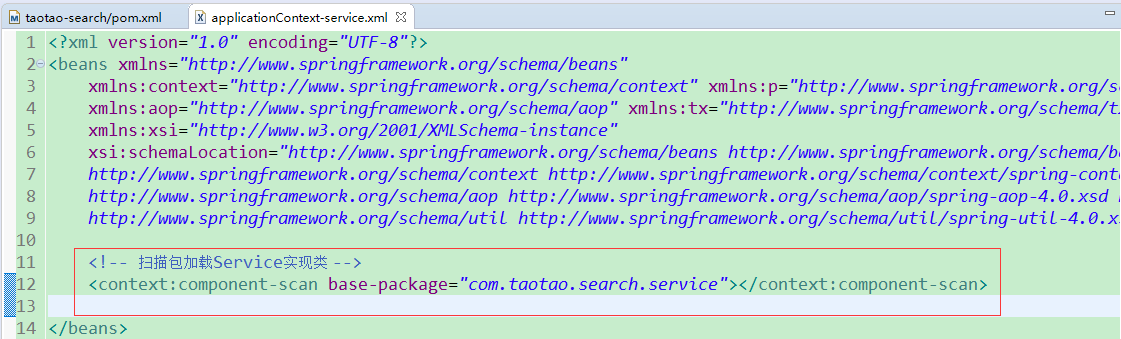
</build>

</project>

### 框架整合

ssm整合：

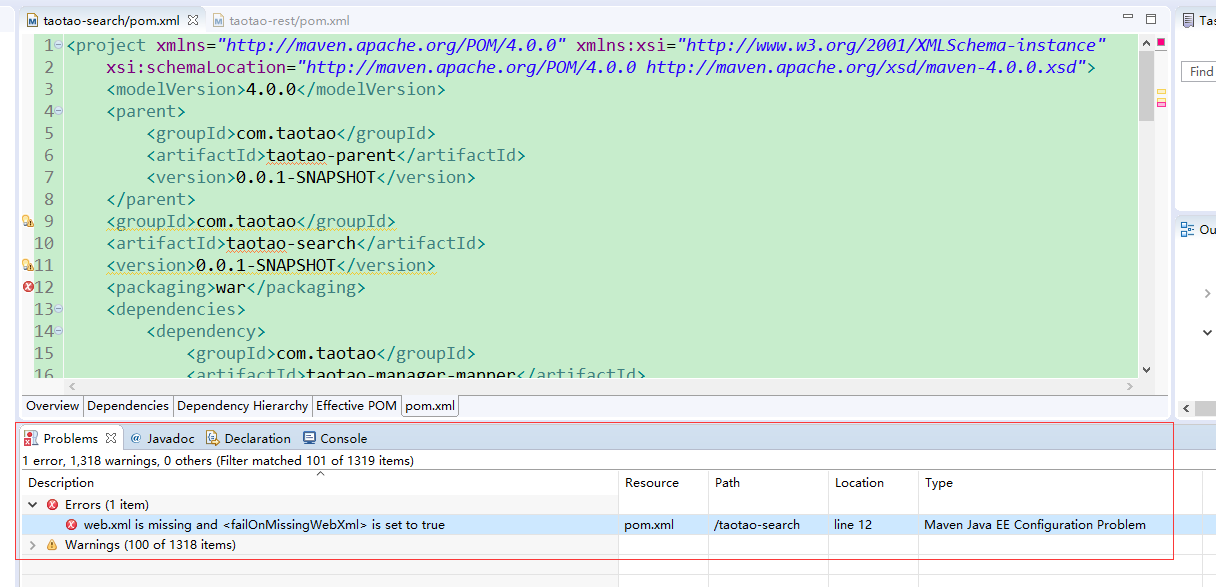




web.xml配置：

taotao-rest相关修改为相应的taotao-search对应的内容

#### 问题分析：



解决方法1：如果工程不是web项目则可在pom.xml中添加配置（配置failOnMissingWebXml）

<build>

  <plugins>

   <plugin>

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

    <artifactId>maven-war-plugin</artifactId>

    <version>2.6</version>

    <configuration>

     <failOnMissingWebXml>false</failOnMissingWebXml>

    </configuration>

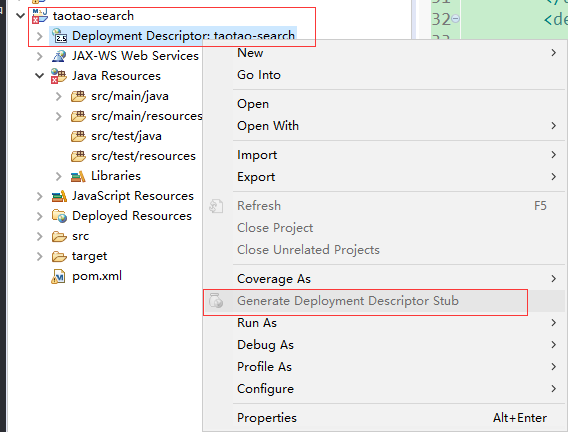
   </plugin>

  </plugins>

 </build>



解决方法2：针对web项目解决：在视图Project Explorer中操作，右击项目——>Java EE Tools——>Generate Deployment Descriptor Stub.然后系统会在src/main/webapp/WEB\_INF文件加下创建web.xml文件。



### Solr服务的搭建

需要在linux系统下搭建solr服务。

* 需要安装tomcat
* 安装jdk

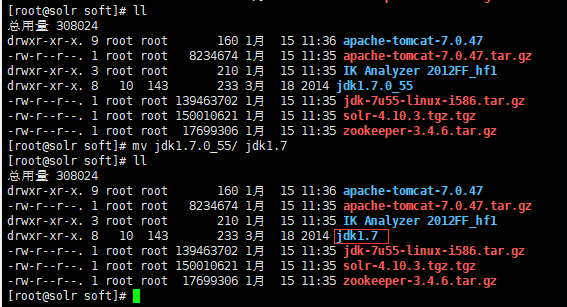
#### CentOS单机版安装：

##### 第一步：安装jdk、安装tomcat

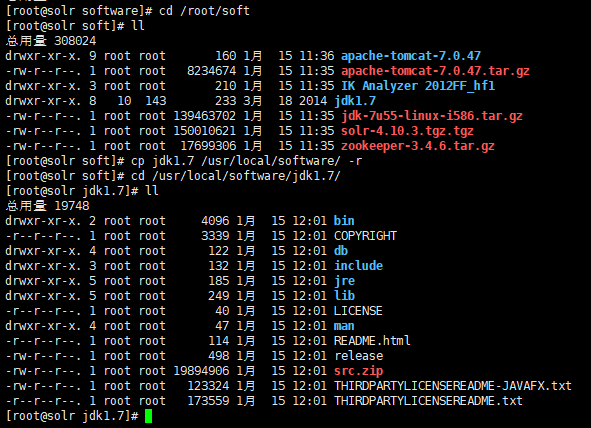
在/root/soft下导入相关的压缩包，安装jdk、tomcat

###### Jdk安装

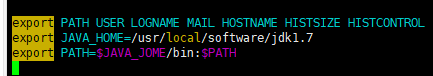
tar -zxvf jdk-7u55-linux-i586.tar.gz



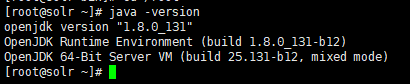
将jdk复制到/usr/local/software目录下(手动创建)



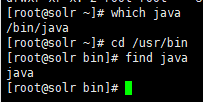
通过vim /etc/profile指令修改配置（jdk配置：配置环境变量）



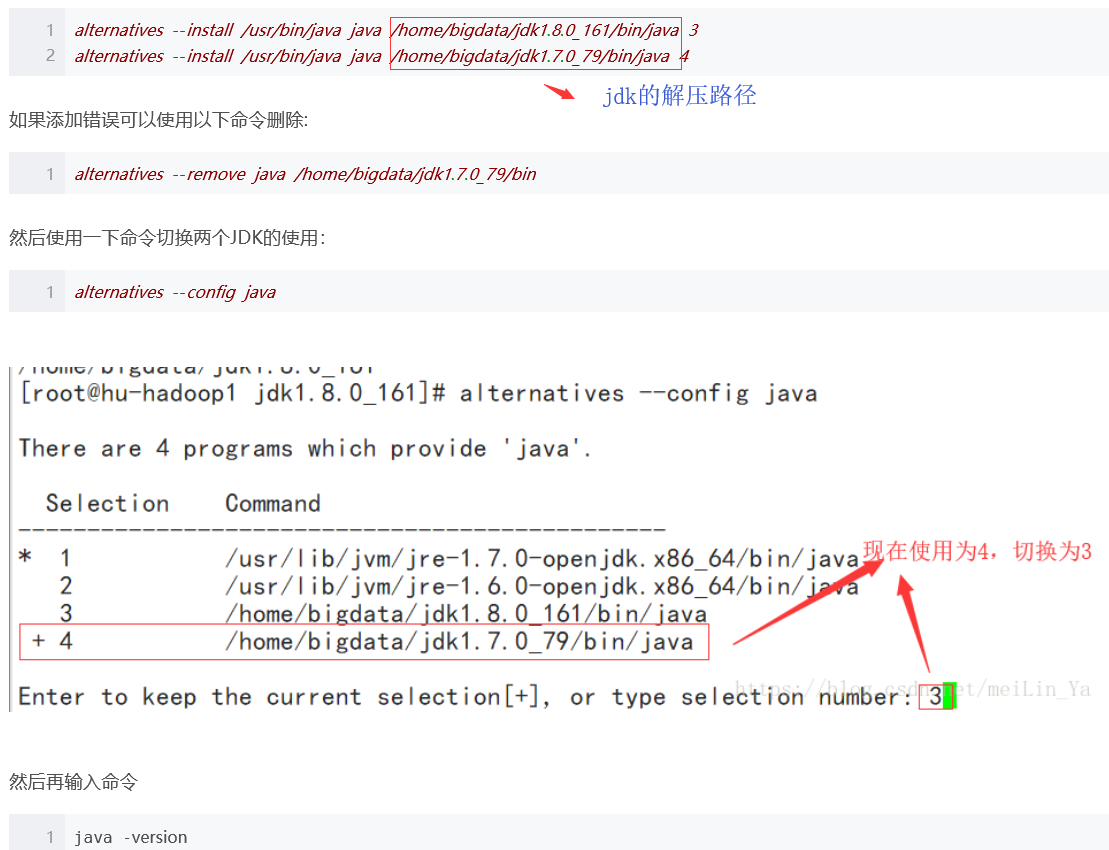
保存配置，通过source /etc/profile指令使配置文件生效，通过java –version查看当前配置的jdk版本



配置完成，发现使用的是默认的jdk版本，而不是配置的jdk1.7，因此此处需要通过“which java”查看当前使用的是哪个路径的java，参考相关教程修正



（此处存在问题，待解决！）



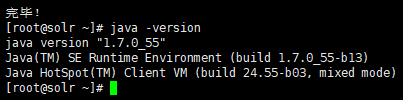
参考链接：<https://blog.csdn.net/meilin_ya/article/details/80650945>

错误描述:安装好jdk之后,通过java -version,javac,java等命令测试是否安装成功时出现错误

-bash: /usr/java/jdk1.7.0\_71/bin/java: /lib/ld-linux.so.2: bad ELF interpreter: No such file or directory

错误原因:没有那个文件或目录,需要安装glibc

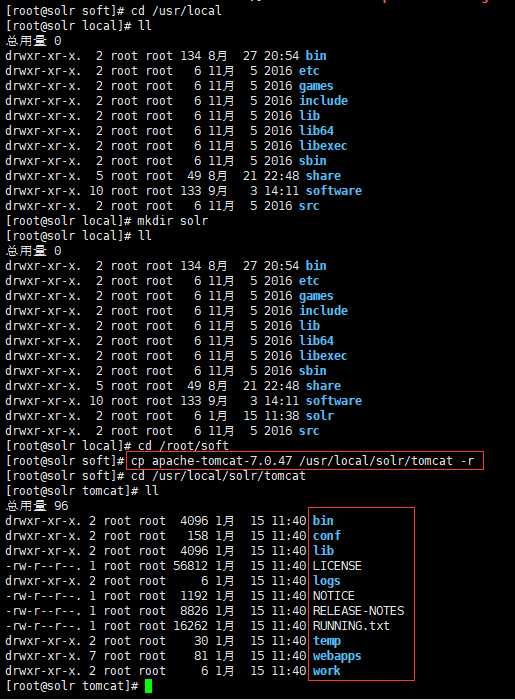
解决办法:终端输入sudo yum install glibc.i686命令,安装好glibc,问题解决



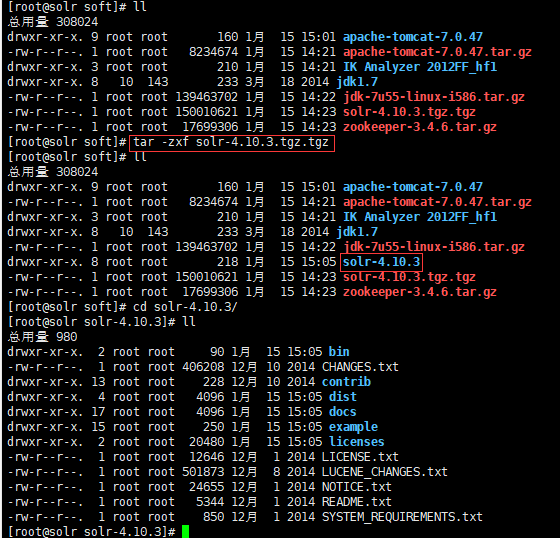
###### Tomcat安装

tar -zxf apache-tomcat-7.0.47.tar.gz

**在/usr/local/目录下创建solr目录，将tomcat复制到该目录**

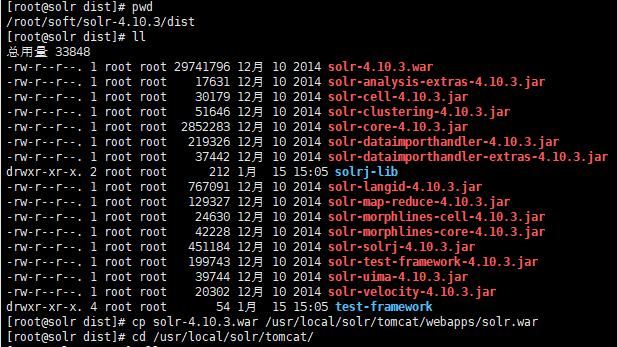


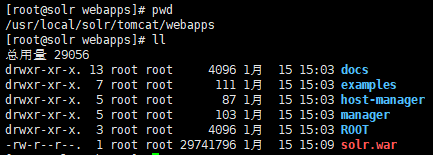
##### 第二步：解压solr压缩包



##### 第三步：把dist/solr-4.10.3.war部署到tomcat下

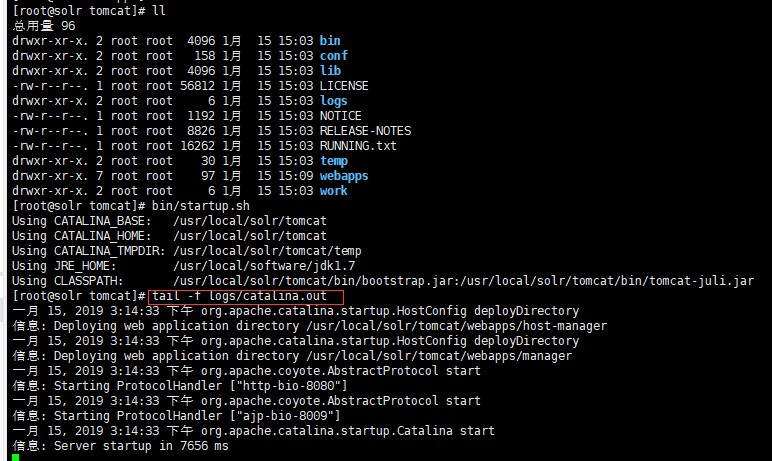
将解压的安装目录下的dist/solr-4.10.3.war服务器部署到tomcat下



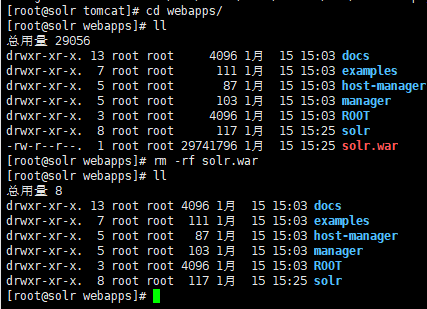


##### 第四步：解压缩war包，启动tomcat解压

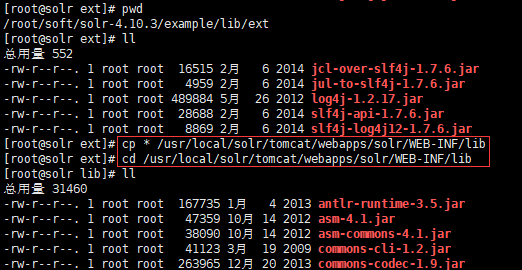
进入到/usr/local/solr/tomcat/bin目录执行“./startup.sh”指令



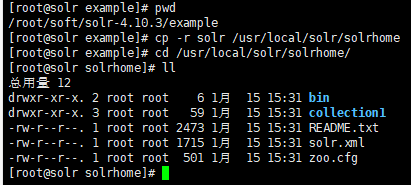
解压完成，可以将原来的solr.war包删除



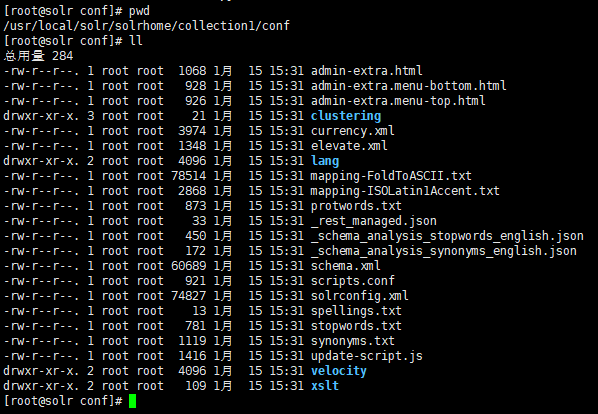
##### 第五步：需要把/root/soft/solr-4.10.3/example/lib/ext目录下的所有的jar包添加到solr工程中



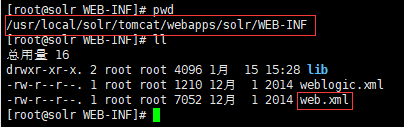
##### 第六步：创建solrhome。把/root/solr-4.10.3/example/solr文件夹复制一份作为solrhome。复制到：/usr/local/solr/solrhome

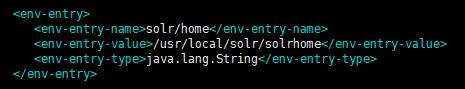


##### 第七步：告诉solr服务solrhome的位置。需要修改web.xml



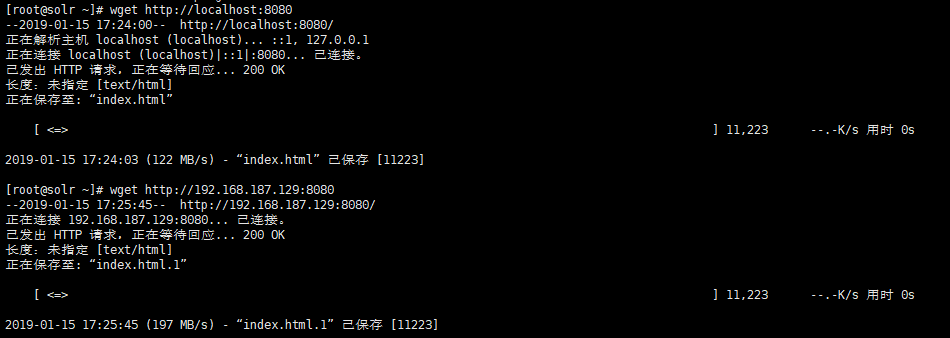
修改/usr/local/solr/tomcat/webapps/solr/WEB-INF下的web.xml文件





##### 第八步：启动tomcat

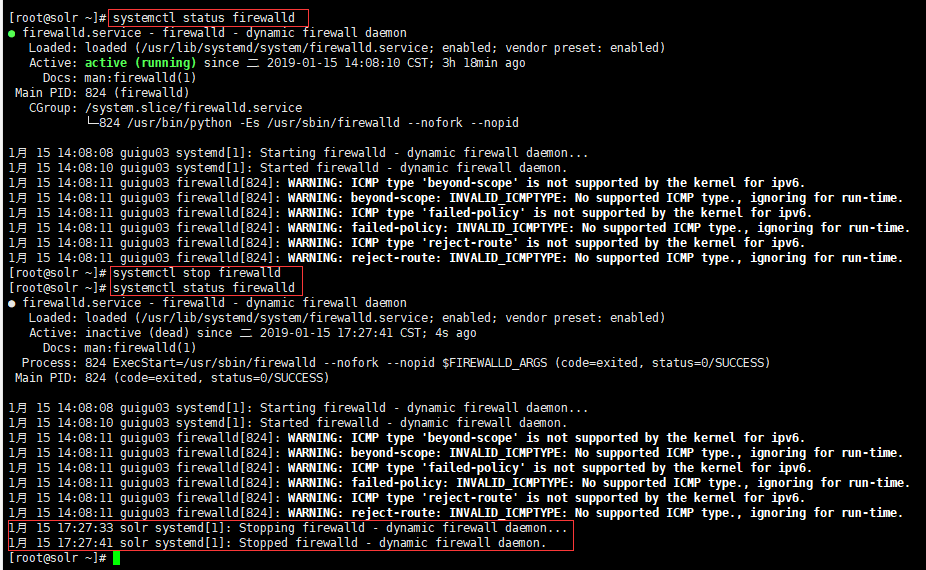
启动tomcat进行测试



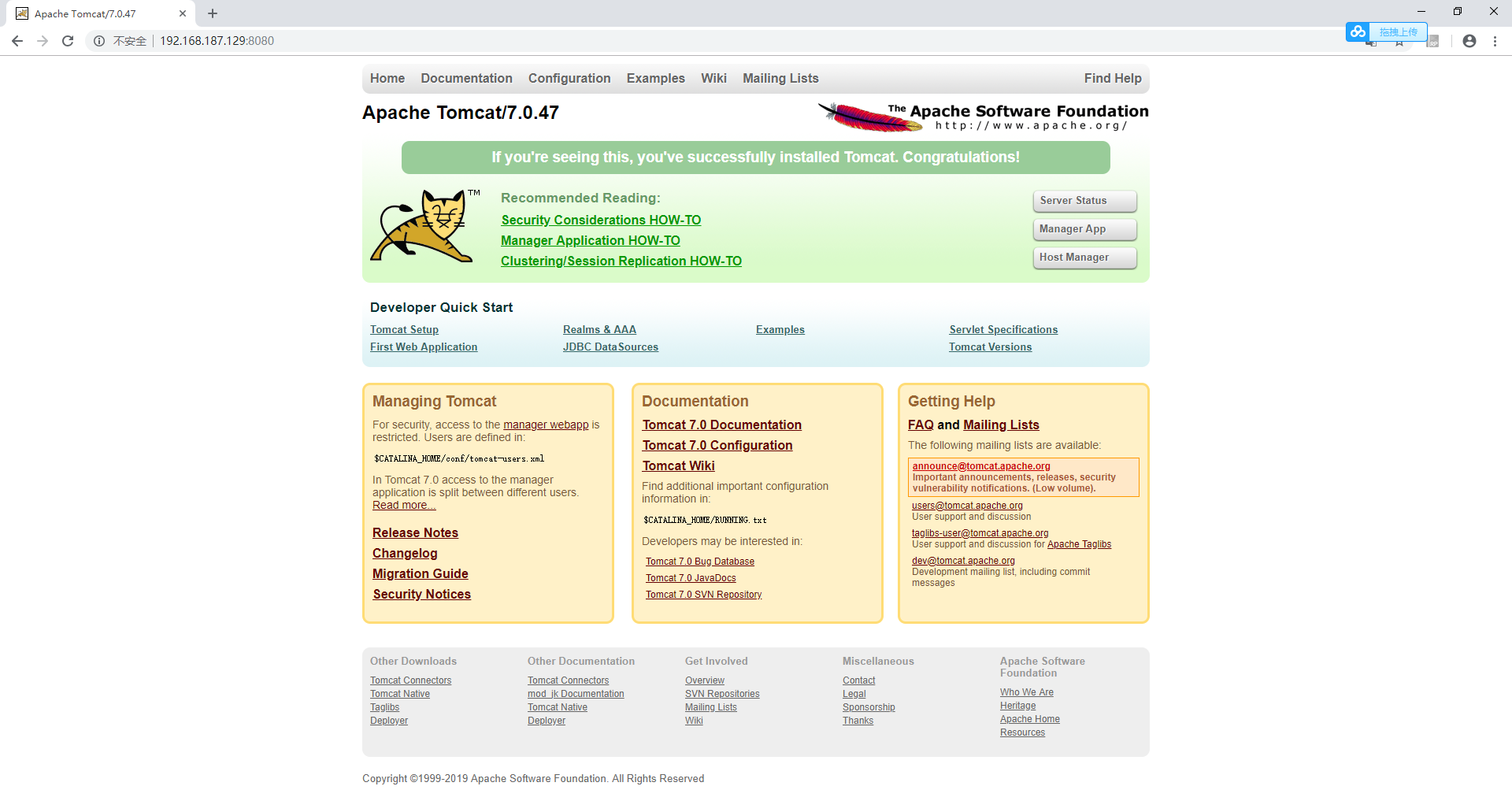
出现外网无法访问的情况，尝试关闭防火墙进行测试

systemctl status firewalld

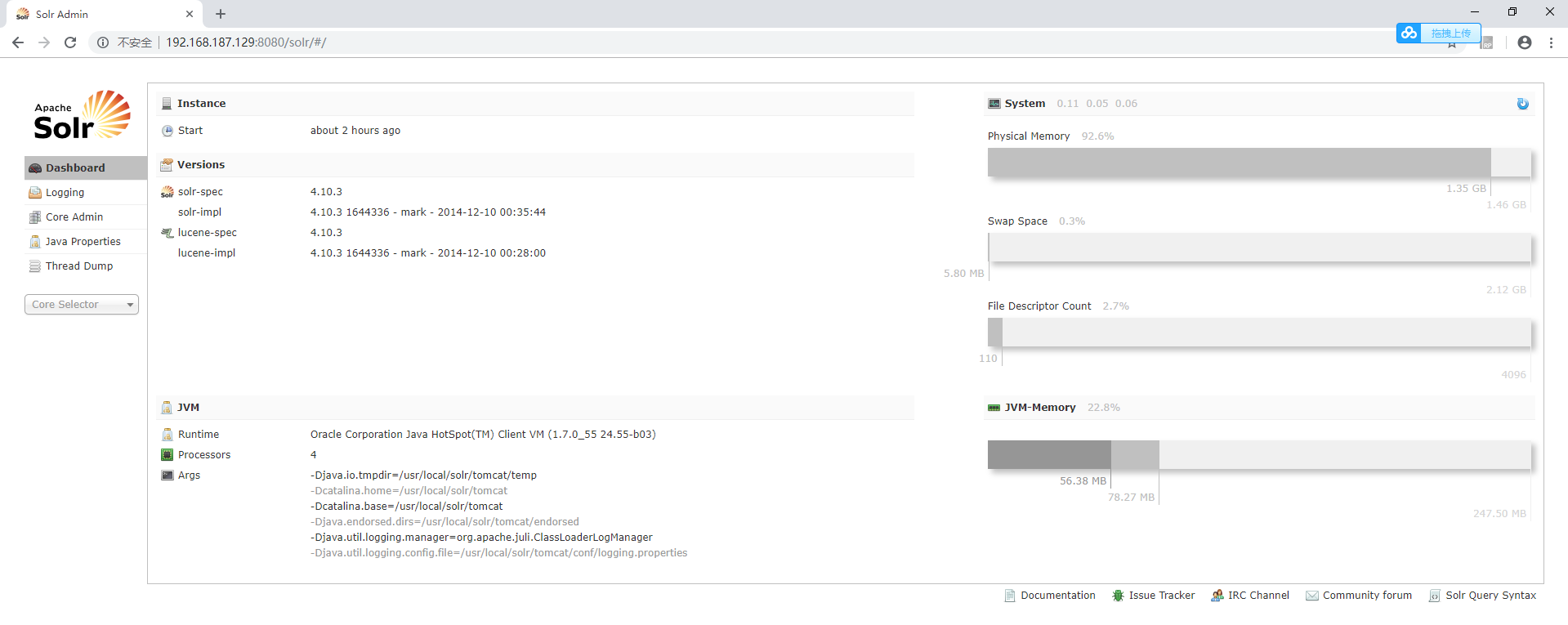
systemctl stop firewalld



<http://192.168.187.129:8080>



<http://192.168.187.129:8080/solr/#/>



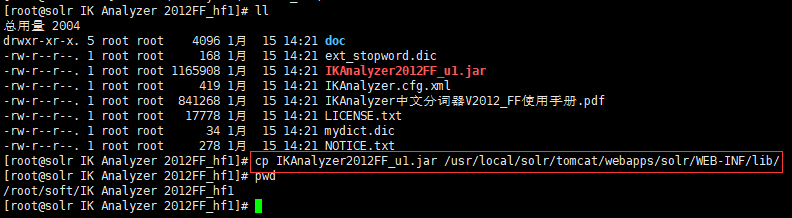
#### 配置中文分析器、自定义业务域

分析器使用IKAnalyzer。

使用方法：

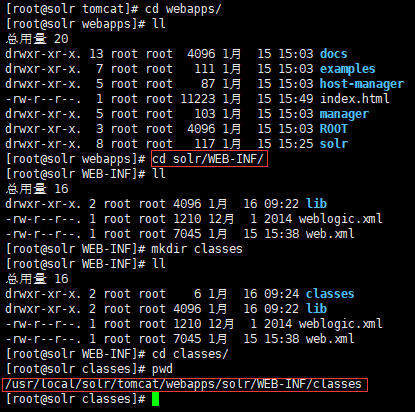
##### 第一步：把IKAnalyzer依赖的jar包添加到solr工程中。把分析器使用的扩展词典添加到classpath中

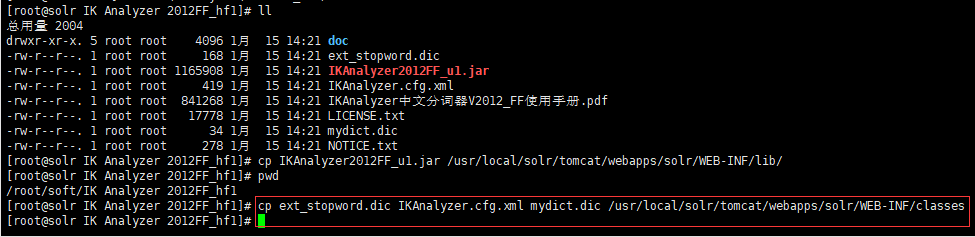
把IKAnalyzer依赖的jar包添加到solr工程中



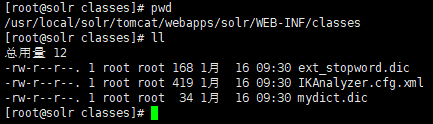
把分析器使用的扩展词典添加到classpath中，如果对应路径没有classes文件夹则需要手动创建，随后再将相关数据复制到该文件夹

（/usr/local/solr/tomcat/webapps/solr/WEB-INF/classes）



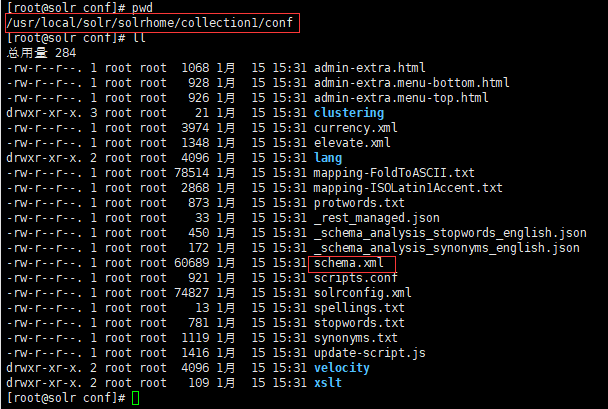


查看复制的数据信息

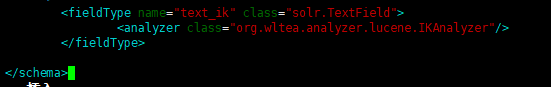


##### 第二步：需要自定义一个FieldType。Schema.xml中定义。可以在FieldType中指定中文分析器。

修改指定路径的文件，在末尾添加如下信息



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



##### 第三步：自定义域，指定域的类型为自定义的FieldType。

Sql语句：指定关键字查询之类的

SELECT

a.id,

a.title,

a.sell\_point,

a.price,

a.image,

b.name category\_name,

c.item\_desc

FROM

tb\_item a

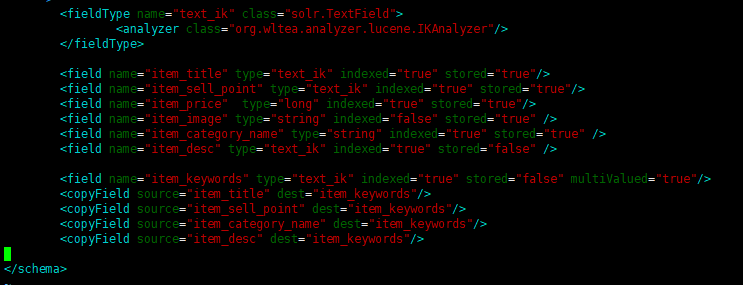
LEFT JOIN tb\_item\_cat b ON a.cid = b.id

LEFT JOIN tb\_item\_desc c ON a.id = c.item\_id

WHERE

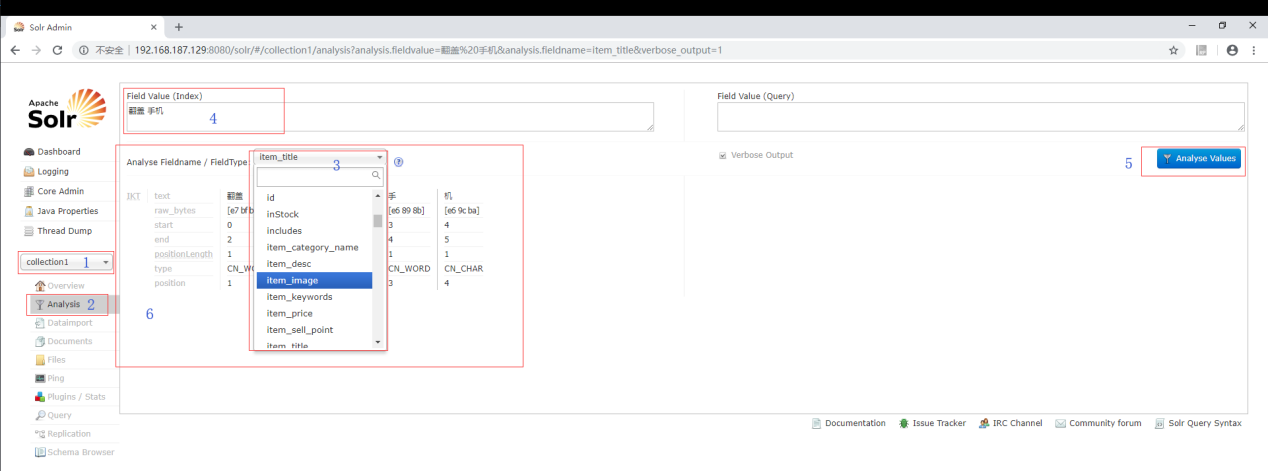
a.status = 1

|  |
| --- |
| 将下面的数据添加到schema.xml文件下方  <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进行测试

如果发现数据没有加载成功，需要确认配置文件schema.xml（如果是直接通过word复制的数据可能会出现和linux系统中空格字符格式不匹配的情况，此处需要注意），其次先关闭tomcat再次重启测试



### 索引库中导入数据

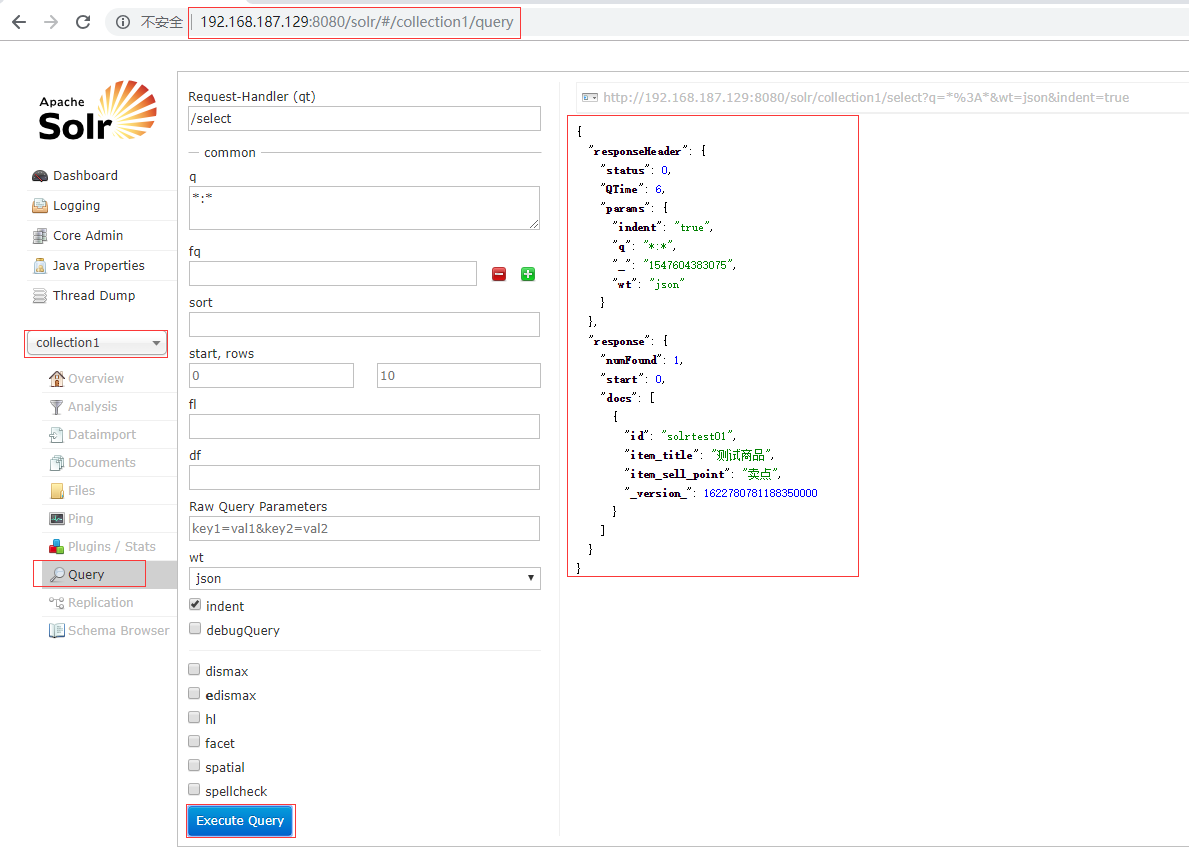
#### Solrj的使用

|  |
| --- |
| **public** **class** SolrJTest {  @Test  **public** **void** testSolrJ() **throws** Exception {  //创建连接  SolrServer solrServer = **new** HttpSolrServer("http://192.168.25.154:8080/solr");  //创建一个文档对象  SolrInputDocument document = **new** SolrInputDocument();  //添加域  document.addField("id", "solrtest01");  document.addField("item\_title", "测试商品");  document.addField("item\_sell\_point", "卖点");  //添加到索引库  solrServer.add(document);  //提交  solrServer.commit();  }    @Test  **public** **void** testQuery() **throws** Exception {  //创建连接  SolrServer solrServer = **new** HttpSolrServer("http://192.168.25.154:8080/solr");  //创建一个查询对象  SolrQuery query = **new** SolrQuery();  query.setQuery("\*:\*");  //执行查询  QueryResponse response = solrServer.query(query);  //取查询结果  SolrDocumentList solrDocumentList = response.getResults();  **for** (SolrDocument solrDocument : solrDocumentList) {  System.***out***.println(solrDocument.get("id"));  System.***out***.println(solrDocument.get("item\_title"));  System.***out***.println(solrDocument.get("item\_sell\_point"));  }  }  } |

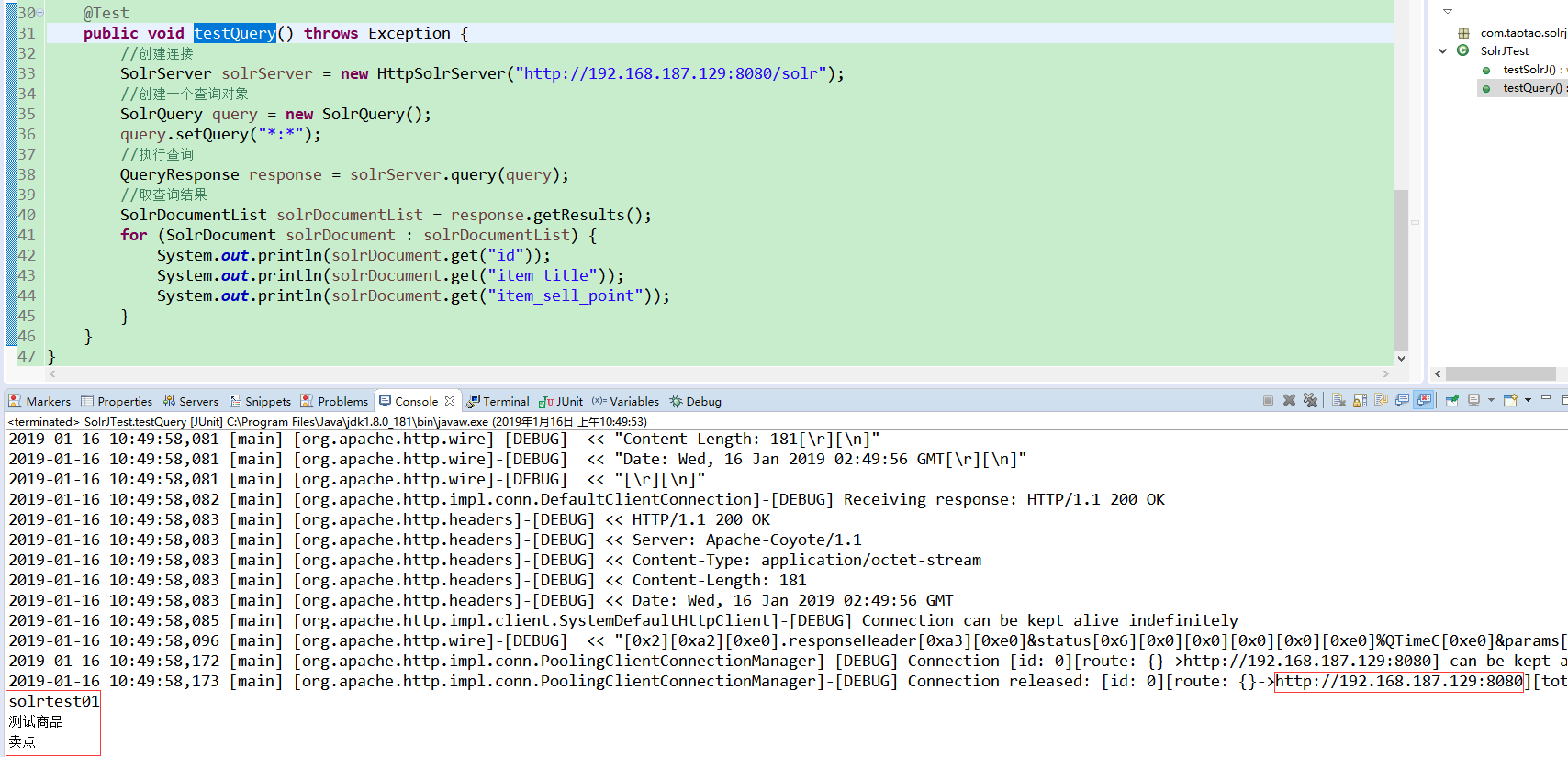
测试分析：

执行testSolrJ方法，随后在网页中访问solr主工程测试是否成功

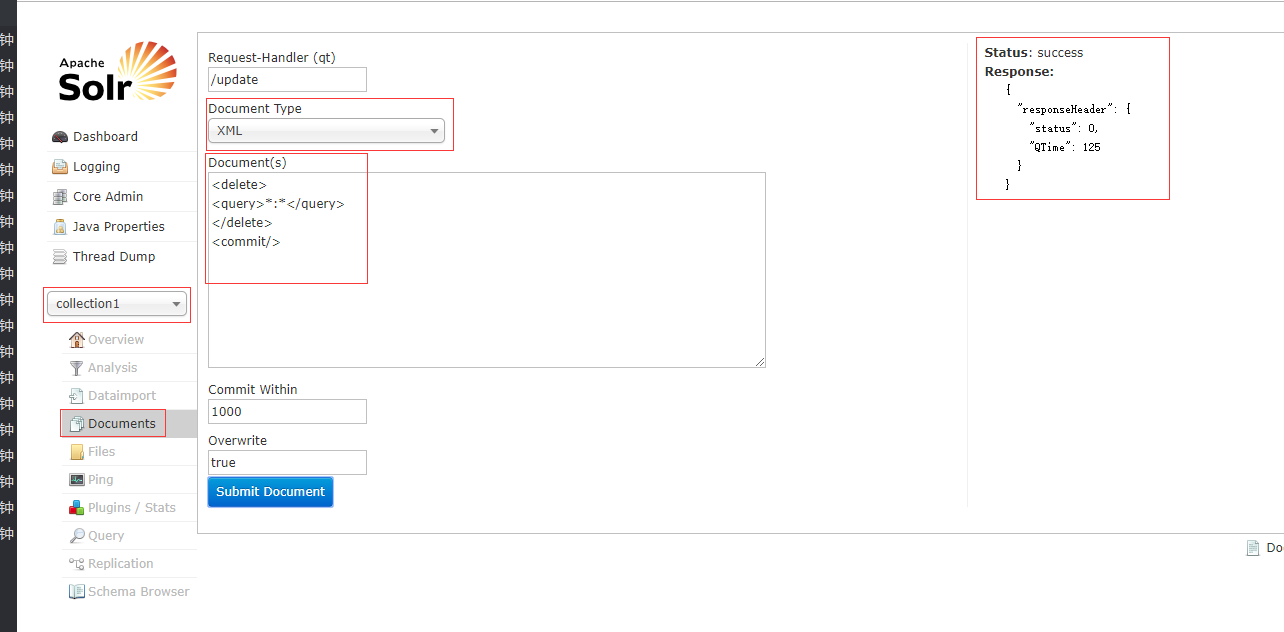




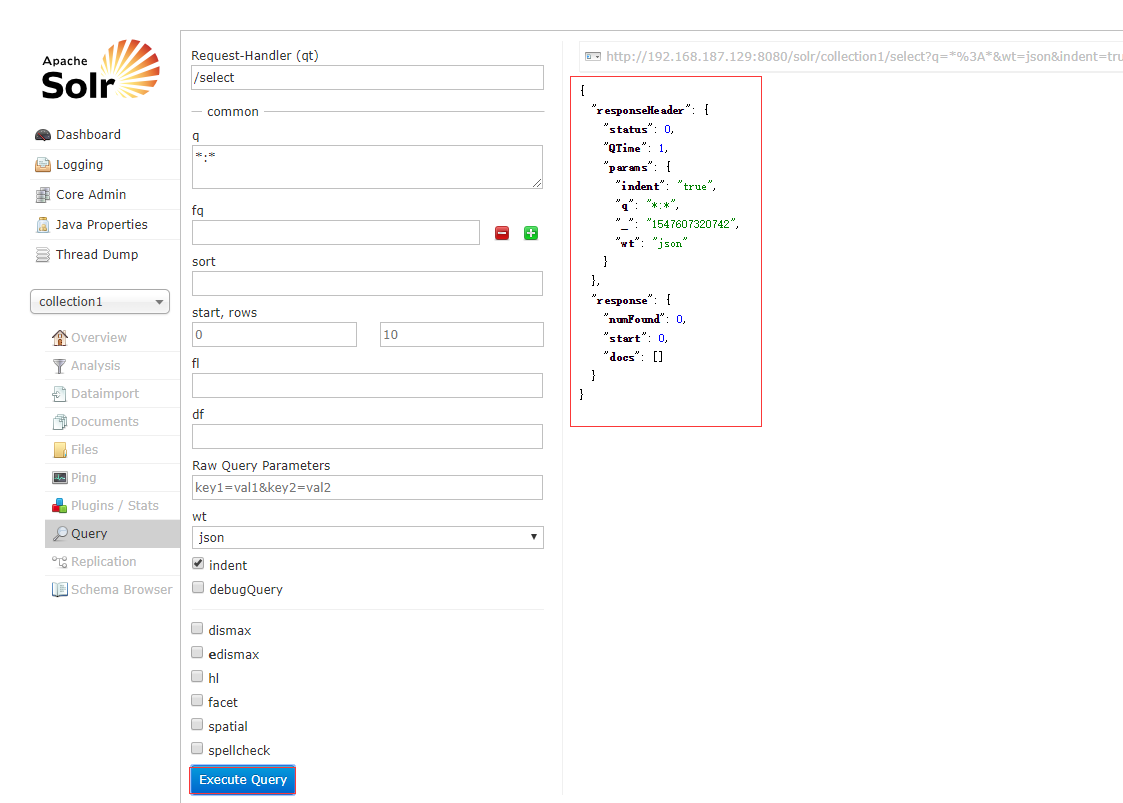
执行testQuery方法，显示查找的数据信息



使用后台管理系统删除数据库：



再次点击查找，数据被清空



#### 导入数据

##### 分析

从数据库中根据sql语句查询数据，遍历数据创建文档对象，把文档对象写入索引库。

##### Dao层

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

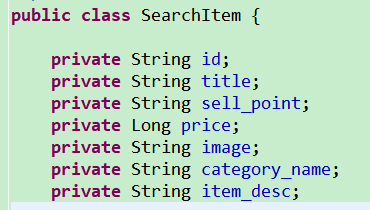
WHERE

a.`status` = 1

需要创建一个mapper文件。

在taotao-search中创建相关的pojo、mapper、service、controller包

创建SearchItem接收查找结果：

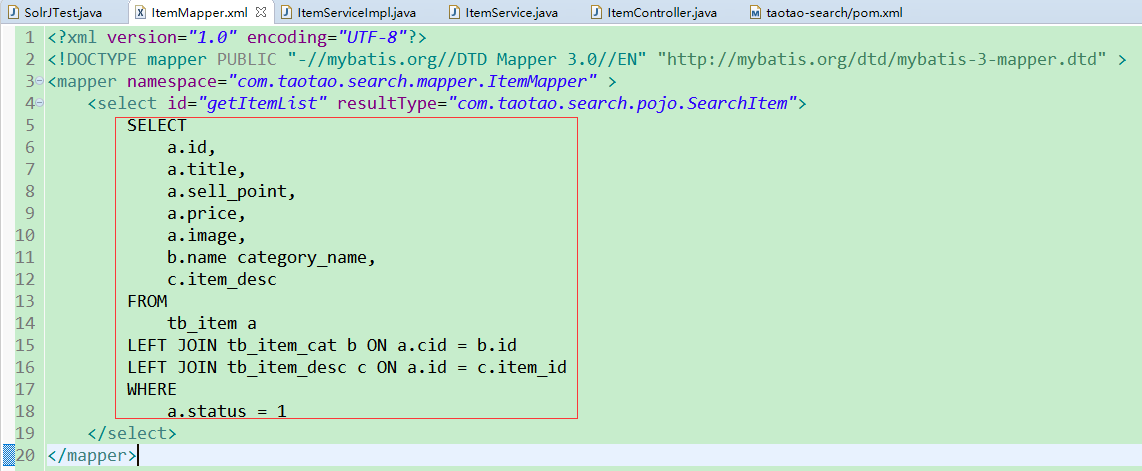


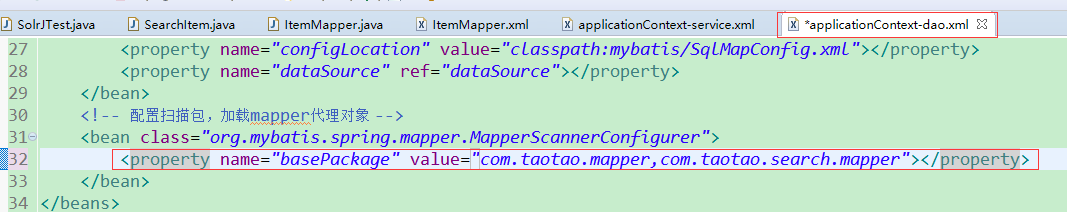
Dao层接口：

|  |
| --- |
| **package** com.taotao.search.mapper;  **import** java.util.List;  **import** com.taotao.search.pojo.SearchItem;  **public** **interface** ItemMapper {  List<SearchItem> getItemList();  } |

Mapper文件：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >  <mapper namespace=*"com.taotao.search.mapper.ItemMapper"* >  <select id=*"getItemList"* resultType=*"com.taotao.search.pojo.SearchItem"*>  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  </select>  </mapper> |





##### Service层

取商品列表，遍历列表，创建文档对象，把文档对象写入索引库。

要操作索引库需要SolrServer对象，可以把SolrServer放到spring容器中，注入到Service。



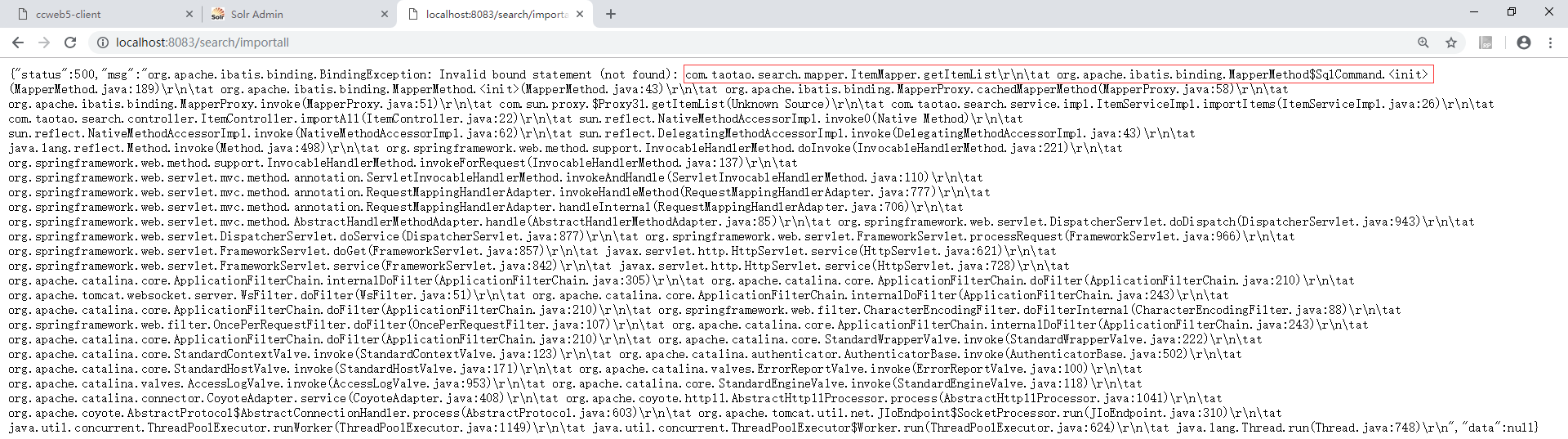
|  |
| --- |
| **public** **interface** ItemService {  **public** TaotaoResult importItems() **throws** Exception;  }  ServiceImpl：  **package** com.taotao.search.service.impl;  **import** java.util.List;  **import** org.apache.solr.client.solrj.SolrServer;  **import** org.apache.solr.common.SolrInputDocument;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.search.mapper.ItemMapper;  **import** com.taotao.search.pojo.SearchItem;  **import** com.taotao.search.service.ItemService;  @Service  **public** **class** ItemServiceImpl **implements** ItemService {  @Autowired  **private** SolrServer solrServer;  @Autowired  **private** ItemMapper itemMapper;  @Override  **public** TaotaoResult importItems() **throws** Exception {  //查询数据库获得商品列表  List<SearchItem> itemList = itemMapper.getItemList();  //遍历列表  **for** (SearchItem item : itemList) {  //创建文档对象  SolrInputDocument document = **new** SolrInputDocument();  //添加域  document.addField("id", item.getId());  document.addField("item\_title", item.getTitle());  document.addField("item\_sell\_point", item.getSell\_point());  document.addField("item\_price", item.getPrice());  document.addField("item\_image", item.getImage());  document.addField("item\_category\_name", item.getCategory\_name());  document.addField("item\_desc", item.getItem\_desc());  //写入索引库  solrServer.add(document);  }  //提交  solrServer.commit();  **return** TaotaoResult.*ok*();  }  } |

##### Controller层

请求一个url，返回TaotaoResult。

|  |
| --- |
| **package** com.taotao.search.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.common.utils.ExceptionUtil;  **import** com.taotao.search.service.ItemService;  @Controller  **public** **class** ItemController {  @Autowired  **private** ItemService itemService;    @RequestMapping("/importall")  @ResponseBody  **public** TaotaoResult importAll() {  **try** {  TaotaoResult result = itemService.importItems();  **return** result;  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  }  } |

可能出现的问题：



原因分析：相关的资源文件没有找到，导致数据绑定失败，需要在pom.xml中添加相关的配置。Pom.xml文件修改如下：

<resources>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

</includes>

<filtering>false</filtering>

</resource>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*.properties</include>

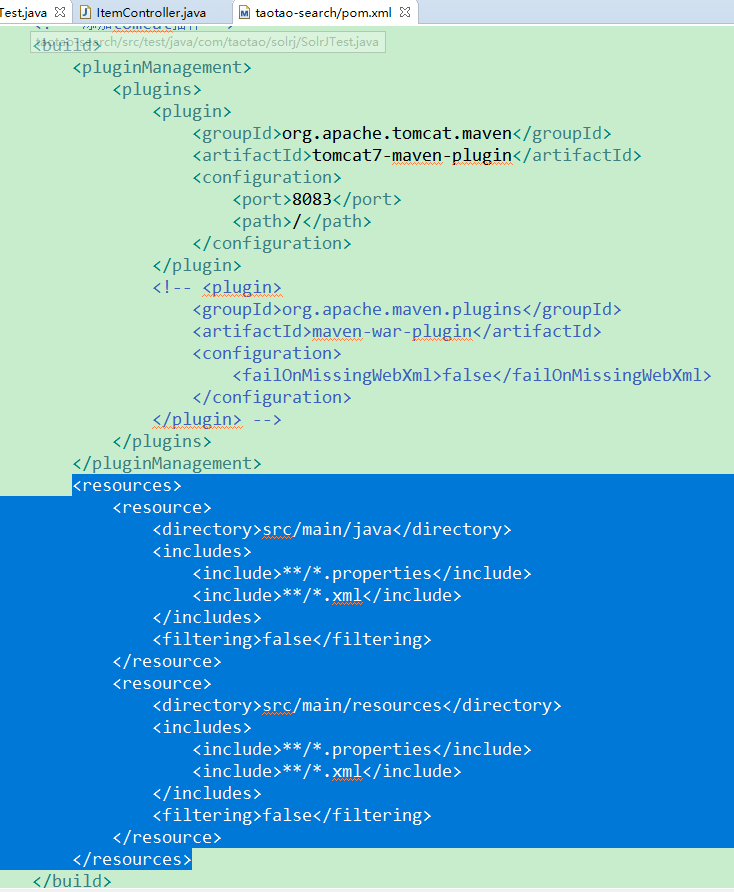
<include>\*\*/\*.xml</include>

</includes>

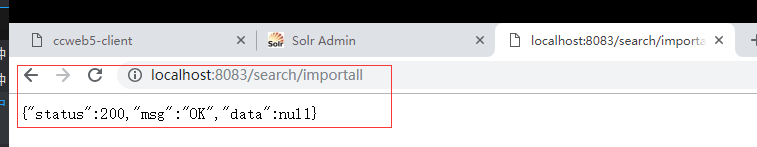
<filtering>false</filtering>

</resource>

</resources>



测试，访问数据：<http://localhost:8083/search/importall>



在solr后台查看数据：

