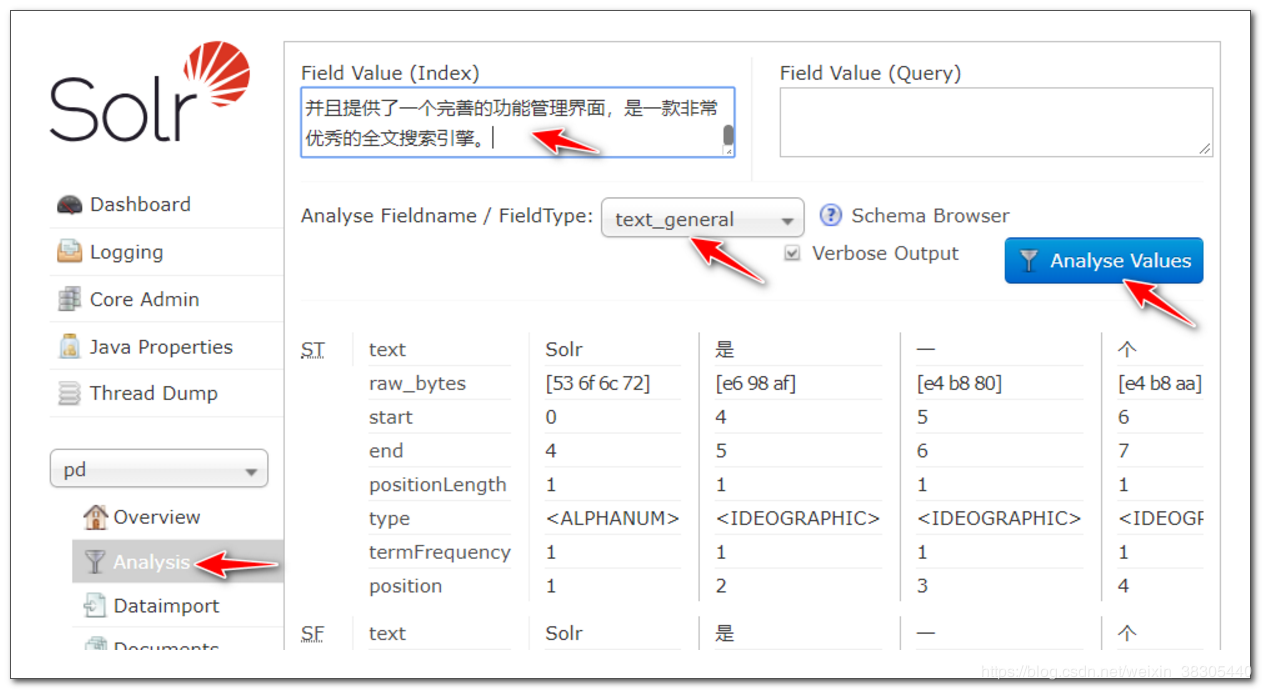
# **中文分词测试**



填入以下文本, 观察分词结果:

Solr是一个高性能，采用Java5开发，基于Lucene的全文搜索服务器。同时对其进行了扩展，提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文搜索引擎。

## **中文分词工具 - ik-analyzer**

https://github.com/magese/ik-analyzer-solr



下载 ik-analyzer 分词 jar 文件,传到 solr目录/server/solr-webapp/webapp/WEB-INF/lib

为了后续操作方便,我们把后面用到的jar文件一同传到服务器,包括四个文件:

ik-analyzer-8.1.0.jar

mysql-connector-java-5.1.46.jar

solr-dataimporthandler-8.1.1.jar

solr-dataimporthandler-extras-8.1.1.jar

复制6个文件到 solr目录/server/solr-webapp/webapp/WEB-INF/classes

# classes目录如果不存在,需要创建该目录

mkdir /usr/local/solr-8.1.1/server/solr-webapp/webapp/WEB-INF/classes

这6个文件复制到 classes 目录下

resources/

IKAnalyzer.cfg.xml

ext.dic

stopword.dic

stopwords.txt

ik.conf

dynamicdic.txt

* 配置 managed-schema

修改 solr目录/server/solr/pd/conf/managed-schema,添加 ik-analyzer 分词器

<!-- ik分词器 -->

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

<analyzer type="index">

<tokenizer class="org.wltea.analyzer.lucene.IKTokenizerFactory" useSmart="false" conf="ik.conf"/>

<filter class="solr.LowerCaseFilterFactory"/>

</analyzer>

<analyzer type="query">

<tokenizer class="org.wltea.analyzer.lucene.IKTokenizerFactory" useSmart="true" conf="ik.conf"/>

<filter class="solr.LowerCaseFilterFactory"/>

</analyzer>

</fieldType>

重启 solr 服务

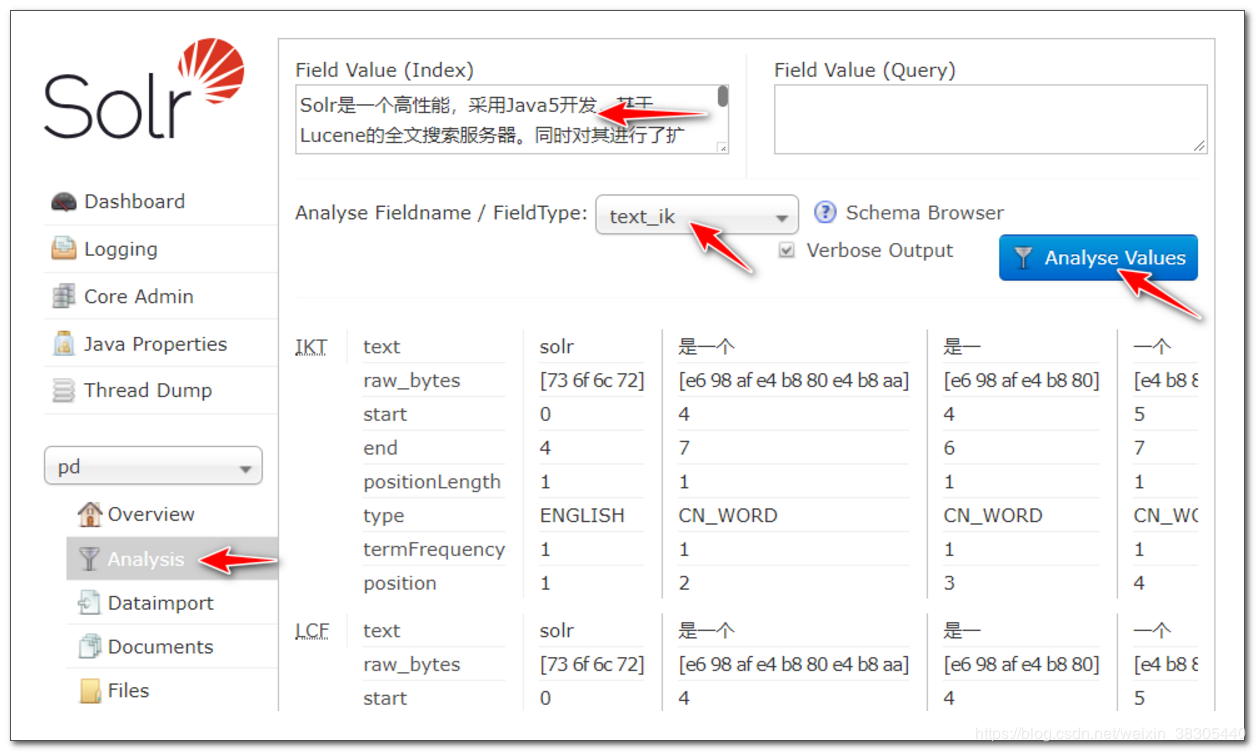
cd /usr/local/solr-8.1.1

bin/solr restart -force

## **使用 ik-analyzer 对中文进行分词测试**

填入以下文本, 选择使用 text\_ik 分词器, 观察分词结果:

Solr是一个高性能，采用Java5开发，基于Lucene的全文搜索服务器。同时对其进行了扩展，提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文搜索引擎。



## **设置停止词**

上传停止词配置文件到 solr目录/server/solr-webapp/webapp/WEB-INF/classes

stopword.dic

stopwords.txt

重启服务,观察分词结果中,停止词被忽略

bin/solr restart -force

# **准备 mysql 数据库数据**

* 用 sqlyog 执行 pd.sql
* 授予 root 用户 跨网络访问权限  
  注意: 此处设置的是远程登录的 root 用户,本机登录的 root 用户密码不变

grant all on \*.\* to 'root'@'%' identified by 'root';

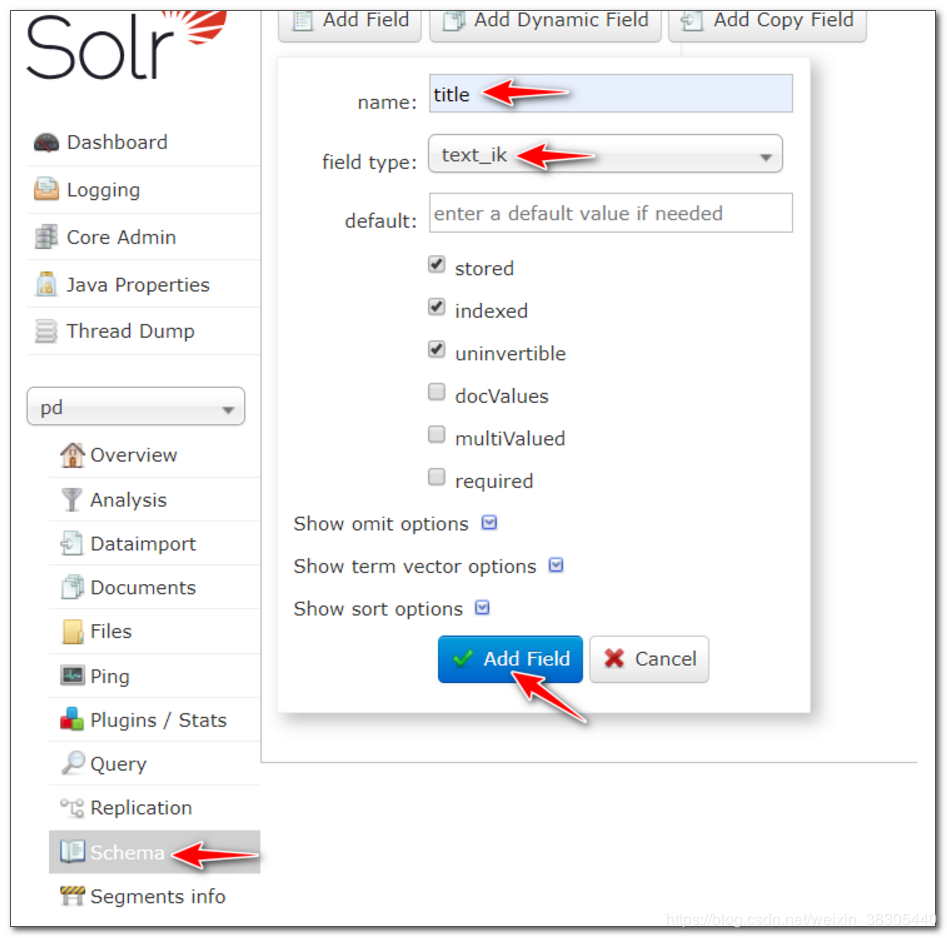
随机修改30%的商品,让商品下架,以便后面做查询测试

UPDATE pd\_item SET STATUS=0 WHERE RAND()<0.3

# **从 mysql 导入商品数据**

## **设置字段**

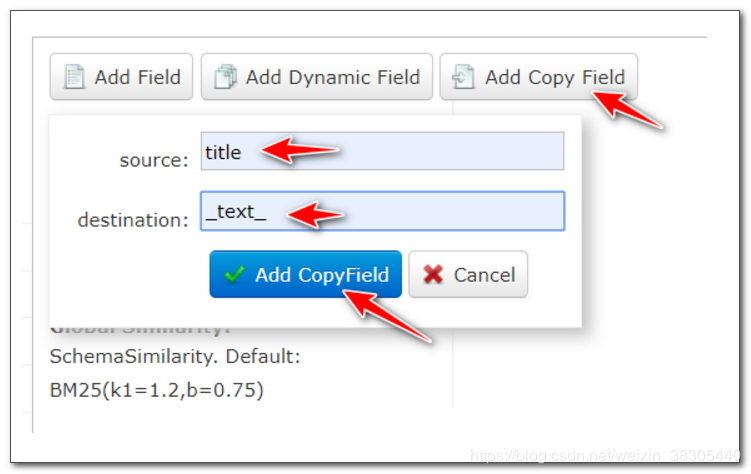
* title text\_ik
* sellPoint text\_ik
* price plong
* barcode string
* image string
* cid plong
* status pint
* created pdate
* updated pdate



## **Copy Field 副本字段**

查询时需要按字段查询,例如 title:电脑, 可以将多个字段的值合并到一个字段进行查询,默认查询字段 \_text\_

将 title 和 sellPoint 复制到 \_text\_ 字段



## **Data Import Handler 配置**

* 添加 jar 文件

solr-dataimporthandler-8.1.1.jar

solr-dataimporthandler-extras-8.1.1.jar

Data Import Handler 的 jar 文件存放在 solr目录/dist 目录下

复制这两个文件和 mysql 的 jar 文件到 solr目录/server/solr-webapp/webapp/WEB-INF/lib

dih-config.xml

修改 mysql 的 ip 地址,传到

solr目录/server/solr/pd/conf

solrconfig.xml 中添加 DIH 配置

<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">

<lst name="defaults">

<str name="config">dih-config.xml</str>

</lst>

</requestHandler>

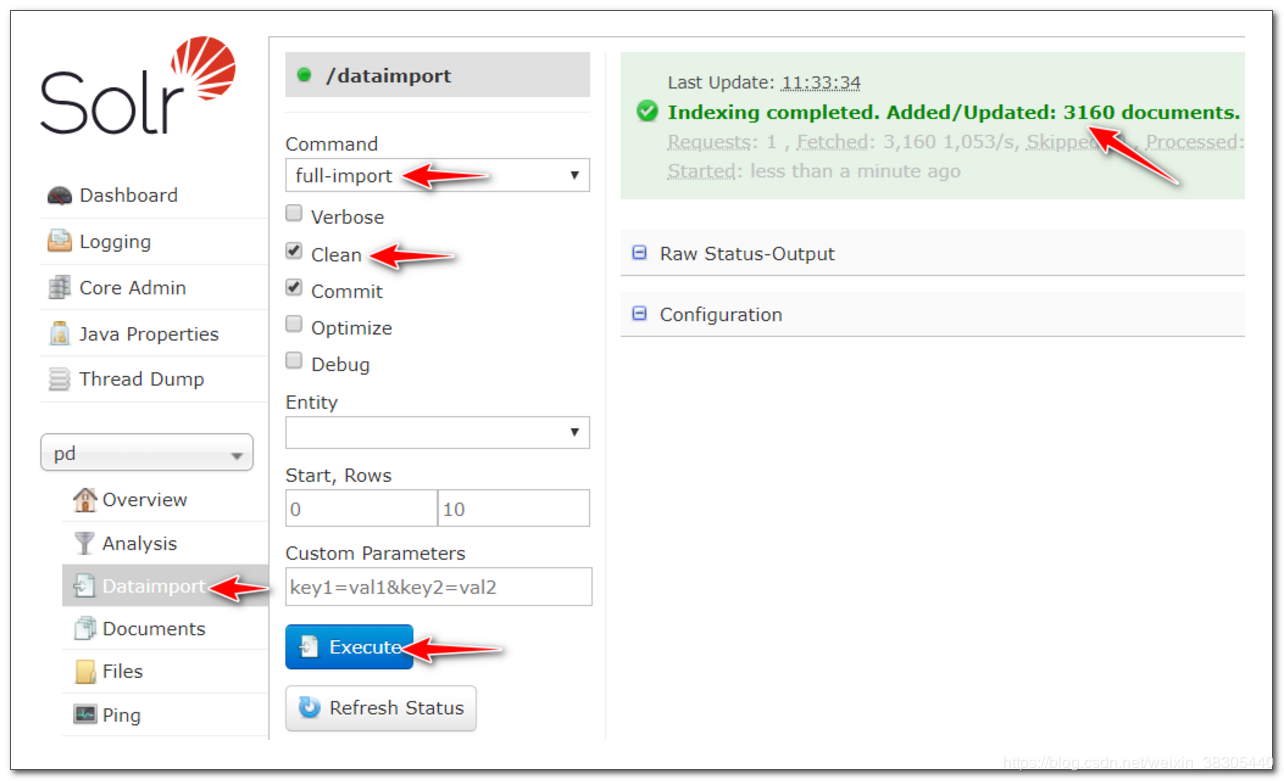
重启 solr

cd /usr/local/solr-8.1.1

bin/solr restart -force

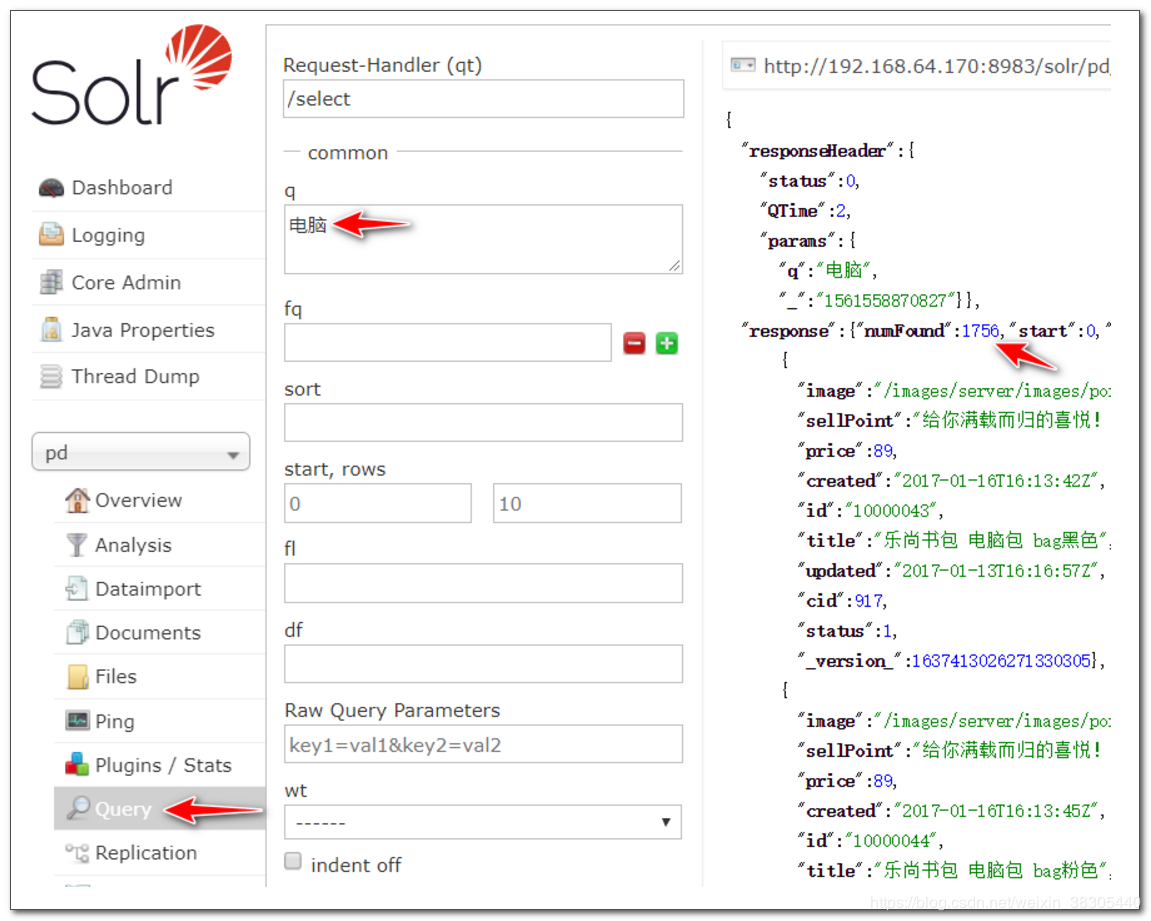
## **导入数据**

重启 solr 后导入数据,确认导入的文档数量为 3160

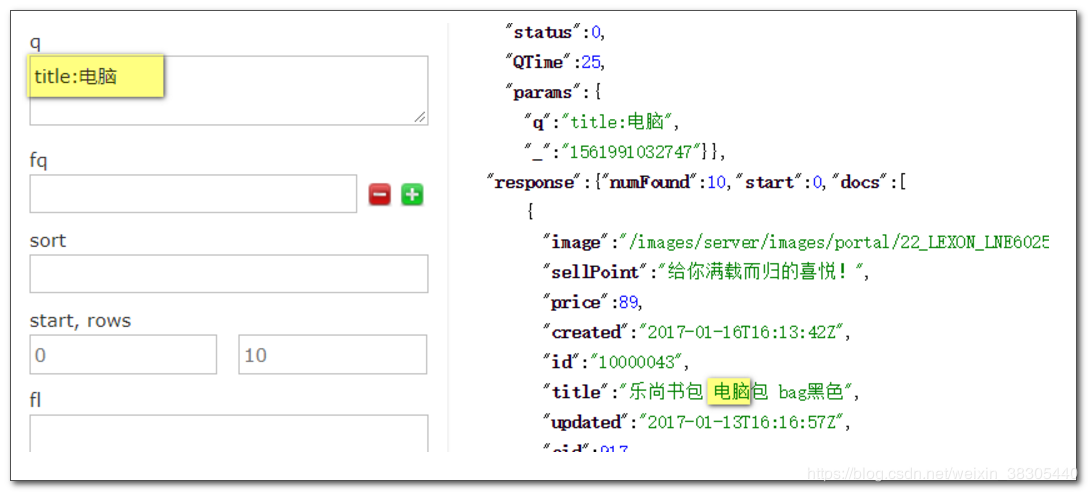


## **查询测试**

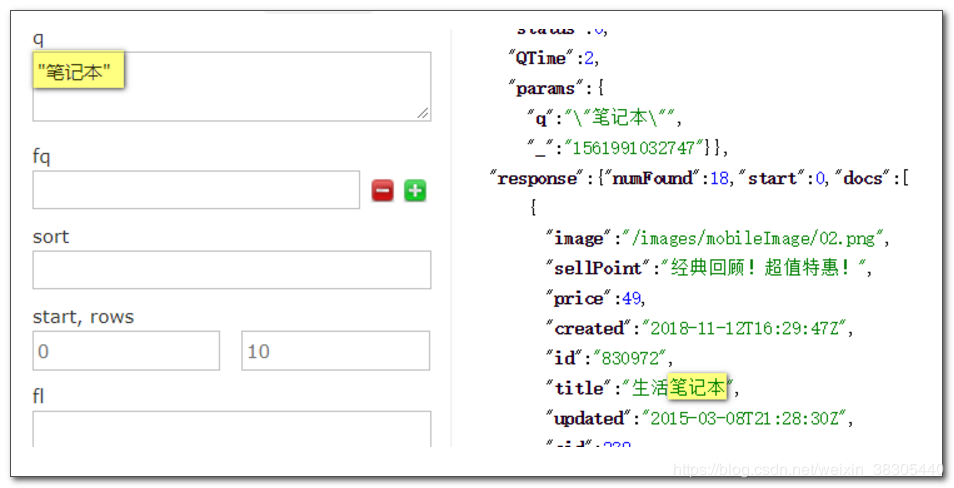
### **在复制字段 \_text\_ 中查找 电脑**



### **在标题中查找 电脑**



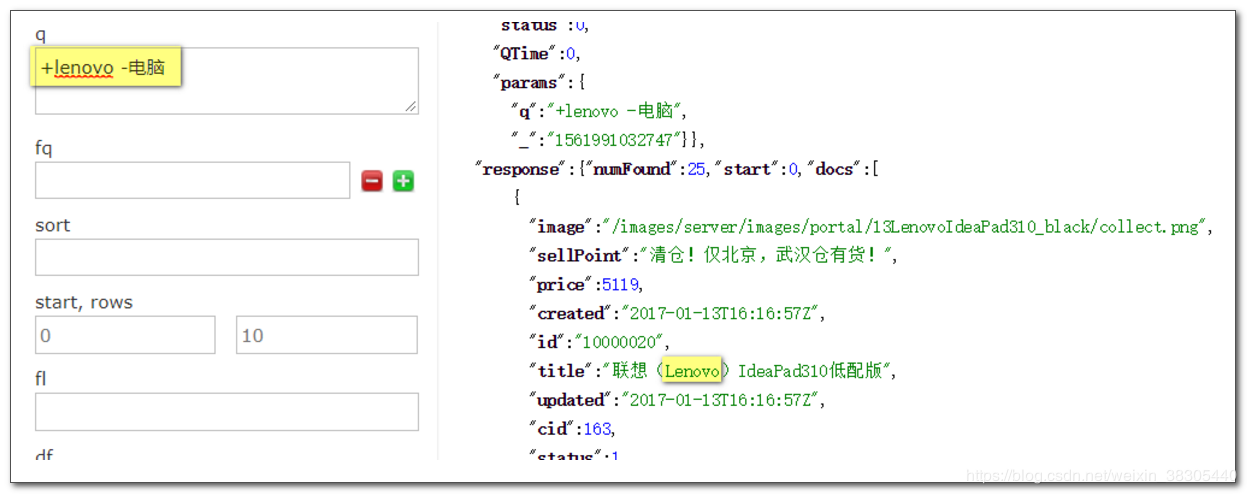
### **用双引号查找完整词 "笔记本"**



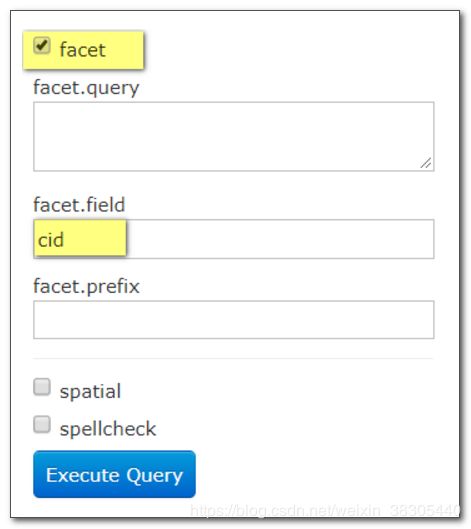
### **搜索 +lenovo +电脑**

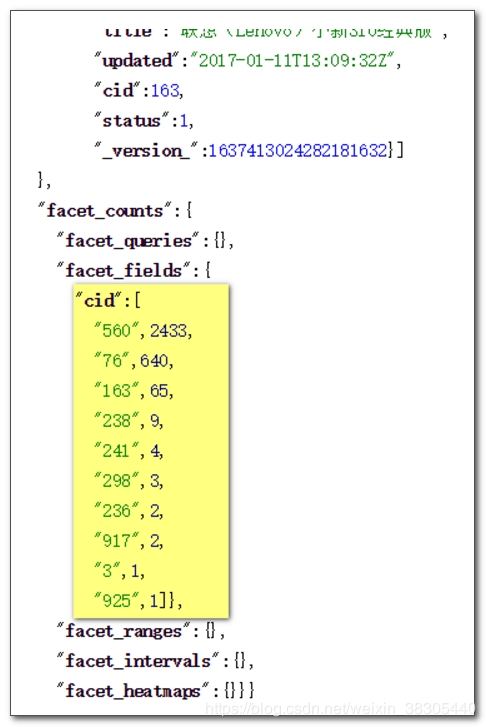


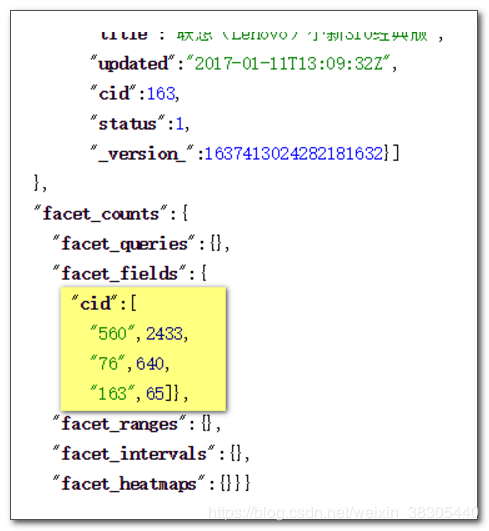
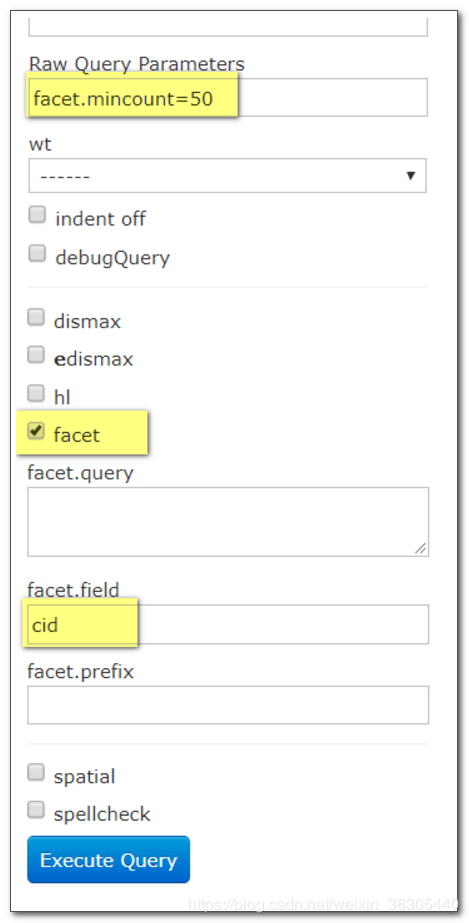
### **搜索 +lenovo -电脑**



### **统计 cid**

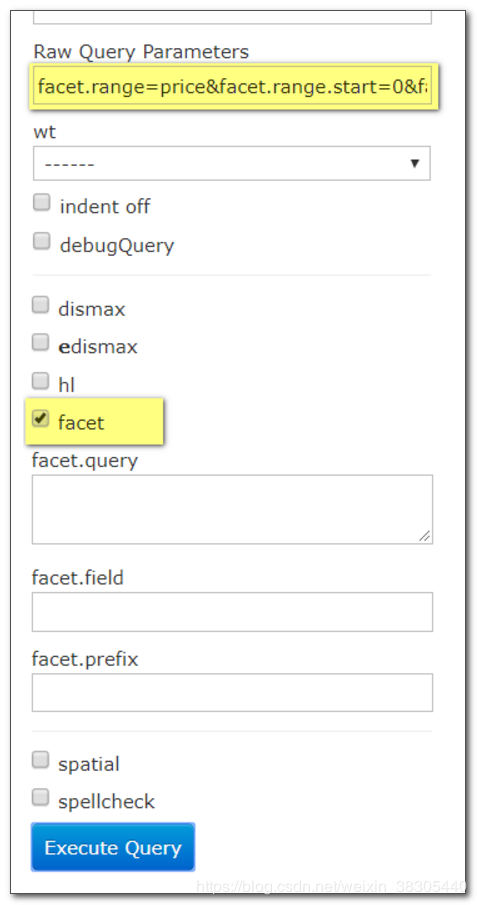


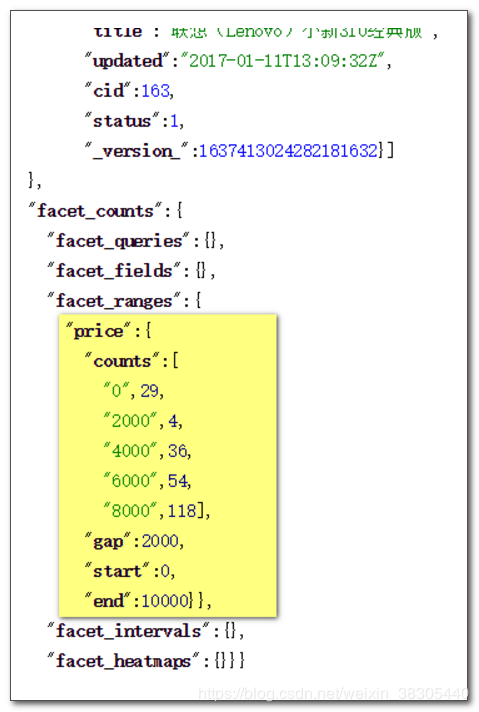




### **价格范围**

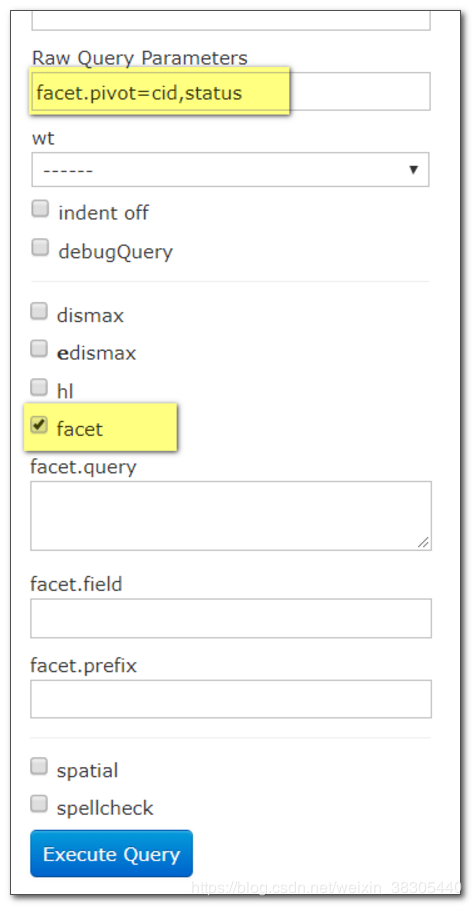
在 Raw Query Parameters 输入框中填入以下内容:  
  
facet.range=price&facet.range.start=0&facet.range.end=10000&facet.range.gap=2000

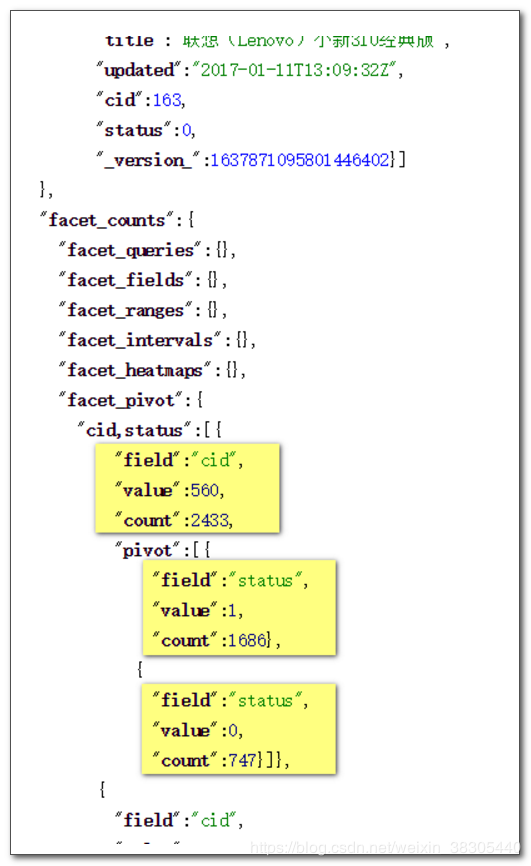




### **多字段统计**

在 Raw Query Parameters 输入框中填入以下内容:  
  
facet.pivot=cid,status



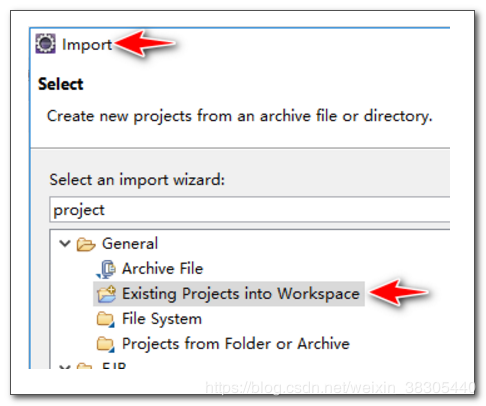


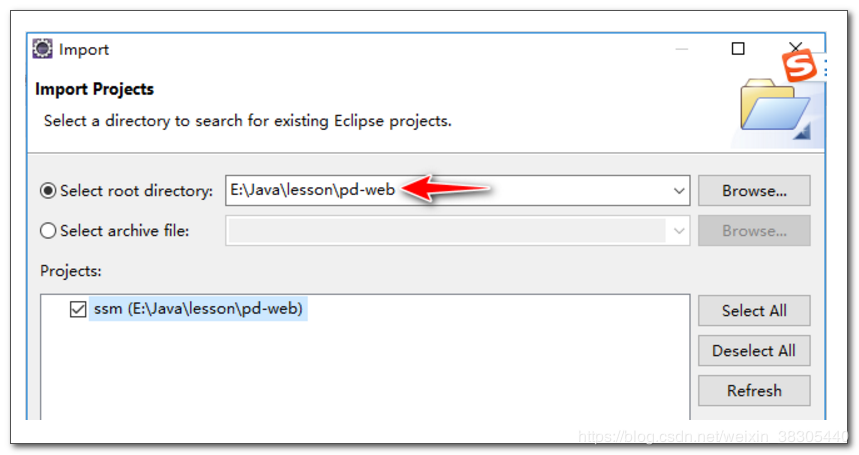
# **拼多商城实现商品的全文检索**

## **修改 hosts 文件, 添加 www.pd.com 映射**

127.0.0.1 www.pd.com

## **eclipse 导入 pd-web 项目**





## **修改数据库连接配置**

application.yml 配置文件中,修改连接配置

spring:

datasource:

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/pd\_store?useUnicode=true&characterEncoding=UTF-8

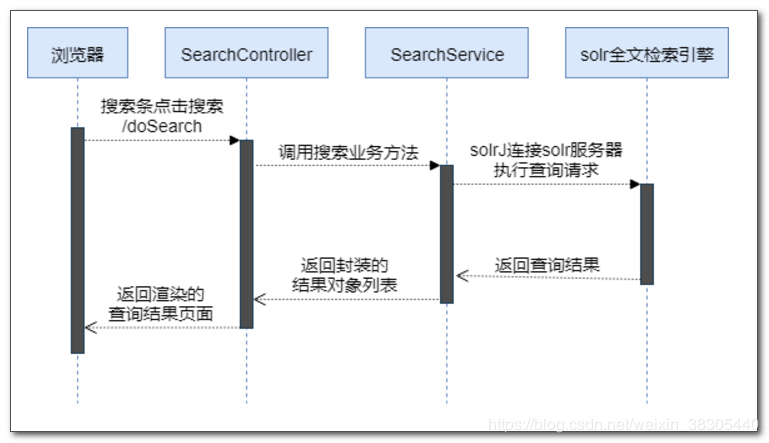
username: root

password: root

## **启动项目, 访问 www.pd.com**



## **商品检索调用分析**



## **pom.xml 添加 solr 和 lombok 依赖**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-solr</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

## **application.yml 添加 solr 连接信息**

spring:

data:

solr: #注意修改ip地址

host: http://192.168.64.170:8983/solr/pd

## **Item 实体类**

package com.pd.pojo;

import java.io.Serializable;

import org.apache.solr.client.solrj.beans.Field;

import lombok.Data;

@Data

public class Item implements Serializable {

private static final long serialVersionUID = 1L;

@Field("id")

private String id;

@Field("title")

private String title;

@Field("sellPoint")

private String sellPoint;

@Field("price")

private Long price;

@Field("image")

private String image;

}

## **SearchService 业务接口**

package com.pd.service;

import java.util.List;

import com.pd.pojo.Item;

public interface SearchService {

List<Item> findItemByKey(String key) throws Exception;

}

## **SearchServiceImpl 业务实现类**

package com.pd.service.impl;

import java.util.List;

import org.apache.solr.client.solrj.SolrClient;

import org.apache.solr.client.solrj.SolrQuery;

import org.apache.solr.client.solrj.response.QueryResponse;

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

import org.springframework.context.annotation.Configuration;

import org.springframework.stereotype.Service;

import com.pd.pojo.Item;

import com.pd.service.SearchService;

@Service

public class SearchServiceImpl implements SearchService {

/\*

\* SolrClient实例是在 SolrAutoConfiguration 类中创建的

\*

\* SolrAutoConfiguration添加了@Configuration注解,

\* 是spring boot自动配置类,其中的solrClient()方法中创建了SolrClient实例

\*/

@Autowired

private SolrClient solrClient;

@Override

public List<Item> findItemByKey(String key) throws Exception {

//封装查询的关键词

//也可以封装其他的查询参数,比如指定字段,facet设置等

SolrQuery query = new SolrQuery(key);

//查询前多少条数据

query.setStart(0);

query.setRows(20);

//执行查询并得到查询结果

QueryResponse qr = solrClient.query(query);

//把查询结果转成一组商品实例

List<Item> beans = qr.getBeans(Item.class);

return beans;

}

}

## **SearchController 控制器**

package com.pd.controller;

import java.util.List;

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import com.pd.pojo.Item;

import com.pd.service.SearchService;

@Controller

public class SearchController {

@Autowired

private SearchService searchService;

@GetMapping("/search/toSearch.html")

public String search(String key, Model model) throws Exception {

List<Item> itemList = searchService.findItemByKey(key);

model.addAttribute("list", itemList);

return "/search.jsp";

}

}