品优购电商系统开发

第 9 章

搜索解决方案-Solr 【1】

传智播客.黑马程序员

课程目标

目标 1：完成 solr 环境安装、中文分析器和业务域的配置

目标 2：会使用 Spring Data Solr 完成增删改查操作

目标 3：完成批量数据导入功能

目标 4：完成按关键字搜索功能

1.Solr 安装与配置

1.1 什么是 Solr

大多数搜索引擎应用都必须具有某种搜索功能，问题是搜索功能往往是巨大的资源消耗并且它们由于沉重的数据库加载而拖垮你的应用的性能。

这就是为什么转移负载到一个外部的搜索服务器是一个不错的主意，Apache Solr 是一个流行的开源搜索服务器，它通过使用类似 REST 的 HTTP API，这就确保你能从几乎任何编程语言来使用 solr。

Solr 是一个开源搜索平台，用于构建搜索应用程序。 它建立在 Lucene(全文搜索引擎)之上。 Solr 是企业级的，快速的和高度可扩展的。 使用 Solr 构建的应用程序非常复杂，可提供高性能。

为了在 CNET 网络的公司网站上添加搜索功能，Yonik Seely 于 2004 年创建了 Solr。并在 2006 年 1 月，它成为 Apache 软件基金会下的一个开源项目。并于 2016 年发布最新版本 Solr 6.0，支持并行 SQL 查询的执行。

Solr 可以和 Hadoop 一起使用。由于 Hadoop 处理大量数据，Solr 帮助我们从这么大的源中找到所需的信息。不仅限于搜索，Solr 也可以用于存储目的。像其他 NoSQL 数据库一样，它是一种非关系数据存储和处理技术。

总之，Solr 是一个可扩展的，可部署，搜索/存储引擎，优化搜索大量以文本为中心的数据。

1.2 Solr 安装

1：安装 Tomcat，解压缩即可。

2：解压 solr。

3：把 solr 下的 dist 目录 solr-4.10.3.war 部署到 Tomcat\webapps 下(去掉版本号)。

4：启动 Tomcat 解压缩 war 包

5:把solr下example/lib/ext 目录下的所有的 jar 包,添加到 solr 的工程中(\WEB-INF\lib 目录下)。

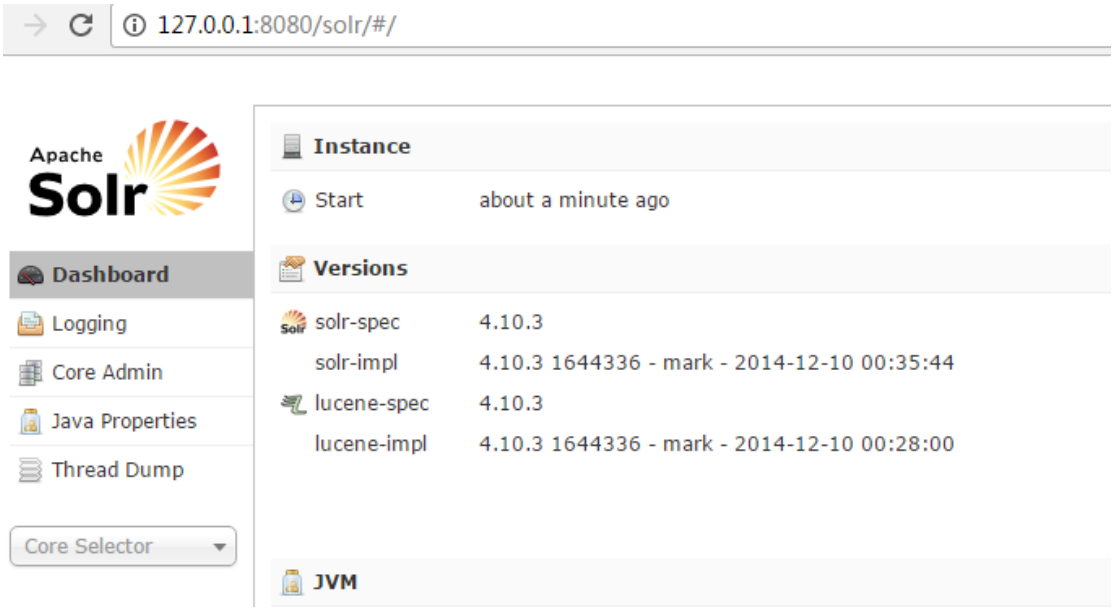
6：创建一个 solrhome 。solr 下的/example/solr 目录就是一个 solrhome。复制此目录到 D 盘改名为 solrhome

7：关联 solr 及 solrhome。需要修改 solr 工程的 web.xml 文件。

```
<env-entry>
    <env-entry-name>solr/home</env-entry-name>
    <env-entry-value>d:\solrhome</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

8：启动 Tomcat

http://IP:8080/solr/



1.3 中文分析器 IK Analyzer

1.3.1 IK Analyzer 简介

IK Analyzer 是一个开源的，基于 java 语言开发的轻量级的中文分词工具包。从 2006 年 12 月推出 1.0 版开始， IKAnalyzer 已经推出了 4 个大版本。最初，它是以开源项目 Luence 为应用主体的，结合词典分词和文法分析算法的中文分词组件。从 3.0 版本开始，IK 发展为面向 Java 的公用分词组件，独立于 Lucene 项目，同时提供了对 Lucene 的默认优化实现。在 2012 版本中，IK 实现了简单的分词歧义排除算法，标志着 IK 分词器从单纯的词典分词向模拟语义分词衍化。

1.3.2 IK Analyzer 配置

步骤：

1、把 IKAnalyzer2012FF_u1.jar 添加到 solr 工程的 lib 目录下

2、创建 WEB-INF/classes 文件夹 把扩展词典、停用词词典、配置文件放到 solr 工程的 WEB-INF/classes 目录下。

3、修改 Solrhome 的 schema.xml 文件，配置一个 FieldType，使用 IKAnalyzer

```
<fieldType name="text_ik" class="solr.TextField">
    <analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>
</fieldType>
```

1.4 配置域

域相当于数据库的表字段，用户存放数据，因此用户根据业务需要去定义相关的 Field（域），一般来说，每一种对应着一种数据，用户对同一种数据进行相同的操作。

域的常用属性：

- name：指定域的名称
- type：指定域的类型
- indexed：是否索引

- stored：是否存储
- required：是否必须
- multiValued：是否多值

1.4.1 域

修改 solrhome 的 schema.xml 文件 设置业务系统 Field

```
<field name="item_goodsid" type="long" indexed="true" stored="true"/>

<field name="item_title" type="text_ik" indexed="true" stored="true"/>

<field name="item_price" type="double" indexed="true" stored="true"/>

<field name="item_image" type="string" indexed="false" stored="true" />

<field name="item_category" type="string" indexed="true" stored="true" />

<field name="item_seller" type="text_ik" indexed="true" stored="true" />

<field name="item_brand" type="string" indexed="true" stored="true" />
```

1.4.2 复制域

复制域的作用在于将某一个 Field 中的数据复制到另一个域中

```
<field name="item_keywords" type="text_ik" indexed="true" stored="false" multiValued="true"/>

<copyField source="item_title" dest="item_keywords"/>

<copyField source="item_category" dest="item_keywords"/>

<copyField source="item_seller" dest="item_keywords"/>

<copyField source="item_brand" dest="item_keywords"/>
```

1.4.3 动态域

当我们需要动态扩充字段时，我们需要使用动态域。对于品优购，规格的值是不确定的，所以我们需要使用动态域来实现。需要实现的效果如下：

```
{
  "id": "4",
  "item_title": "测试商品  移动4G 5寸",
  "item_price": 444,
  "item_goodsid": 149187842867940,
  "item_category": "手机",
  "item_brand": "华为",
  "item_spec_网络制式": "3G",
  "item_spec_屏幕尺寸": "5寸",
  "item_seller": "华为专卖店",
  "_version_": 1576982705199906800
},
```

配置：

```
<dynamicField name="item_spec_*" type="string" indexed="true" stored="true" />
```

2.Spring Data Solr 入门

2.1 Spring Data Solr 简介

虽然支持任何编程语言的能力具有很大的市场价值，你可能感兴趣的问题是：我如何将 Solr 的应用集成到 Spring 中？可以，Spring Data Solr 就是为了方便 Solr 的开发所研制的一个框架，其底层是对 SolrJ（官方 API）的封装。

2.2 Spring Data Solr 入门小 Demo

2.2.1 搭建工程

（1）创建 maven 工程，pom.xml 中引入依赖

```
<dependencies>

  <dependency>
```

```
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-solr</artifactId>
        <version>1.5.5.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>4.2.4.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.9</version>
    </dependency>
  </dependencies>
```

（2）在 src/main/resources 下创建　applicationContext-solr.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:solr="http://www.springframework.org/schema/data/solr"
    xsi:schemaLocation="http://www.springframework.org/schema/data/solr
        http://www.springframework.org/schema/data/solr/spring-solr-1.0.xsd
```

```
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- solr服务器地址 -->
    <solr:solr-server id="solrServer" url="http://127.0.0.1:8080/solr" />
    <!-- solr模板，使用solr模板可对索引库进行CRUD的操作 -->
    <bean id="solrTemplate" class="org.springframework.data.solr.core.SolrTemplate">
        <constructor-arg ref="solrServer" />
    </bean>
</beans>
```

2.2.2 @Field 注解

创建 cn.itcast.pojo 包，将品优购的 TbItem 实体类拷入本工程 ，属性使用@Field 注解标识 。如果属性与配置文件定义的域名称不一致，需要在注解中指定域名称。

```
public class TbItem implements Serializable{

    @Field
    private Long id;

    @Field("item_title")
    private String title;
```

```
    @Field("item_price")
    private BigDecimal price;

    @Field("item_image")
    private String image;

    @Field("item_goodsid")
    private Long goodsId;

    @Field("item_category")
    private String category;

    @Field("item_brand")
    private String brand;

    @Field("item_seller")
    private String seller;
    .......
}
```

2.2.3 增加（修改）

创建测试类 TestTemplate.java

```
@RunWith(SpringJUnit4ClassRunner.class)
```

```
@ContextConfiguration(locations="classpath:applicationContext-solr.xml")

public class TestTemplate {



    @Autowired

    private SolrTemplate solrTemplate;



    @Test

    public void testAdd(){

        TbItem item=new TbItem();

        item.setId(1L);

        item.setBrand("华为");

        item.setCategory("手机");

        item.setGoodsId(1L);

        item.setSeller("华为 2 号专卖店");

        item.setTitle("华为 Mate9");

        item.setPrice(new BigDecimal(2000));

        solrTemplate.saveBean(item);

        solrTemplate.commit();

    }

}
```

2.2.4 按主键查询

```
    @Test
```

```
public void testFindOne(){

    TbItem item = solrTemplate.getById(1, TbItem.class);

    System.out.println(item.getTitle());

}
```

2.2.5 按主键删除

```
@Test

public void testDelete(){

    solrTemplate.deleteById("1");

    solrTemplate.commit();

}
```

2.2.6 分页查询

首先循环插入 100 条测试数据

```
@Test

public void testAddList(){

    List<TbItem> list=new ArrayList();



    for(int i=0;i<100;i++){

        TbItem item=new TbItem();

        item.setId(i+1L);

        item.setBrand("华为");

        item.setCategory("手机");

        item.setGoodsId(1L);
```

```
            item.setSeller("华为 2 号专卖店");
            item.setTitle("华为 Mate"+i);
            item.setPrice(new BigDecimal(2000+i));
            list.add(item);
        }

        solrTemplate.saveBeans(list);
        solrTemplate.commit();
    }
```

编写分页查询测试代码：

```
    @Test
    public void testPageQuery(){
        Query query=new SimpleQuery("*:*");
        query.setOffset(20);//开始索引（默认 0）
        query.setRows(20);//每页记录数(默认 10)
        ScoredPage<TbItem> page = solrTemplate.queryForPage(query, TbItem.class);
        System.out.println("总记录数： "+page.getTotalElements());
        List<TbItem> list = page.getContent();
        showList(list);
    }
    //显示记录数据
```

```
    private void showList(List<TbItem> list){
        for(TbItem item:list){
            System.out.println(item.getTitle() +item.getPrice());
        }
    }
```

2.2.7 条件查询

Criteria 用于对条件的封装：

```
@Test
    public void testPageQueryMutil(){
        Query query=new SimpleQuery("*:*");
        Criteria criteria=new Criteria("item_title").contains("2");
        criteria=criteria.and("item_title").contains("5");
        query.addCriteria(criteria);
        //query.setOffset(20);//开始索引（默认 0）
        //query.setRows(20);//每页记录数(默认 10)
        ScoredPage<TbItem> page = solrTemplate.queryForPage(query, TbItem.class);
        System.out.println("总记录数："+page.getTotalElements());
        List<TbItem> list = page.getContent();
        showList(list);
    }
```

2.2.8 删除全部数据

```
@Test
public void testDeleteAll(){
    Query query=new SimpleQuery("*:*");
    solrTemplate.delete(query);
    solrTemplate.commit();
}
```

3.品优购-批量数据导入

3.1 需求分析

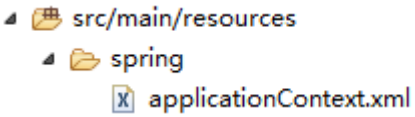
编写专门的导入程序，将商品数据导入到 Solr 系统中

3.2 查询商品数据列表

3.2.1 工程搭建

（1）创建 pinyougou-solr-util（jar） ,引入 pinyougou-dao 以及 spring 相关依赖

（2）创建 spring 配置文件



内容为：

```
<context:component-scan base-package="com.pinyougou.solrutil">
</context:component-scan>
```

3.2.2 代码编写

创建 com.pinyougou.solrutil 包，创建类 SolrUtil ,实现商品数据的查询(已审核商品)

```
@Component
public class SolrUtil {

    @Autowired
    private TbItemMapper itemMapper;



    /**
     * 导入商品数据
     */
    public void importItemData(){
        TbItemExample example=new TbItemExample();
        Criteria criteria = example.createCriteria();
        criteria.andStatusEqualTo("1");//已审核
        List<TbItem> itemList = itemMapper.selectByExample(example);
        System.out.println("===商品列表===");
        for(TbItem item:itemList){
            System.out.println(item.getTitle());
        }
        System.out.println("===结束===");
    }


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

```
        ApplicationContext context=new
ClassPathXmlApplicationContext("classpath*:spring/applicationContext*.xml");

        SolrUtil solrUtil=  (SolrUtil) context.getBean("solrUtil");

        solrUtil.importItemData();

    }

}
```

3.3 数据导入 Solr 索引库

3.3.1 实体类

（1）将 demo 工程中添加了@Field 注解的实体类拷贝到 pinyougou-pojo 中

（2）在 pinyougou-pojo 中引入依赖

```
    <dependency>

        <groupId>org.springframework.data</groupId>

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

        <version>1.5.5.RELEASE</version>

    </dependency>
```

3.3.2 添加 Solr 配置文件

添加 applicationContext-solr.xml 到 spring 目录

```
    <!-- solr 服务器地址 -->

    <solr:solr-server id="solrServer" url="http://127.0.0.1:8080/solr" />

    <!-- solr 模板，使用 solr 模板可对索引库进行 CRUD 的操作 -->
```

```
<bean id="solrTemplate" class="org.springframework.data.solr.core.SolrTemplate">
    <constructor-arg ref="solrServer" />
</bean>
```

3.2.3 调用模板类导入 solr

修改 pinyougou-solr-util 的 SolrUtil.java

```
@Autowired
private SolrTemplate solrTemplate;


/**
 * 导入商品数据
 */
public void importItemData(){

    TbItemExample example=new TbItemExample();
    Criteria criteria = example.createCriteria();
    criteria.andStatusEqualTo("1");//已审核
    List<TbItem> itemList = itemMapper.selectByExample(example);
    System.out.println("===商品列表===");
    for(TbItem item:itemList){
        System.out.println(item.getTitle());
    }
    solrTemplate.saveBeans(itemList);
```

```
        solrTemplate.commit();

        System.out.println("===结束===");

    }
```

3.4 规格导入动态域

3.4.1@Dynamic 注解

修改 TbItem.java ， 添加属性

```
    @Dynamic

    @Field("item_spec_*")

    private Map<String,String> specMap;

    public Map<String, String> getSpecMap() {

        return specMap;

    }

    public void setSpecMap(Map<String, String> specMap) {

        this.specMap = specMap;

    }
```

3.4.2 修改导入工具

修改 pinyougou-solr-util 的 SolrUtil.java ,引入 fastJSON 依赖

```
/**

     * 导入商品数据

     */

    public void importItemData(){
```

```
        TbItemExample example=new TbItemExample();

        Criteria criteria = example.createCriteria();

        criteria.andStatusEqualTo("1");//已审核

        List<TbItem> itemList = itemMapper.selectByExample(example);

        System.out.println("===商品列表===");

        for(TbItem item:itemList){

            Map specMap= JSON.parseObject(item.getSpec());//将 spec 字段中的 json 字符串转换为 map

            item.setSpecMap(specMap);//给带注解的字段赋值

            System.out.println(item.getTitle());

        }

        solrTemplate.saveBeans(itemList);

        solrTemplate.commit();

        System.out.println("===结束===");

    }
```

4.品优购-关键字搜索

4.1 需求分析

打开搜索页面，在搜索框输入要搜索的关键字，点击搜索按钮即可进行搜索，展示搜索结果

4.2 后端代码

4.2.1 服务接口层

（1）创建 pinyougou-search-interface 模块（搜索服务接口），依赖 pinyougou-pojo

（2）创建 com.pinyougou.search.service 包，创建业务接口

```
public interface ItemSearchService {

    /**
     * 搜索
     * @param keywords
     * @return
     */
    public Map<String,Object> search(Map searchMap);

}
```

4.2.2 服务实现层

（1）创建 war 工程 pinyougou-search-service ，引入 pinyougou-search-interface spring dubbox 等相关依赖（参加其它服务工程）Tomcat 运行端口 9004

（2）添加 web.xml （参加其它服务工程）

（3）在 src/main/resources/spring 下 applicationContext-service.xml （参见其它服务工程）dubbo 端口 20884

（4）在 src/main/resources/spring 下创建 spring 配置文件 applicationContext-solr.xml（同 demo 工程）

（5）编写服务实现类 ItemSearchServiceImpl.java

```
@Service(timeout=3000)

public class ItemSearchServiceImpl implements ItemSearchService{
```

```
    @Autowired
    private SolrTemplate solrTemplate;



    @Override
    public Map<String, Object> search(Map searchMap) {
         Map<String,Object> map=new HashMap<>();
        Query query=new SimpleQuery();
        //添加查询条件
        Criteria criteria=new
Criteria("item_keywords").is(searchMap.get("keywords"));
        query.addCriteria(criteria);
        ScoredPage<TbItem> page = solrTemplate.queryForPage(query, TbItem.class);
        map.put("rows", page.getContent());
        return map;
    }
}
```

4.2.3 控制层

（1）创建 pinyougou-search-web 工程，引入依赖（参见其它 web 模块），tomcat 运行端口 9104

（2）添加 web.xml （参加其它 web 工程）

（3）添加配置文件 （内容参加其它 web 工程）

（4）创建包 com.pinyougou.search.controller 编写控制器类

```
@RestController
```

```
@RequestMapping("/itemsearch")
public class ItemSearchController {

    @Reference
    private ItemSearchService itemSearchService;



    @RequestMapping("/search")
    public Map<String, Object> search(@RequestBody Map searchMap ){
        return  itemSearchService.search(searchMap);
    }
}
```

4.3 前端代码

4.3.1 拷贝资源

将下列资源拷贝至 pinyougou-search-web

将 angularJS 拷贝到插件文件夹

拷贝 base.js 到 js 文件夹

4.3.2 服务层

pinyougou-search-web 工程创建 searchService.js

```
//搜索服务层
app.service("searchService",function($http){
    this.search=function(searchMap){
        return $http.post('itemsearch/search.do',searchMap);
    }
});
```

4.3.3 控制层

pinyougou-search-web 工程 searchController.js

```
app.controller('searchController',function($scope,searchService){
    //搜索
    $scope.search=function(){
        searchService.search( $scope.searchMap ).success(
            function(response){
                $scope.resultMap=response;//搜索返回的结果
            }
        );
    }
});
```

4.3.4 页面

pinyougou-search-web 工程 search.html

引入 js

```
<script type="text/javascript" src="plugins/angularjs/angular.min.js"> </script>

<script type="text/javascript" src="js/base.js"> </script>

<script type="text/javascript" src="js/service/searchService.js"> </script>

<script type="text/javascript" src="js/controller/searchController.js"> </script>
```

指定控制器

```
<body ng-app="pinyougou" ng-controller="searchController">
```

绑定搜索框

```
<div class="input-append">

    <input type="text" id="autocomplete" ng-model="searchMap.keywords" type="text" class="input-error input-xxlarge" />

    <button class="sui-btn btn-xlarge btn-danger" ng-click="search()" type="button">搜索</button>

</div>
```

循环显示数据

```
<li class="yui3-u-1-5" ng-repeat="item in resultMap.rows">

    <div class="list-wrap">

        <div class="p-img">

            <a href="item.html" target="_blank"><img src="{{item.image}}" /></a>

        </div>

        <div class="price">

            <strong>

                <em>¥</em>

                <i>{{item.price}}</i>
```

```
            </strong>
        </div>
        <div class="attr">
            <em>{{item.title}}</em>
        </div>
        <div class="cu">
            <em><span>促</span>满一件可参加超值换购</em>
        </div>
        <div class="commit">
            <i class="command">已有 2000 人评价</i>
        </div>
        <div class="operate">
            <a href="success-cart.html" target="_blank" class="sui-btn btn-bordered
btn-danger">加入购物车</a>
            <a href="javascript:void(0);" class="sui-btn btn-bordered">对比</a>
            <a href="javascript:void(0);" class="sui-btn btn-bordered">关注</a>
        </div>
    </div>
</li>
```

5.搜索页与首页对接

5.1 需求分析

用户在首页的搜索框输入关键字，点击搜索后自动跳转到搜索页查询

5.2 代码实现

5.2.1 首页传递关键字

修改 pinyougou-portal-web 的 contentController.js

```
//搜索跳转

$scope.search=function(){

    location.href="http://localhost:9104/search.html#?keywords="+$scope.keywords;

}
```

修改 pinyougou-portal-web 的 index.html

```
<div class="input-append">

    <input type="text" id="autocomplete" type="text" ng-model="keywords"
class="input-error input-xxlarge" />

    <button class="sui-btn btn-xlarge btn-danger" ng-click="search()" type="button">
搜索</button>

</div>
```

5.2.2 搜索页接收关键字

修改 pinyougou-search-web 的 searchController.js

添加 location 服务用于接收参数

```
app.controller('searchController',function($scope,$location,searchService){

......
```

接收参数并进行查询

```
    //加载查询字符串

    $scope.loadkeywords=function(){
```

```
        $scope.searchMap.keywords=  $location.search()['keywords'];

        $scope.search();

    }
```