项目实训报告

题　　目：Kafka 搭建与简单编程

专业班级：19 软件技术 3-1 班

指导教师：魏勇

目录

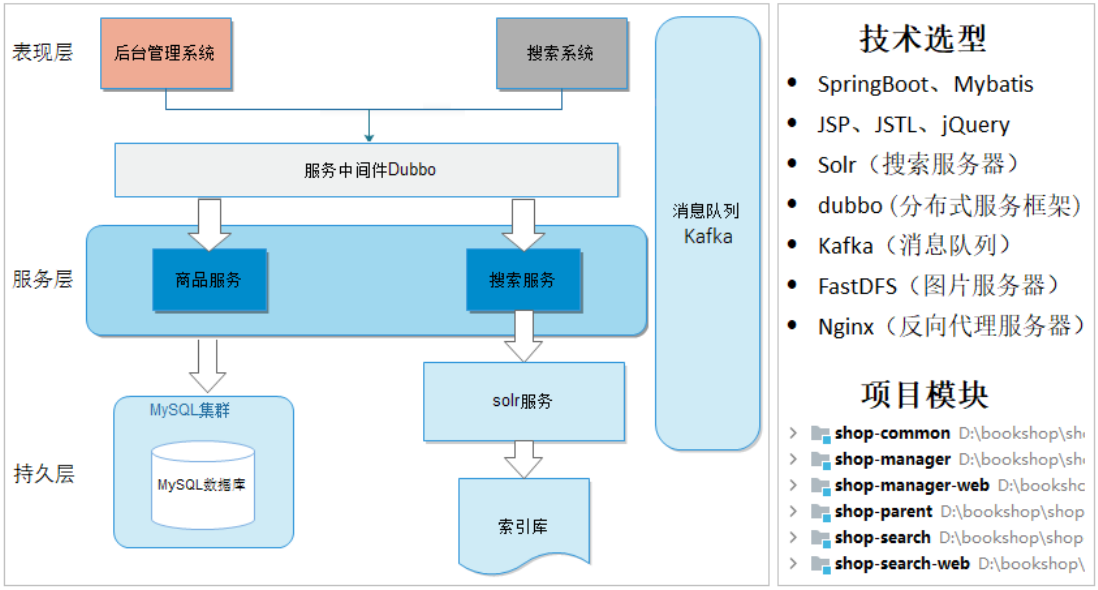
1.项目名称

Kafka 搭建与简单编程

2.项目简介

本项目的目标是在 Linux 操作系统下搭建 Zookeeper 和 Kafka 集群并对其实现简单编程，利用 Zookeeper 对 Kafka 集群进行调度。以分布式架构编写图书后台管理系统和搜索系统，使用 Kafka 作为消息中间件进行系统间的通知，管理员在后台管理系统对图书进行操作时，（新增、删除、上下架等），Kafka 给予搜索服务发送通知并通知 SOLR 服务更新索引库。

系统架构图及技术选型



项目各模块分析

使用单独的 Maven 工程构建系统的基本框架：

shop-parent	Maven 依赖的统一管理
shop-common	存放公用的类或功能模块

使用聚合工程搭建后台系统服务层：

shop-manager	聚合工程的父工程，负责把所有子工程聚合在一起
shop-manager-bean	聚合工程的子工程，JavaBean
shop-manager-mapper	聚合工程的子工程，DAO 层
shop-manager-interface	聚合工程的子工程，业务处理层的接口
shop-manager-service	聚合工程的子工程，业务处理层

使用聚合工程搭建搜索系统服务层：

shop-search	聚合工程的父工程，负责把所有子工程聚合在一起
shop-search-interface	聚合工程的子工程，业务处理层的接口
shop-search-service	聚合工程的子工程，业务处理层

使用单独的 Maven 工程搭建后台系统和搜索系统表现层：

shop-manager-web	单独的 Maven 工程，页面控制层
shop-search-web	单独的 Maven 工程，页面控制层

3.项目分工

表 1 列出项目小组成员及分工。

表 1 项目分工情况表

序号	学号	姓名	班级	项目分工	备注
1	1901010221	梁宇成	19 软件 3-1 班	搭建 Kafka 环境 构建系统主框架 搜索系统的开发	
2	1901010222	梁志嵩	19 软件 3-1 班	项目模块的测试	
3	1901010231	彭鸿伟	19 软件 3-1 班	前端页面的开发 后台系统的开发	
4	1901010730	容鑫	19 软件 3-1 班	公共模块的开发	

4.工作任务分配

表 2 列出本项目组完成的主要模块及开发者。

表 2 项目模块及开发者明细

序号	开发者	模块名[1]	开发语言[2]	开发形式	备注
1	梁宇成	shop-search shop-search-web	Java、H5	☑新增 ☑修改 ☑测试	
2	彭鸿伟	shop-manager shop-search-web	Java、H5	☑新增 ☑修改 ☐测试	
3	容鑫	shop-common	Java	☑新增 ☐修改 ☐测试	
4	梁志嵩	所有项目模块	Java	☐新增 ☐修改 ☑测试	

[1] 模块指用程序代码编写的方法或 H5 编写的页面等。如果是程序代码编写的方法，需列举该方法的参数及类型，返回参数类型及意义。如果用 Java 编写，还需注明所在的包名和类名。

[2] 开发语言包括 Java, H5, JavaScript 和其它。

5.开发过程

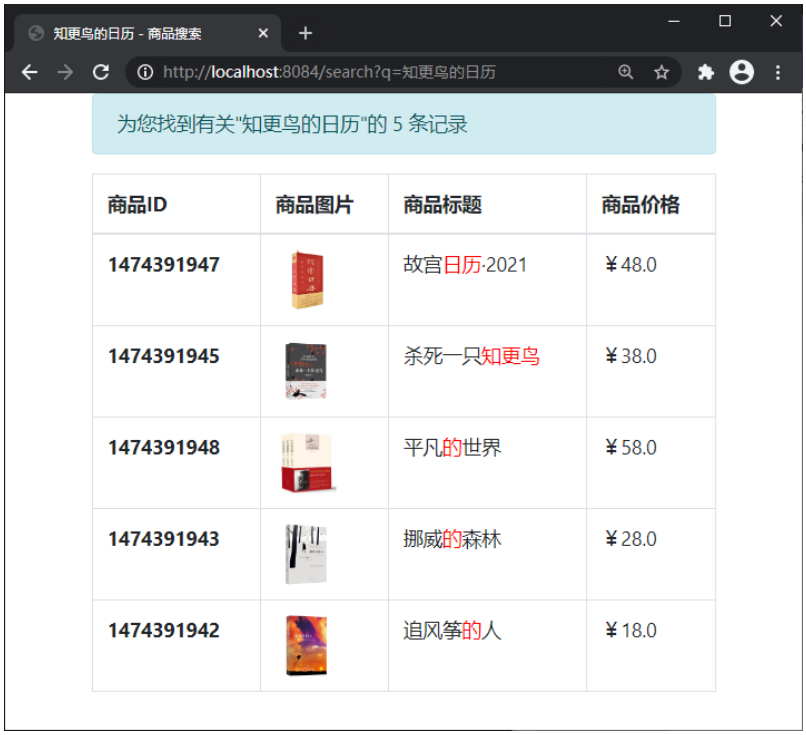
5.1 shop-search, shop-search-web 模块实现[梁宇成]

功能描述：本模块实现了图书条目的搜索功能，引入 Apache SOLR 由其提供搜索服务。用户在前端键入字符串时，SOLR 服务会自动将字符串拆分成多个关键字，并在搜索结果页的标题（书名）关键字给予红色高亮显示。值得注意的是，搜索的结果是来源于 SOLR 服务所构建好的索引库，而不是直接从 MySQL 数据库中进行查询。

运行与测试：

①打开 http://localhost:8084/输入需要搜索的图书关键字，可以是多个关键字。

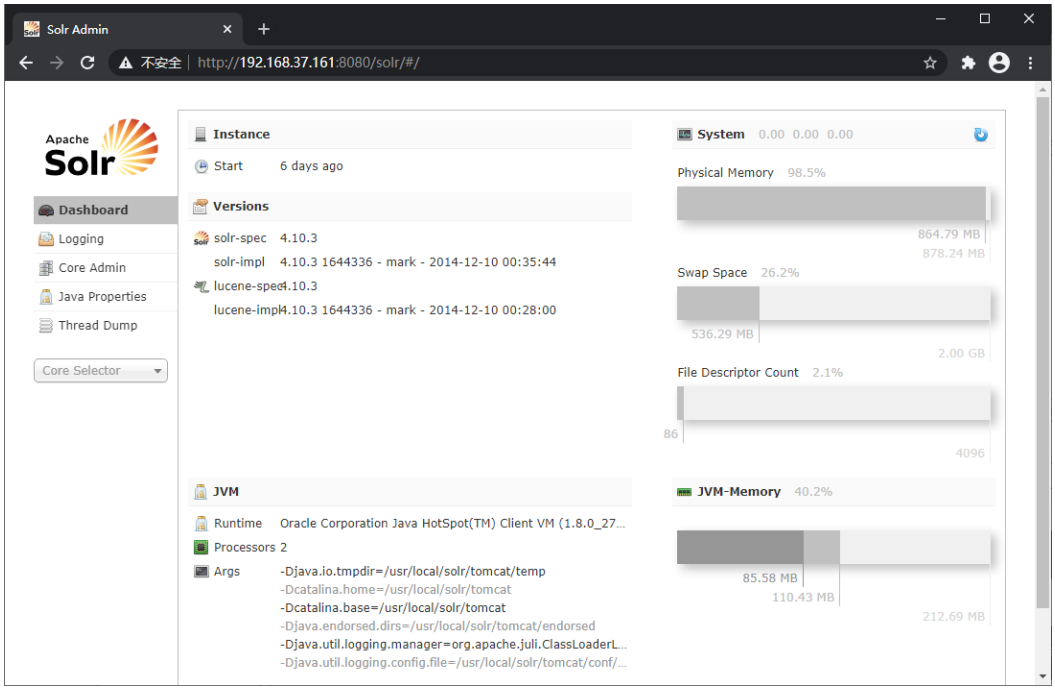
②通过观察查询结果页发现：之前输入的字符串被自动拆分成多个关键字去查询图书，被拆分的关键字注有红色高亮字体。



技术分析：

(1)SOLR 服务的配置简介

①实现搜索系统的全局功能，首先要在 Linux 服务器上搭建好 SOLR。



②要实现图书搜索的功能，需要配置商品的业务域，图书的数据库表：

```
CREATE TABLE `tb_item` (
  `id` bigint(10) NOT NULL AUTO_INCREMENT COMMENT '商品id，同时也是商品编号',
  `title` varchar(100) NOT NULL COMMENT '商品标题',
  `sell_point` varchar(150) DEFAULT NULL COMMENT '商品卖点',
  `price` bigint(20) NOT NULL COMMENT '商品价格，单位为：分',
  `num` int(10) NOT NULL COMMENT '库存数量',
  `barcode` varchar(30) DEFAULT NULL COMMENT '商品条形码',
  `image` varchar(500) DEFAULT NULL COMMENT '商品图片',
  `cid` bigint(10) NOT NULL COMMENT '所属类目，叶子类目',
  `status` tinyint(4) NOT NULL DEFAULT '1' COMMENT '商品状态，1-正常，2-下架，
  `created` datetime NOT NULL COMMENT '创建时间',
  `updated` datetime NOT NULL COMMENT '更新时间',
```

③搜索图书标题需要中文分词，需要配置中文分词器 IK。

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 /usr/local/solr/tomcat/webapps/solr/WEB-INF/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
```

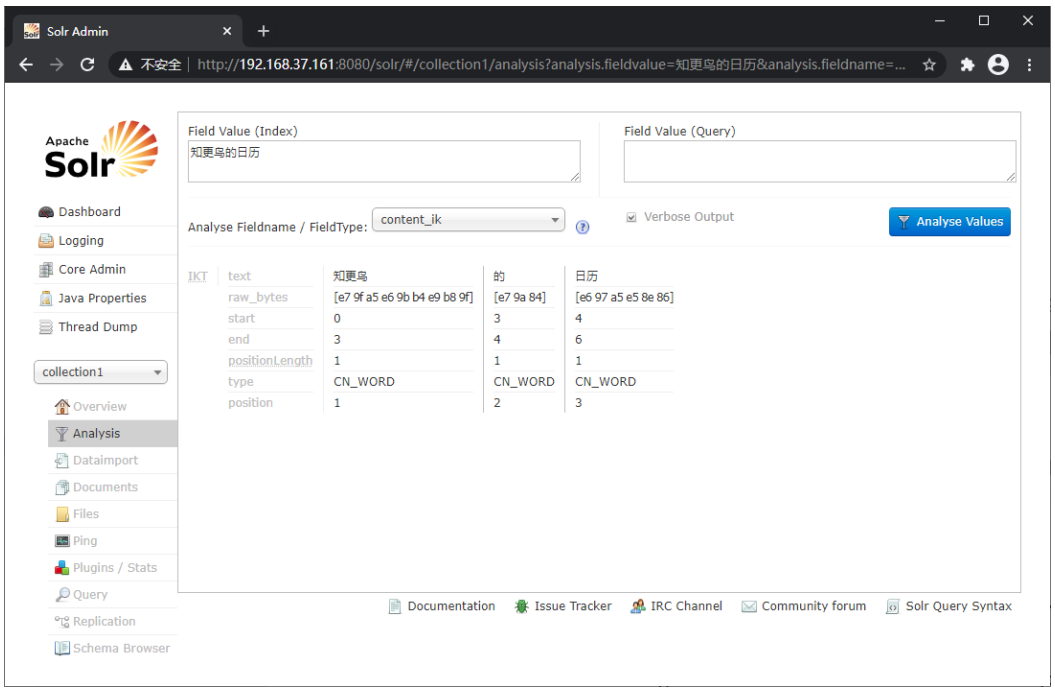
在最后面添加如下：

```
    <!-- IKAnalyzer  声明一种分词器-->
     <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

```
/usr/local/solr/tomcat/bin/startup.sh
```

测试中文分词功能：输入“知更鸟的日历”，分词效果如下：



④根据商品数据库表配置业务域，编辑 schema.xml 文件。

根据商品数据库表配置业务域，仍然是编辑 schema.xml 文件。

cd /usr/local/solr/solrhome/collection1/conf/

vim schema.xml

以下的字段就是我们的商品表构建索引需要的字段，也就是说，我们搜索某一件商品，然后跳转到商品列表页面，这些字段已经足以显示所有内容了。

```
<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" />
```

```
<!-- 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" />

<!-- taotao -->
<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

/usr/local/solr/tomcat/bin/shutdown.sh

/usr/local/solr/tomcat/bin/startup.sh

(2)实现图书的搜索功能，编写 Service 层

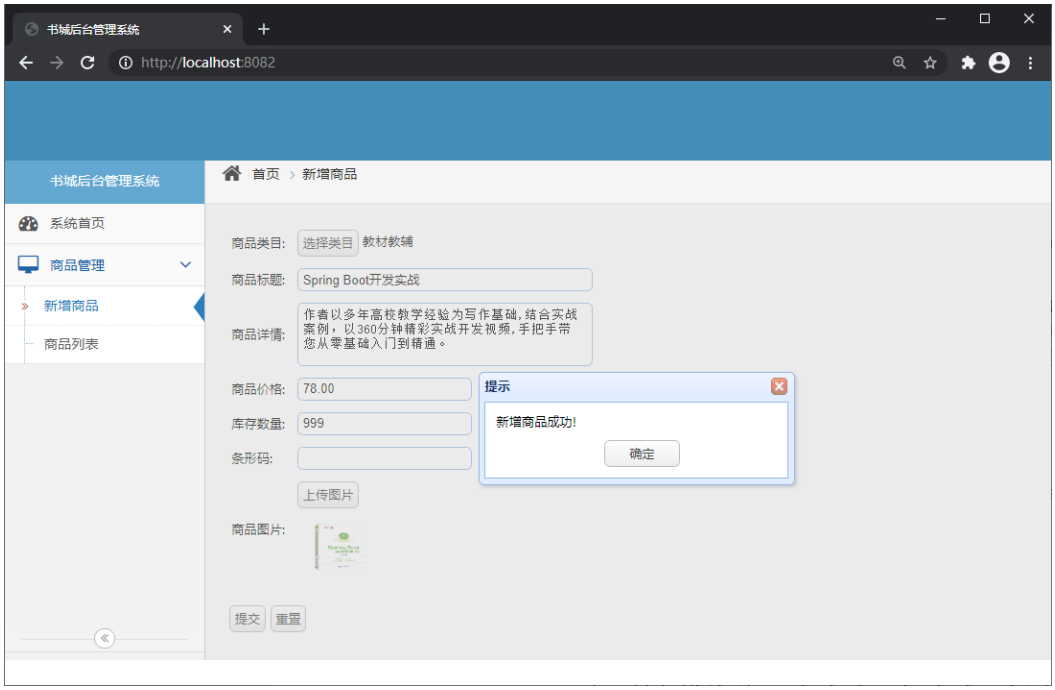
```java
@Service // 这里得使用Dubbo的注解
public class SearchServiceImpl implements SearchService {
    @Autowired
    private SolrClient solrClient;

    @Override
    public BookResult<Item> search(String query, int page, int rows) {
        // 封装查询对象
        SolrQuery solrQuery = new SolrQuery();
        // 设置查询语句
        if (StringUtils.isNotBlank(query)) {
            solrQuery.setQuery("item_title:" + query + " AND item_status:1");
        } else {
            solrQuery.setQuery("item_status:1");
        }
        // 设置分页
        solrQuery.setStart((page - 1) * rows);
        solrQuery.setRows(rows);
        // 设置高亮
        solrQuery.setHighlight(true);
        solrQuery.addHighlightField("item_title");
        solrQuery.setHighlightSimplePre("<font color='red'>");
        solrQuery.setHighlightSimplePost("</font>");
        // 声明返回结果对象bookResult
        BookResult<Item> bookResult = new BookResult<>();
        try {
            // 执行查询
            QueryResponse response = this.solrClient.query(solrQuery);
            SolrDocumentList results = response.getResults();
            // 获取高亮数据
            Map<String, Map<String, List<String>>> map = response.getHighlighting();
            // 解析结果集
            // 声明存放商品的集合
            List<Item> list = new ArrayList<>();
            for (SolrDocument solrDocument : results) {
                Item item = new Item();
                // 解析Document
                // 商品id
                item.setId(Long.parseLong(solrDocument.get("id").toString()));
                // 获取高亮的数据
                List<String> hlist = map.get(solrDocument.get("id").toString()).get("item_title");
                // 商品title
                if (hlist != null && hlist.size() > 0) {
                    item.setTitle(hlist.get(0));
                } else {
                    item.setTitle(solrDocument.get("item_title").toString());
                }
                // 商品图片image、价格price、分类cid
                // 由于此处代码与第62行类似，故做了省略……
                list.add(item);
            }
            // 封装返回数据bookResult
            // 设置结果集
            bookResult.setRows(list);
            // 设置查询的数据总条数
            bookResult.setTotal(results.getNumFound());
            return bookResult;
        } catch (Exception e) {
            e.printStackTrace();
        }
        // 如果查询有异常，就返回一个空的结果
        return bookResult;
    }
}
```

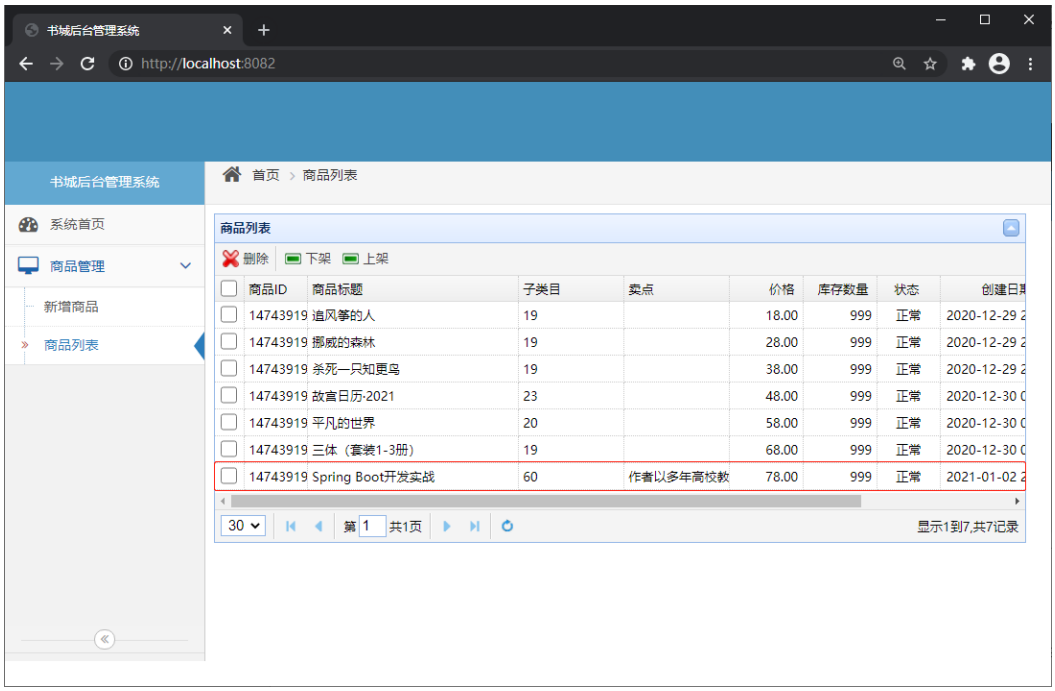
shop-manager, shop-manager-web 模块实现[彭鸿伟]

功能描述：本模块实现了对图书的新增、删除以及查询功能。新增功能是指把图书信息提交到数据库中，删除功能是指在数据库删除此条记录信息，而查询功能（商品列表）则是在商品列表的页面中显示数据库所保存的全部图书信息。

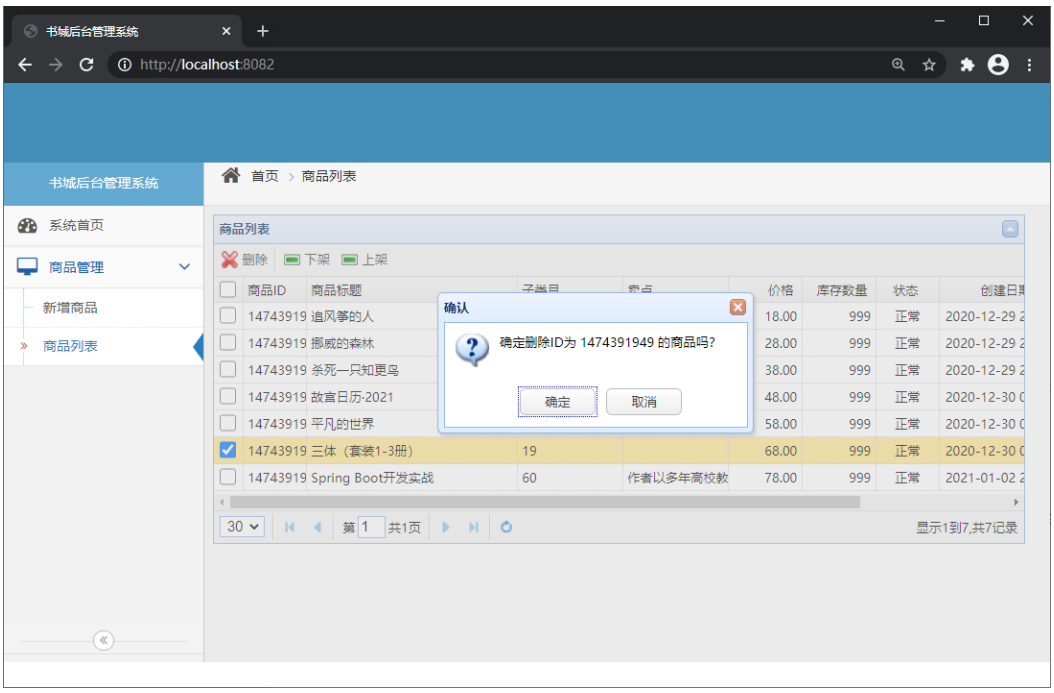
运行与测试：打开后台管理系统地址 http://localhost:8082/

(1)首先是新增图书功能，输入图书信息并点击提交后，在商品列表页查询结果。

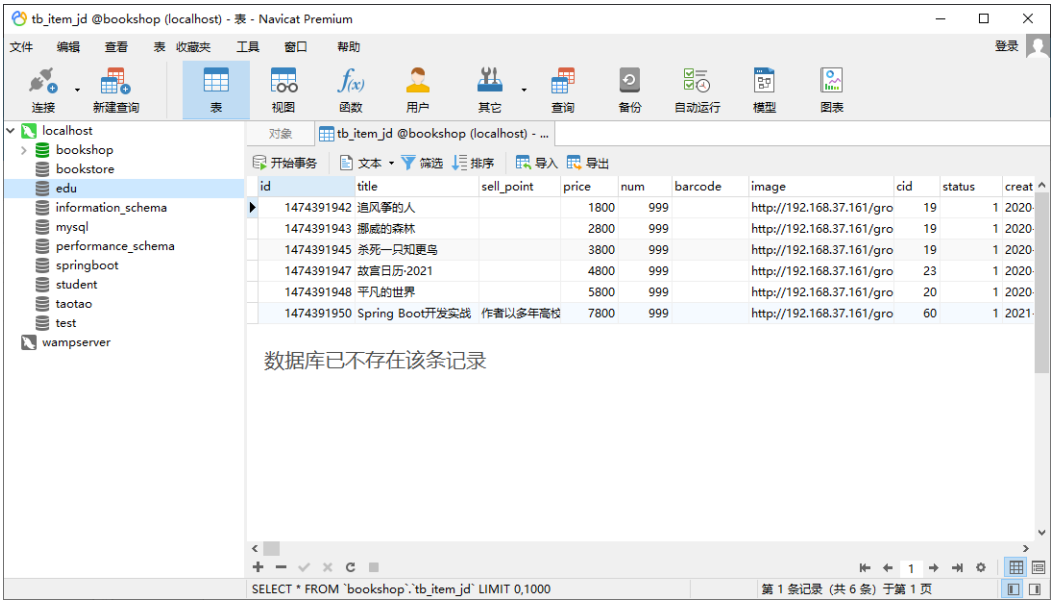




(2)然后是删除图书功能，选中图书信息并点击删除后，使用数据库工具查询结果。



(3)可见数据库中已经不存在这条记录，表明删除成功。



技术分析：

①图书新增功能的主要实现代码

(1)编写 Service 层

```java
/**
 * 新增商品
 *
 * @param item 商品的信息
 * @param desc 商品的描述
 */
@Override
public void addItem(Item item, String desc) {
    // 保存商品，设置商品的状态，商品是上架还是下架，如果商品是上架(1)，下架(0)
    item.setStatus(1);

    // 设置商品的创建时间和更新时间
    item.setCreated(new Date());
    item.setUpdated(item.getCreated());
    itemMapper.insert(item);
}
```

(2)编写 Controller 层

```java
// 新增商品
@RequestMapping("/rest/addItem")
public String saveItem(Item item, String desc) {
    itemService.addItem(item, desc);
    System.out.println("addItem: " + item);
    return "add success";
}
```

②图书删除功能的主要实现代码

(1)编写 Service 层

```java
/**
 * 删除商品
 *
 * @param itemId 商品的ID
 */
@Override
public void deleteItem(String itemId) {
    String[] itemIds = itemId.split( regex: ",");
    for (String id : itemIds) {
        itemMapper.deleteByPrimaryKey(id);
    }
}
```

(2)编写 Controller 层

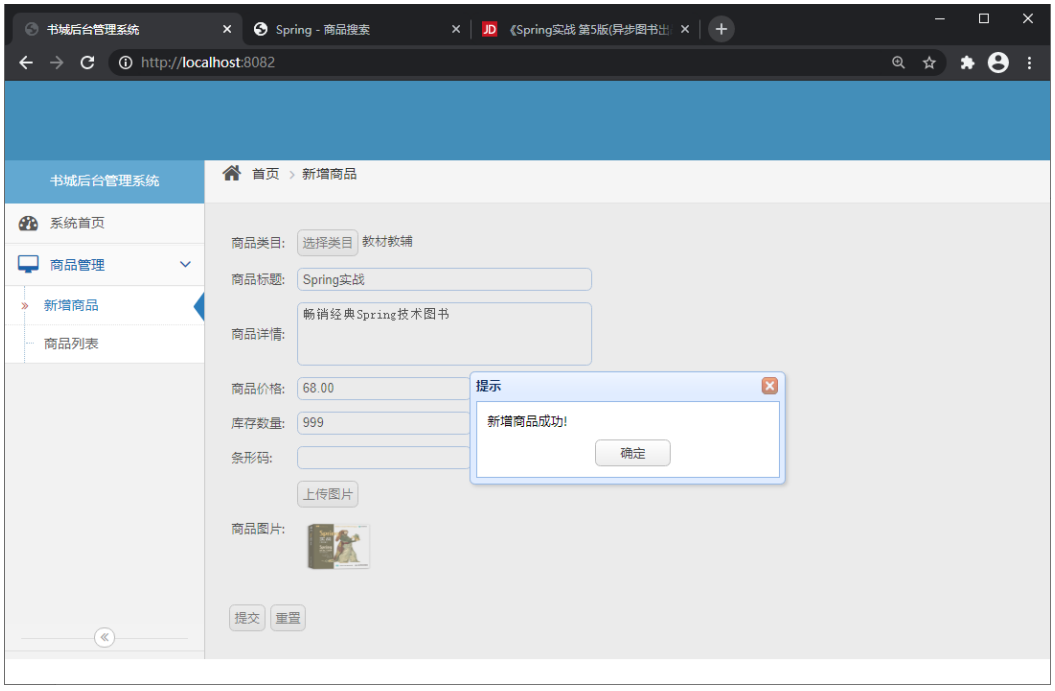
```
// 删除商品
@RequestMapping("/rest/deleteItem")
public Map<String, Object> deleteItem(@RequestParam("ids") String itemId) {
    itemService.deleteItem(itemId);
    // 返回状态码，实现操作商品后前端弹窗提示
    Map<String, Object> map = new HashMap<>();
    map.put("status", "200");
    return map;
}
```

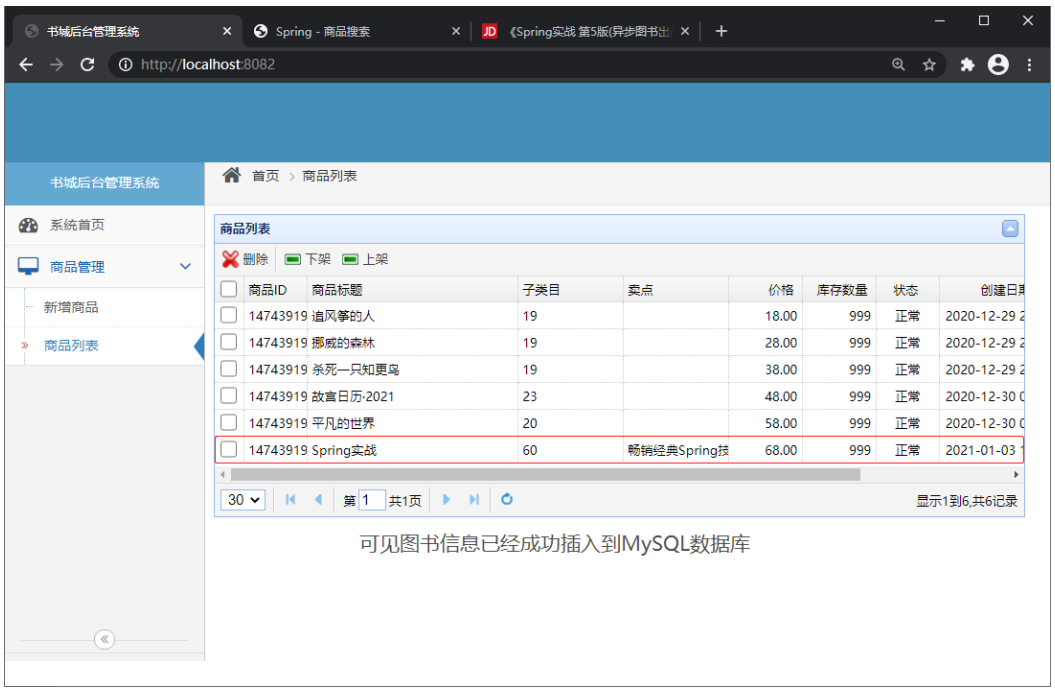
5.2 shop-search, shop-manager 模块修改[梁宇成，彭鸿伟]

功能描述：①本模块实现了对图书的上下架功能，引入 Kafka 作为消息中间件。管理员点击上架时，将选中的图书提交到 SOLR 索引库，从而在搜索系统能够查询到；点击下架后将图书记录从索引库删除，搜索系统此时将查询不到该图书记录。

②本模块完善了图书的新增和删除功能，引入 Kafka 作为消息中间件。新增图书时，将图书信息录入到数据库的基础上，同时提交到 SOLR 索引库；删除图书时，在数据库中删除记录的基础上，同时将该记录从索引库删除。

修改原因：由于后台系统实现图书的 MySQL 数据管理，而搜索系统实现图书的 SOLR 索引库管理，就会存在同步问题：当后台系统对图书进行增删等操作时，搜索系统（索引库）却没有同步更新。如图所示：





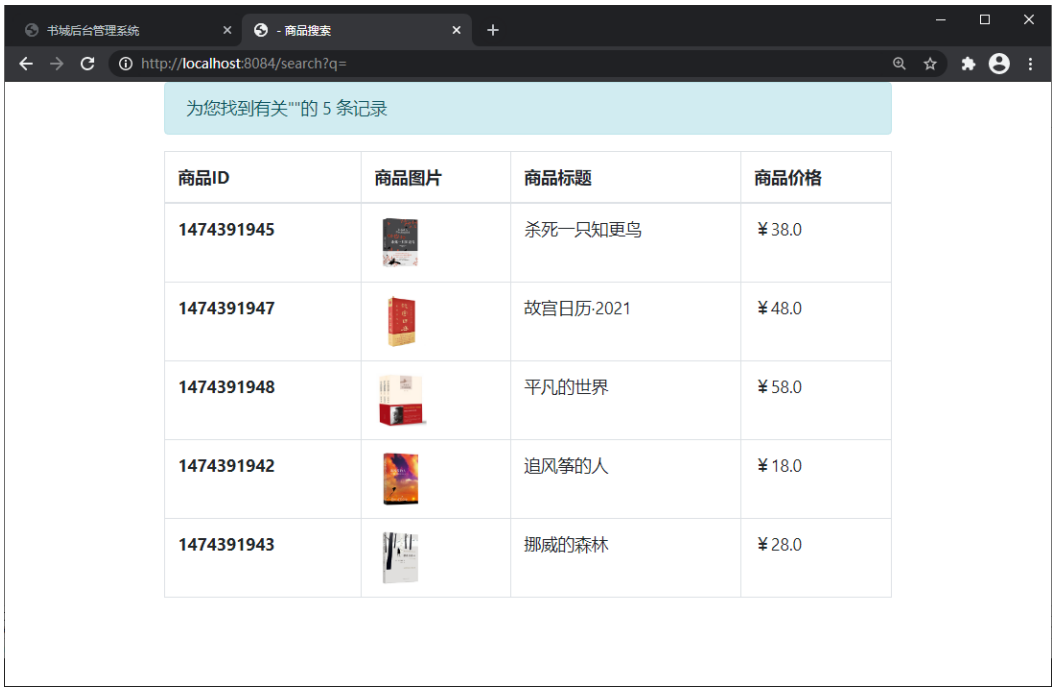
可见图书信息已经成功插入到MySQL数据库

但在搜索系统中却查找不到该记录，
显然增加的图书记录还没存到索引库中去

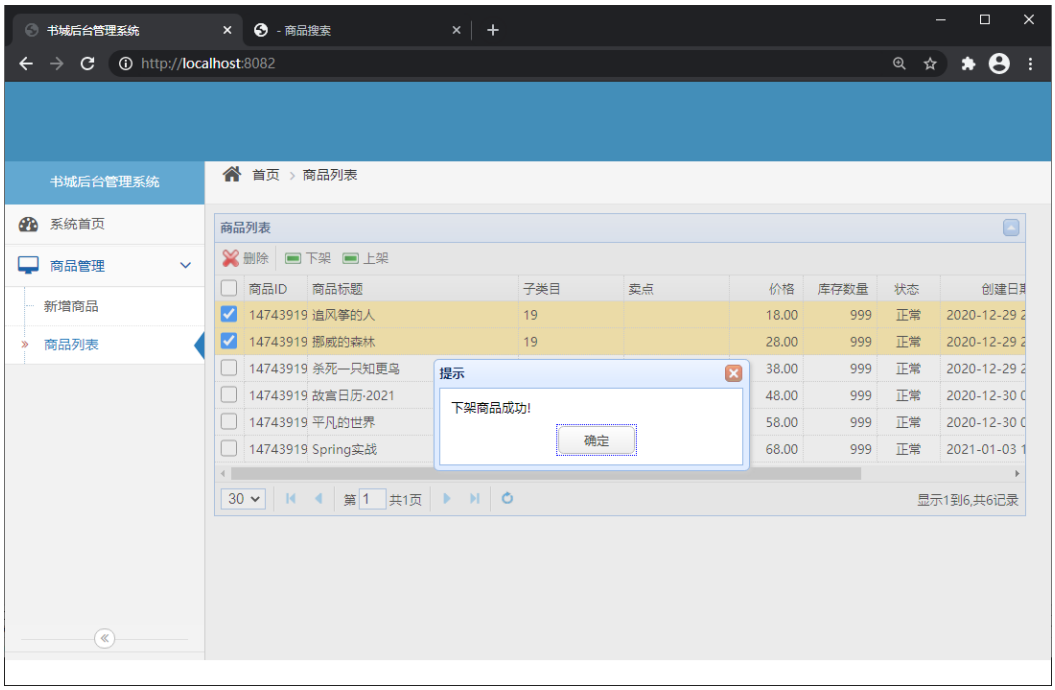
解决方案：在后台管理系统进行图书操作管理时，发送消息通知 Kafka，搜索系统监听 Kafka 是否有消息。如果没有消息，将持续监听；如果有消息，则执行更新索引库的方法。

运行与测试：

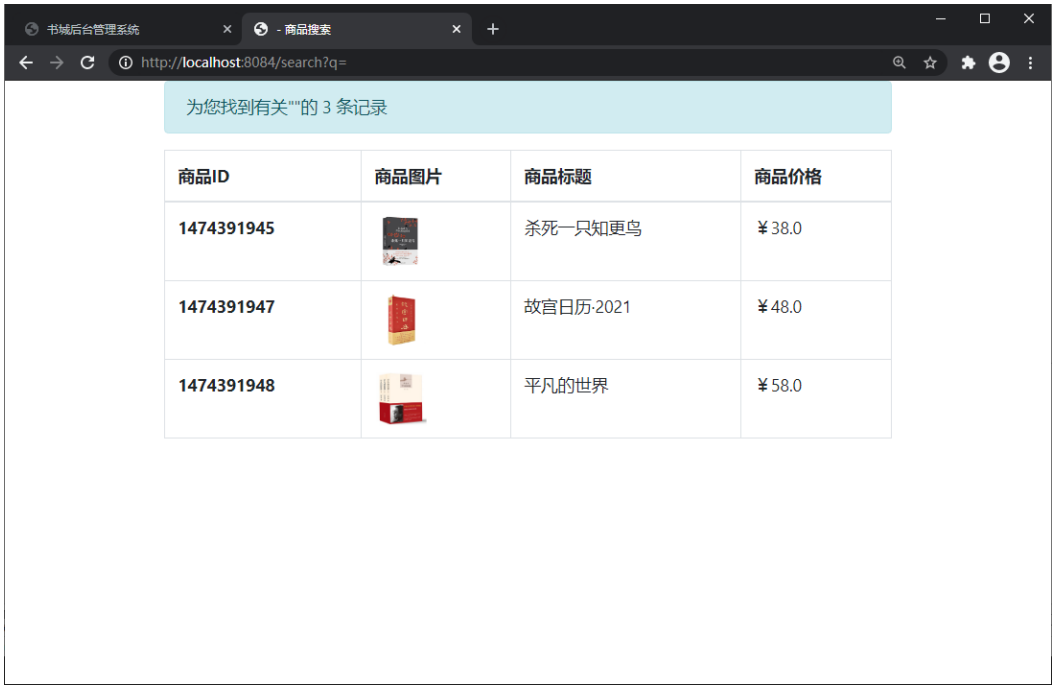
(1)首先在搜索系统查询索引库的全部记录



(2)选中其中两本图书，作下架处理（即删除索引记录）



(3)再次从搜索系统查询索引库，可见选中的两本图书已经查询不到。



技术分析：

①图书上架功能的主要实现代码

(1)编写 Service 层

```java
/**
 * 上架商品
 *
 * @param item    商品的信息
 * @param itemId 商品的ID
 */
@Override
public void putItem(Item item, String itemId) {
    String[] itemIds = itemId.split( regex: ",");
    for (String id : itemIds) {
        Long ID = Long.valueOf(id);
        item.setId(ID);
        item.setStatus(1);
        itemMapper.updateByPrimaryKeySelective(item);
        // 发送消息，通知搜索系统更新索引，只需告诉搜索系统商品的id即可
        kafkaTemplate.send( topic: "addItem",  data: "" + id);
    }
}

/**
 * 下架商品
 *
 * @param item    商品的信息
 * @param itemId 商品的ID
 */
@Override
public void removeItem(Item item, String itemId) {
    String[] itemIds = itemId.split( regex: ",");
    for (String id : itemIds) {
        Long Id = Long.valueOf(id);
        item.setId(Id);
        item.setStatus(2);
        itemMapper.updateByPrimaryKeySelective(item);
        // 发送消息，通知搜索系统更新索引，只需告诉搜索系统商品的id即可
        kafkaTemplate.send( topic: "deleteItem",  data: "" + id);
    }
}
```

(2)编写 Controller 层

```
// 上架商品
@RequestMapping("/rest/putItem")
public Map<String, Object> putItem(@RequestParam("ids") String itemId, Item item) {
    itemService.putItem(item, itemId);
    // 返回状态码，实现操作商品后前端弹窗提示
    Map<String, Object> map = new HashMap<>();
    map.put("status", "200");
    return map;
}

// 下架商品
@RequestMapping("/rest/removeItem")
public Map<String, Object> removeItem(@RequestParam("ids") String itemId, Item item) {
    itemService.removeItem(item, itemId);
    // 返回状态码，实现操作商品后前端弹窗提示
    Map<String, Object> map = new HashMap<>();
    map.put("status", "200");
    return map;
}
```

②使用 Kafka 实现消息通知的代码

(1)改造 shop-manager-service 的方法，加入发送消息逻辑。

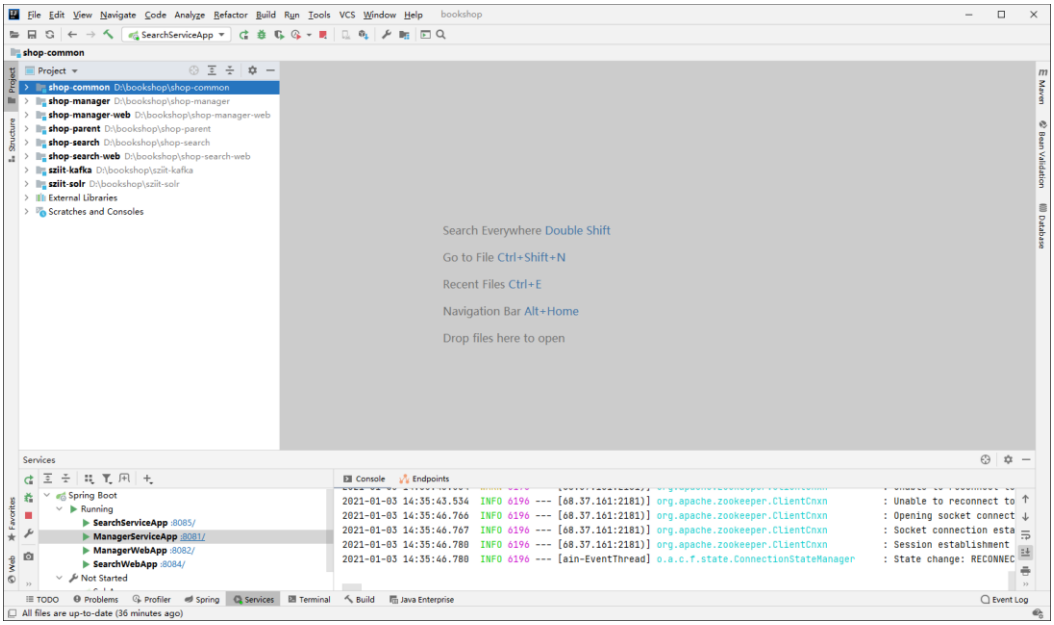
```
// 使用Kafka发送消息需要用到此对象
@Autowired
private KafkaTemplate<String, String> kafkaTemplate;

    /**
     * 新增商品
     *
     * @param item 商品的信息
     * @param desc 商品的描述
     */
    @Override
    public void addItem(Item item, String desc) {
        // 此处省略主要方法……
        // 发送消息，通知搜索系统更新索引，只需告诉搜索系统商品的id即可
        kafkaTemplate.send( topic: "addItem",  data: "" + item.getId());
    }
```

(2)添加消息监听，在 shop-search-service 方法前面添加消息监听的注解。

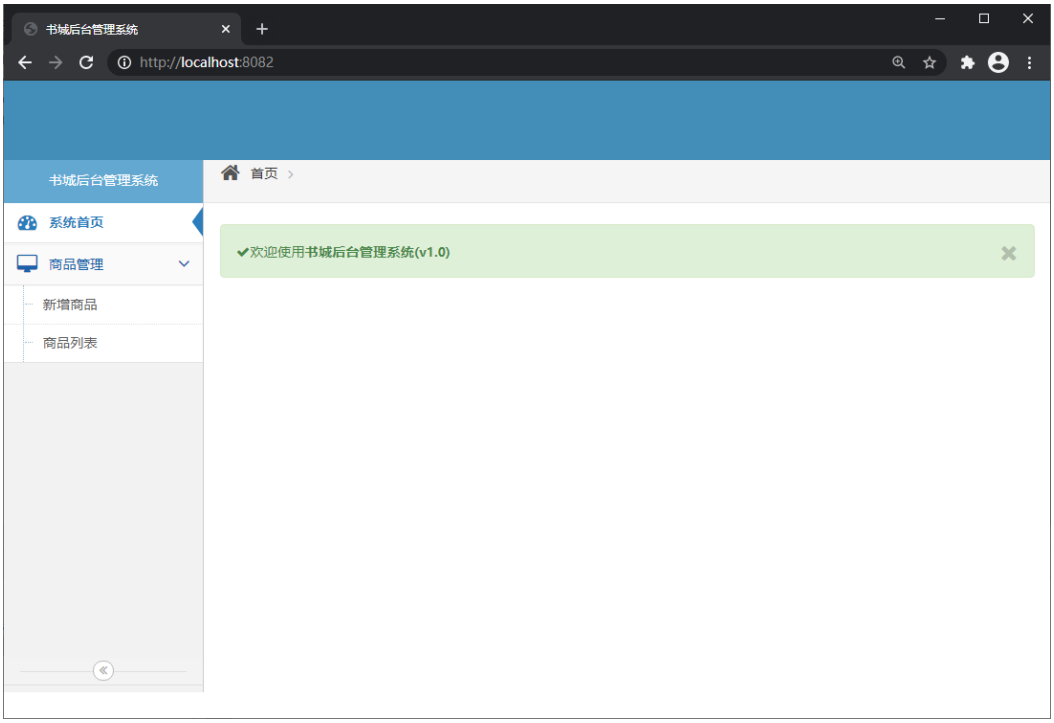
```java
@Component
public class ReceiveMessage {
    @Autowired
    private ItemMapper itemMapper;

    @Autowired
    private SolrClient solrClient;

    // 新增、上架商品：监听item的信息
    // 接收消息需要使用到这个注解(topics={"所需要监听Topic的名称"})
    @KafkaListener(topics = {"addItem"})
    public void receiveAddItemMessage(String itemId) {
        // 1. 查询数据库，根据商品id获取
        Item item = itemMapper.selectByPrimaryKey(Long.parseLong(itemId));
        System.out.println("addSearch: " + item);

        // 2. 构建索引，保存到索引库中
        SolrInputDocument document = new SolrInputDocument();
        // 商品id
        document.setField( name: "id", item.getId().toString());
        // 商品标题
        document.setField( name: "item_title", item.getTitle());
        // 商品价格、商品图片、商品类目id、商品状态
        // 由于此处与上面一行代码类似，故做了省略处理...

        try {
            // 3. 提交到Solr索引库
            solrClient.add(document);
            solrClient.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // 删除、下架商品：监听item的信息
    @KafkaListener(topics = {"deleteItem"})
    public void receiveDeleteItemMessage(String itemId) {
        // 1. 查询数据库，根据商品id获取
        Item item = itemMapper.selectByPrimaryKey(itemId);
        System.out.println("deleteSearch: " + item);

        try {
            solrClient.deleteById(itemId);
            solrClient.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5.3 bookshop 项目所有模块总测试[容鑫，梁宇成，梁志嵩]

功能描述：测试本项目所有模块的功能是否正常运行，给代码补充注释。
测试用例与测试过程：使用 IDEA，对逐个类逐个方法进行仔细检查。
①所有项目模块运行正常



②所有前端页面运行正常



③服务器（集群）运行正常
测试结果分析：无发现明显错误，所有模块运行正常，故测试通过。

6.展示图片