



品优购电商系统开发

第 14 章

SpringBoot&MQ 同步索引库

传智播客.黑马程序员



课程目标

目标 1：掌握 Spring Boot 框架的搭建方法

目标 2：掌握 MQ 基本使用方法

目标 3：运用 SpringBoot、ActiveMQ 开发同步索引库业务

1.Spring Boot 入门

1.1 什么是 Spring Boot

Spring 诞生时是 Java 企业版（Java Enterprise Edition，JEE，也称 J2EE）的轻量级替代品。无需开发重量级的 Enterprise JavaBean（EJB），Spring 为企业级 Java 开发提供了一种相对简单的方法，通过依赖注入和面向切面编程，用简单的 Java 对象（Plain Old Java Object，POJO）实现了 EJB 的功能。

虽然 Spring 的组件代码是轻量级的，但它的配置却是重量级的。一开始，Spring 用 XML 配置，而且是很多 XML 配置。Spring 2.5 引入了基于注解的组件扫描，这消除了大量针对应用程序自身组件的显式 XML 配置。Spring 3.0 引入了基于 Java 的配置，这是一种类型安全的可重构配置方式，可以代替 XML。所有这些配置都代表了开发时的损耗。因为在思考 Spring 特性配置和解决业务问题之间需要进行思维切换，所以写配置挤占了写应用程序逻辑的时间。和所有框架一样，Spring 实用，但与此同时它要求的回报也不少。

除此之外，项目的依赖管理也是件吃力不讨好的事情。决定项目里要用哪些库就已经够让人头痛的了，你还要知道这些库的哪个版本和其他库不会有冲突，这难题实在太棘手。并且，依赖管理也是一种损耗，添加依赖不是写应用程序代码。一旦选错了依赖的版本，随之而来的不兼容问题毫无疑问会是生产力杀手。

Spring Boot 让这一切成为了过去。

Spring Boot 是 Spring 社区较新的一个项目。该项目的目的是帮助开发者更容易的创建基于 Spring 的应用程序和服务，让更多的人更快的对 Spring 进行入门体验，为 Spring 生态系统提供了一种固定的、约定优于配置风格的框架。

Spring Boot 具有如下特性：

- （1）为基于 Spring 的开发提供更快的入门体验
- （2）开箱即用，没有代码生成，也无需 XML 配置。同时也可以修改默认值来满足特



定的需求。

(3) 提供了一些大型项目中常见的非功能性特性，如嵌入式服务器、安全、指标，健康检测、外部配置等。

(4) Spring Boot 并不是不对 Spring 功能上的增强，而是提供了一种快速使用 Spring 的方式。

1.2 Spring Boot 入门小 Demo

1.2.1 起步依赖

创建 Maven 工程 springboot_demo（打包方式 jar）

在 pom.xml 中添加如下依赖

```
<parent>

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

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>1.4.0.RELEASE</version>

</parent>

<dependencies>

    <dependency>

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

        <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

</dependencies>
```

我们会惊奇地发现，我们的工程自动添加了好多好多 jar 包

- 📦 Maven Dependencies
 - ▶ 📦 spring-boot-starter-web-1.4.0.RELEASE.jar -
 - ▶ 📦 spring-boot-starter-1.4.0.RELEASE.jar - D:\re
 - ▶ 📦 spring-boot-1.4.0.RELEASE.jar - D:\repositor
 - ▶ 📦 spring-boot-autoconfigure-1.4.0.RELEASE.ja
 - ▶ 📦 spring-boot-starter-logging-1.4.0.RELEASE.j
 - ▶ 📦 logback-classic-1.1.7.jar - D:\repository_ssh\
 - ▶ 📦 logback-core-1.1.7.jar - D:\repository_ssh\c
 - ▶ 📦 slf4j-api-1.7.21.jar - D:\repository_ssh\org\s
 - ▶ 📦 jcl-over-slf4j-1.7.21.jar - D:\repository_ssh\o
 - ▶ 📦 jul-to-slf4j-1.7.21.jar - D:\repository_ssh\org
 - ▶ 📦 log4j-over-slf4j-1.7.21.jar - D:\repository_ssl
 - ▶ 📦 spring-core-4.3.2.RELEASE.jar - D:\repositor
 - ▶ 📦 snakeyaml-1.17.jar - D:\repository_ssh\org\
 - ▶ 📦 spring-boot-starter-tomcat-1.4.0.RELEASE.ja
 - ▶ 📦 tomcat-embed-core-8.5.4.jar - D:\repository
 - ▶ 📦 tomcat-embed-el-8.5.4.jar - D:\repository_s
 - ▶ 📦 tomcat-embed-websocket-8.5.4.jar - D:\rep
 - ▶ 📦 hibernate-validator-5.2.4.Final.jar - D:\repos
 - ▶ 📦 validation-api-1.1.0.Final.jar - D:\repository_
 - ▶ 📦 jboss-logging-3.3.0.Final.jar - D:\repository_

.....

而这些 jar 包正式我们做开发时需要导入的 jar 包。因为这些 jar 包被我们刚才引入的 spring-boot-starter-web 所引用了，所以我们引用 spring-boot-starter-web 后会自动把依赖传递过来。

1.2.2 变更 JDK 版本

我们发现默认情况下工程的 JDK 版本是 1.6,而我们通常用使用 1.7 的版本，所以我们需要在 pom.xml 中添加以下配置

```
<properties>

    <java.version>1.7</java.version>

</properties>
```

添加后更新工程，会发现版本已经变更为 1.7

1.2.3 引导类

只需要创建一个引导类。



件中需要添加如下配置

```
<!-- 使用组件扫描，不用将 controller 在 spring 中配置 -->

<context:component-scan base-package="cn.itcast.demo.controller" />

<!-- 使用注解驱动不用在下边定义映射器和适配器 -->

<mvc:annotation-driven>

    <mvc:message-converters register-defaults="true">

        <bean
class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter">

            <property name="supportedMediaTypes" value="application/json"/>

            <property name="features">

                <array>

                    <value>WriteMapNullValue</value>

                    <value>WriteDateUseDateFormat</value>

                </array>

            </property>

        </bean>

    </mvc:message-converters>

</mvc:annotation-driven>
```

但是我们用 SpringBoot，这一切都省了。我们直接写 Controller 类

```
package cn.itcast.demo.controller;

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

@RestController

public class HelloWorldController {
```



```
@RequestMapping("/info")

public String info(){

    return "HelloWorld";

}

}
```

我们运行启动类来运行程序

在浏览器地址栏输入 <http://localhost:8080/info> 即可看到运行结果

1.2.5 修改 tomcat 启动端口

在 src/main/resources 下创建 application.properties

```
server.port=8088
```

重新运行引导类。地址栏输入

<http://localhost:8088/info>

1.2.6 读取配置文件信息

在 src/main/resources 下的 application.properties 增加配置

```
url=http://www.itcast.cn
```

我要在类中读取这个配置信息，修改 HelloWorldController

```
@Autowired

private Environment env;

@RequestMapping("/info")

public String info(){

    return "HelloWorld~~"+env.getProperty("url");

}
```



```
}
```

1.2.7 热部署

我们在开发中反复修改类、页面等资源，每次修改后都是需要重新启动才生效，这样每次启动都很麻烦，浪费了大量的时间，能不能在我修改代码后不重启就能生效呢？可以，在 pom.xml 中添加如下配置就可以实现这样的功能，我们称之为热部署。

```
<dependency>

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

    <artifactId>spring-boot-devtools</artifactId>

</dependency>
```

赶快试试看吧，是不是很爽。 😄

1.3 Spring Boot 与 ActiveMQ 整合

1.3.1 使用内嵌服务

(1) 在 pom.xml 中引入 ActiveMQ 起步依赖

```
<dependency>

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

    <artifactId>spring-boot-starter-activemq</artifactId>

</dependency>
```

(2) 创建消息生产者

```
/**

 * 消息生产者

 * @author Administrator

 */
```




```
@RestController

public class QueueController {

    @Autowired

    private JmsMessagingTemplate jmsMessagingTemplate;

    @RequestMapping("/send")

    public void send(String text){

        jmsMessagingTemplate.convertAndSend("itcast", text);

    }

}
```

(3) 创建消息消费者

```
@Component

public class Consumer {

    @JmsListener(destination="itcast")

    public void readMessage(String text){

        System.out.println("接收到消息: "+text);

    }

}
```

测试：启动服务后，在浏览器执行

<http://localhost:8088/send.do?text=aaaaa>

即可看到控制台输出消息提示。Spring Boot 内置了 ActiveMQ 的服务，所以我们不用单独启动也可以执行应用程序。

1.3.2 使用外部服务

在 src/main/resources 下的 application.properties 增加配置，指定 ActiveMQ 的地址



```
spring.activemq.broker-url=tcp://192.168.25.135:61616
```

运行后，会在 activeMQ 中看到发送的 queue

Queues

Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
itcast	0	1	1	1	Browse Active Consumers Active Producers	Send To Purge Delete

1.3.3 发送 Map 信息

(1) 修改 QueueController.java

```
@RequestMapping("/sendmap")

public void sendMap(){

    Map map=new HashMap<>();

    map.put("mobile", "13900001111");

    map.put("content", "恭喜获得 10 元代金券");

    jmsMessagingTemplate.convertAndSend("itcast_map",map);

}
```

(2) 修改 Consumer.java

```
@JmsListener(destination="itcast_map")

public void readMap(Map map){

    System.out.println(map);

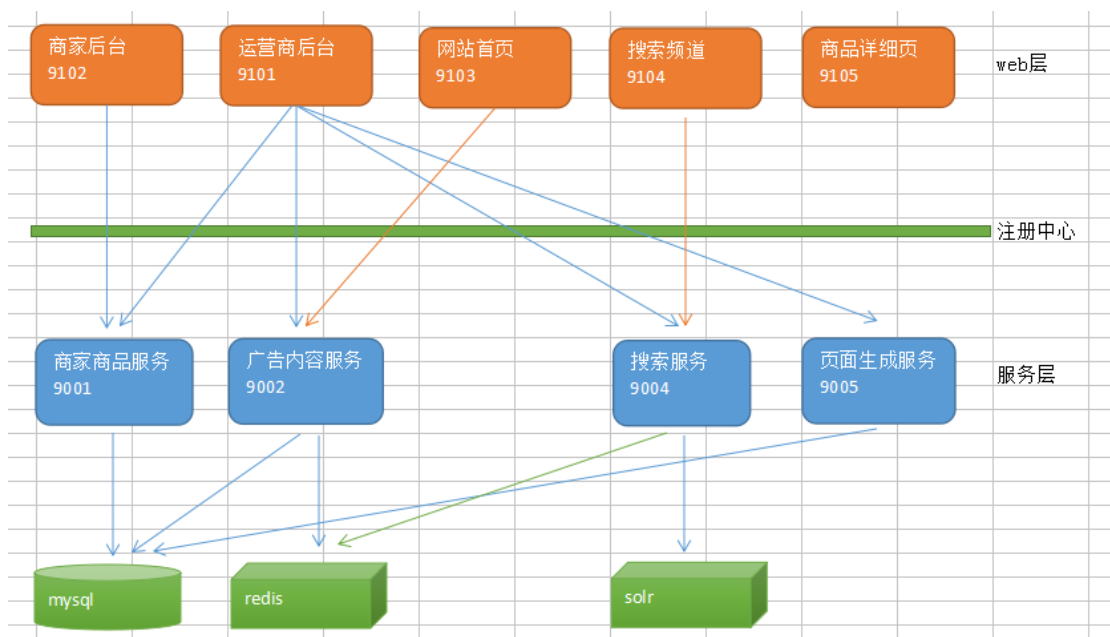
}
```

1. JMS 入门

1.1 消息中间件

1.1.1 品优购系统模块调用关系分析

我们现在讲品优购已经完成的功能模块梳理如下：



我们已经完成了 5 个 web 模块和 4 个服务模块。其中运营商后台的调用关系最多，用到了商家商品服务、广告内容服务、搜索服务和页面生成服务。这种模块之间的依赖也称之为耦合。而耦合越多，之后的维护工作就越困难。那么如果改善系统模块调用关系、减少模块之间的耦合呢？我们接下来就介绍一种解决方案----消息中间件。

1.1.2 什么是消息中间件

消息中间件利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行[分布式系统](#)的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。对于消息中间件，常见的角色大致也就有 **Producer**（生产者）、**Consumer**（消费者）

常见的消息中间件产品：

(1) ActiveMQ

ActiveMQ 是 Apache 出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持 JMS1.1 和 J2EE 1.4 规范的 JMS Provider 实现。我们在本次课程中介绍 ActiveMQ 的使用。

(2) RabbitMQ

AMQP 协议的领导实现，支持多种场景。淘宝的 MySQL 集群内部有使用它进行通讯，OpenStack 开源云平台的通信组件，最先在金融行业得到运用。

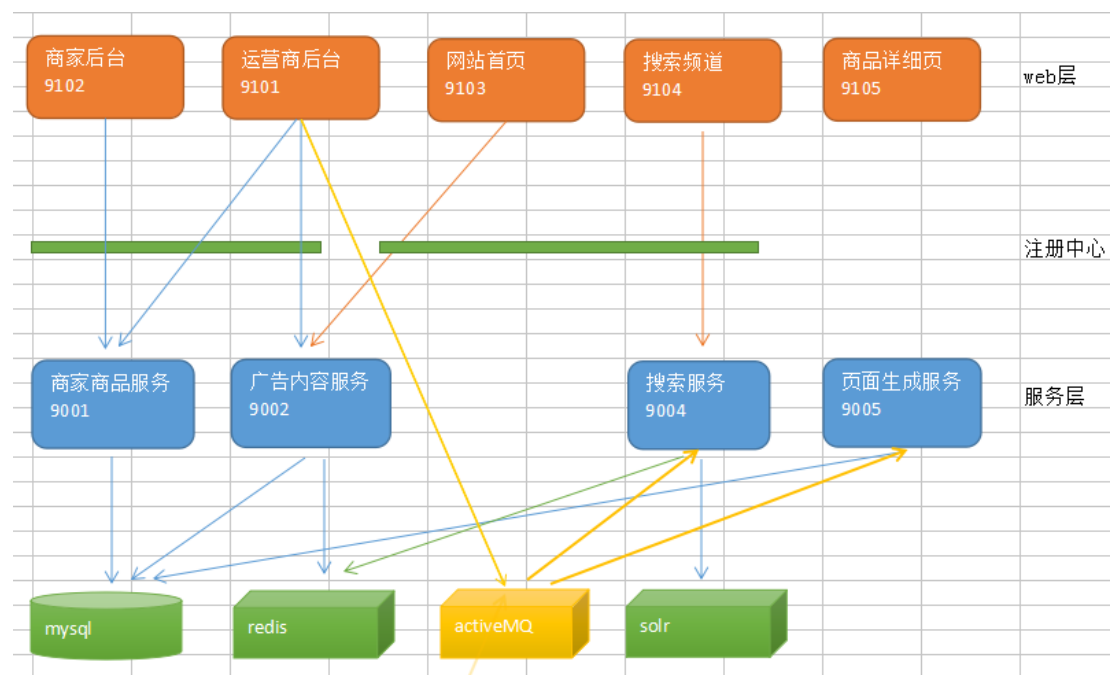
(3) ZeroMQ

史上最快的消息队列系统

(4) Kafka

Apache 下的一个子项目。特点：高吞吐，在一台普通的服务器上既可以达到 10W/s 的吞吐速率；完全的分布式系统。适合处理海量数据。

1.1.3 改造系统模块调用关系



我们通过引入消息中间件 activeMQ,使得运营商系统与搜索服务、页面生成服务解除了耦合。



1.2 JMS 简介

1.2.1 什么是 JMS

JMS (**Java** Messaging Service) 是 Java 平台上有关面向消息中间件的技术规范, 它便于消息系统中的 Java 应用程序进行消息交换, 并且通过提供标准的产生、发送、接收消息的接口简化企业应用的开发。

JMS 本身只定义了一系列的接口规范, 是一种与厂商无关的 API, 用来访问消息收发系统。它类似于 JDBC(**java** Database Connectivity): 这里, JDBC 是可以用来访问许多不同关系**数据库**的 API, 而 JMS 则提供同样与厂商无关的访问方法, 以访问消息收发服务。许多厂商目前都支持 JMS, 包括 IBM 的 MQSeries、BEA 的 Weblogic JMS service 和 Progress 的 SonicMQ, 这只是几个例子。JMS 使您能够通过消息收发服务(有时称为消息中介程序或路由器)从一个 JMS 客户机向另一个 JMS 客户机发送消息。消息是 JMS 中的一种类型对象, 由两部分组成: 报头和消息主体。报头由路由信息以及有关该消息的元数据组成。消息主体则携带着应用程序的数据或有效负载。

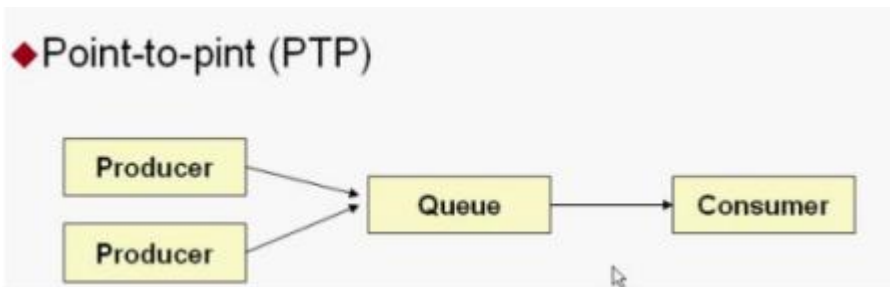
JMS 定义了五种不同的消息正文格式, 以及调用的消息类型, 允许你发送并接收以一些不同形式的数据, 提供现有消息格式的一些级别的兼容性。

- TextMessage--一个字符串对象
- MapMessage--一套名称-值对
- ObjectMessage--一个序列化的 Java 对象
- BytesMessage--一个字节的數據流
- StreamMessage -- Java 原始值的数据流

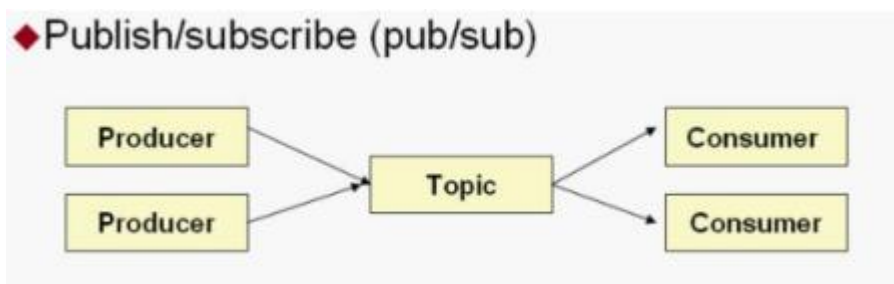
1.2.2 JMS 消息传递类型

对于消息的传递有两种类型:

一种是点对点的, 即一个生产者和一个消费者一一对应;



另一种是发布/ 订阅模式，即一个生产者产生消息并进行发送后，可以由多个消费者进行接收。



1.3 ActiveMQ 下载与安装

1.3.1 下载

官方网站下载: <http://activemq.apache.org/>

1.3.2 安装 (Linux)

(1) 将 apache-activemq-5.12.0-bin.tar.gz 上传至服务器

(2) 解压此文件

```
tar zxvf apache-activemq-5.12.0-bin.tar.gz
```

(3) 为 apache-activemq-5.12.0 目录赋权

```
chmod 777 apache-activemq-5.12.0
```

(4) 进入 apache-activemq-5.12.0\bin 目录

(5) 赋予执行权限

```
chmod 755 activemq
```

知识点小贴士

linux 命令 chmod 755 的意思

chmod 是 **Linux** 下设置文件权限的命令，后面的数字表示不同用户或用户组的权限。

一般是三个数字：

第一个数字表示**文件所有者**的权限

第二个数字表示**与文件所有者同属一个用户组的其他用户**的权限

第三个数字表示**其它用户组**的权限。

权限分为三种：**读（r=4）**，**写（w=2）**，**执行（x=1）**。综合起来还有可读可执行（rx=5=4+1）、可读可写（rw=6=4+2）、可读可写可执行（rwx=7=4+2+1）。

所以，chmod 755 设置用户的权限为：

- 1.文件所有者可读可写可执行 --7
- 2.与文件所有者同属一个用户组的其他用户可读可执行 --5
- 3.其它用户组可读可执行 --5

1.3.3 启动

```
./activemq start
```

出现下列提示表示成功！

```
[root@localhost bin]# ./activemq start
INFO: Loading "/root/apache-activemq-5.12.0/bin/env"
INFO: Using java "/usr/local/src/java/jdk1.7.0_71/bin/java"
INFO: Starting - inspect logfiles specified in logging.properties and log4j.properties to get details
INFO: pidfile created : "/root/apache-activemq-5.12.0/data/activemq.pid" (pid '2461')
```

假设服务器地址为 192.168.25.135 ， 打开浏览器输入地址

<http://192.168.25.135:8161/> 即可进入 ActiveMQ 管理页面



点击进入管理页面



输入用户名和密码 均为 admin



进入主界面



→ 192.168.25.129:8161/admin/

ActiveMQ

Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Welcome!

Welcome to the Apache ActiveMQ Console of **localhost** (ID:localhost.localdomain-41032-1502969174439-0:1)

You can find more information about Apache ActiveMQ on the [Apache ActiveMQ Site](#)

Broker

Name	localhost
Version	5.12.0
ID	ID:localhost.localdomain-41032-1502969174439-0:1
Uptime	58 minutes
Store percent used	0
Memory percent used	0
Temp percent used	0

点对点消息列表：

Home | **Queues** | Topics | Subscribers | Connections | Network | Scheduled | Send

Queue Name

Queues

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
test-queue	1	0	1	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete

列表各列信息含义如下：

Number Of Pending Messages ：等待消费的消息 这个是当前未出队列的数量。

Number Of Consumers ：消费者 这个是消费者端的消费者数量

Messages Enqueued ：进入队列的消息 进入队列的总数量,包括出队列的。

Messages Dequeued ：出了队列的消息 可以理解为是消费这消费掉的数量。



2. JMS 入门小 Demo

2.1 点对点模式

点对点的模式主要建立在一个队列上面，当连接一个队列的时候，发送端不需要知道接收端是否正在接收，可以直接向 ActiveMQ 发送消息，发送的消息，将会先进入队列中，如果有接收端在监听，则会发向接收端，如果没有接收端接收，则会保存在 activemq 服务器，直到接收端接收消息，点对点的消息模式可以有多个发送端，多个接收端，但是一条消息，只会被一个接收端给接收到，哪个接收端先连上 ActiveMQ，则会先接收到，而后来的接收端则接收不到那条消息。

2.1.1 消息生产者

(1) 创建工程 jmsDemo ，引入依赖

```
<dependency>

    <groupId>org.apache.activemq</groupId>

    <artifactId>activemq-client</artifactId>

    <version>5.13.4</version>

</dependency>
```

(2) 创建类 QueueProducer main 方法代码如下：

```
//1.创建连接工厂

ConnectionFactory connectionFactory=new
ActiveMQConnectionFactory("tcp://192.168.25.135:61616");

//2.获取连接

Connection connection = connectionFactory.createConnection();

//3.启动连接

connection.start();

//4.获取 session （参数 1：是否启动事务,参数 2：消息确认模式）
```



```
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

//5.创建队列对象

Queue queue = session.createQueue("test-queue");

//6.创建消息生产者

MessageProducer producer = session.createProducer(queue);

//7.创建消息

TextMessage textMessage = session.createTextMessage("欢迎来到神奇的品优购世界");

//8.发送消息

producer.send(textMessage);

//9.关闭资源

producer.close();

session.close();

connection.close();
```

上述代码中第 4 步创建 session 的两个参数：

第 1 个参数 是否使用事务

第 2 个参数 消息的确认模式

- AUTO_ACKNOWLEDGE = 1 自动确认
- CLIENT_ACKNOWLEDGE = 2 客户端手动确认
- DUPS_OK_ACKNOWLEDGE = 3 自动批量确认
- SESSION_TRANSACTED = 0 事务提交并确认

运行后通过 ActiveMQ 管理界面查询



[Home](#) | [Queues](#) | [Topics](#) | [Subscribers](#) | [Connections](#) | [Network](#) | [Scheduled](#) | [Send](#)

Queue Name

Queues

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
test-queue	1	0	1	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete

2.1.2 消息消费者

创建类 QueueConsumer，main 方法代码如下：

```
//1.创建连接工厂

ConnectionFactory connectionFactory=new
ActiveMQConnectionFactory("tcp://192.168.25.135:61616");

//2.获取连接

Connection connection = connectionFactory.createConnection();

//3.启动连接

connection.start();

//4.获取 session （参数 1：是否启动事务,参数 2：消息确认模式）

Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

//5.创建队列对象

Queue queue = session.createQueue("test-queue");

//6.创建消息消费

MessageConsumer consumer = session.createConsumer(queue);

//7.监听消息

consumer.setMessageListener(new MessageListener() {
```



```
public void onMessage(Message message) {

    TextMessage textMessage=(TextMessage)message;

    try {

        System.out.println("接收到消息: "+textMessage.getText());

    } catch (JMSEException e) {

        // TODO Auto-generated catch block

        e.printStackTrace();

    }

}

});

//8.等待键盘输入

System.in.read();

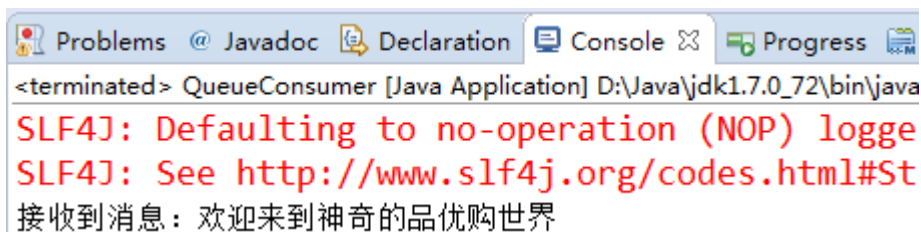
//9.关闭资源

consumer.close();

session.close();

connection.close();
```

执行后看到控制台输出



The screenshot shows an IDE window with tabs for Problems, Javadoc, Declaration, Console, and Progress. The Console tab is active, displaying the following output:

```
<terminated> QueueConsumer [Java Application] D:\Java\jdk1.7.0_72\bin\java
SLF4J: Defaulting to no-operation (NOP) logger
SLF4J: See http://www.slf4j.org/codes.html#St
接收到消息: 欢迎来到神奇的品优购世界
```

2.1.3 运行测试

同时开启 2 个以上的消费者，再次运行生产者，观察每个消费者控制台的输出，会发现只有一个消费者会接收到消息。



2.2 发布/订阅模式

2.2.1 消息生产者

创建类 TopicProducer ， main 方法代码如下：

```
//1.创建连接工厂

ConnectionFactory connectionFactory=new
ActiveMQConnectionFactory("tcp://192.168.25.135:61616");

//2.获取连接

Connection connection = connectionFactory.createConnection();

//3.启动连接

connection.start();

//4.获取 session （参数 1: 是否启动事务,参数 2: 消息确认模式）

Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

//5.创建主题对象

Topic topic = session.createTopic("test-topic");

//6.创建消息生产者

MessageProducer producer = session.createProducer(topic);

//7.创建消息

TextMessage textMessage = session.createTextMessage("欢迎来到神奇的品优购世界");

//8.发送消息

producer.send(textMessage);

//9.关闭资源

producer.close();

session.close();
```



```
connection.close();
```

运行效果如下：

test-topic	0	1	0	Send To Active Subscribers Active Producers Delete
------------	---	---	---	--

2.2.2 消息消费者

创建类 TopicConsumer ,main 方法代码如下：

```
//1.创建连接工厂

ConnectionFactory connectionFactory=new
ActiveMQConnectionFactory("tcp://192.168.25.135:61616");

//2.获取连接

Connection connection = connectionFactory.createConnection();

//3.启动连接

connection.start();

//4.获取 session （参数 1： 是否启动事务,参数 2： 消息确认模式）

Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

//5.创建主题对象

//Queue queue = session.createQueue("test-queue");

Topic topic = session.createTopic("test-topic");

//6.创建消息消费

MessageConsumer consumer = session.createConsumer(topic);

//7.监听消息

consumer.setMessageListener(new MessageListener() {

    public void onMessage(Message message) {
```



```
        TextMessage textMessage=(TextMessage)message;

        try {

            System.out.println("接收到消息: "+textMessage.getText());

        } catch (JMSEException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

    }

});

//8.等待键盘输入

System.in.read();

//9.关闭资源

consumer.close();

session.close();

connection.close();
```

2.2.3 运行测试

同时开启 2 个以上的消费者，再次运行生产者，观察每个消费者控制台的输出，会发现每个消费者会接收到消息。



3.Spring 整合 JMS

3.1 点对点模式

3.1.1 消息生产者

(1) 创建工程 springjms_producer，在 POM 文件中引入 SpringJms 、 activeMQ 以及单元测试相关依赖

(2) 在 src/main/resources 下创建 spring 配置文件 applicationContext-jms-producer.xml

```
<context:component-scan base-package="cn.itcast.demo"></context:component-scan>

<!-- 真正可以产生 Connection 的 ConnectionFactory，由对应的 JMS 服务厂商提供-->

<bean id="targetConnectionFactory"
class="org.apache.activemq.ActiveMQConnectionFactory">

    <property name="brokerURL" value="tcp://192.168.25.135:61616"/>

</bean>

<!-- Spring 用于管理真正的 ConnectionFactory 的 ConnectionFactory -->

<bean id="connectionFactory"
class="org.springframework.jms.connection.SingleConnectionFactory">

    <!-- 目标 ConnectionFactory 对应真实的可以产生 JMS Connection 的 ConnectionFactory -->

    <property name="targetConnectionFactory" ref="targetConnectionFactory"/>

</bean>

<!-- Spring 提供的 JMS 工具类，它可以进行消息发送、接收等 -->

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">

    <!-- 这个 connectionFactory 对应的是我们定义的 Spring 提供的那个 ConnectionFactory
对象 -->

    <property name="connectionFactory" ref="connectionFactory"/>
```



```
</bean>

<!--这个是队列目的地，点对点的 文本信息-->

<bean id="queueTextDestination"
class="org.apache.activemq.command.ActiveMQQueue">

    <constructor-arg value="queue_text"/>

</bean>
```

(3) 在 cn.itcast.demo 包下创建消息生产者类

```
@Component

public class QueueProducer {

    @Autowired

    private JmsTemplate jmsTemplate;

    @Autowired

    private Destination queueTextDestination;

    /**
     * 发送文本消息
     * @param text
     */

    public void sendTextMessage(final String text){

        jmsTemplate.send(queueTextDestination, new MessageCreator() {

            public Message createMessage(Session session) throws JMSException {

                return session.createTextMessage(text);
            }
        });
    }
}
```



```
        }

        });

    }

}
```

(4) 单元测试

在 src/test/java 创建测试类

```
@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations="classpath:applicationContext-jms-producer.xml")

public class TestQueue {

    @Autowired

    private QueueProducer queueProducer;

    @Test

    public void testSend(){

        queueProducer.sendMessage("SpringJms-点对点");

    }

}
```

3.1.2 消息消费者

- (1) 创建工程 springjms_consumer，在 POM 文件中引入依赖（同上一个工程）
- (2) 创建配置文件 applicationContext-jms-consumer-queue.xml

```
<!-- 真正可以产生 Connection 的 ConnectionFactory，由对应的 JMS 服务厂商提供-->

<bean id="targetConnectionFactory"
```



```
class="org.apache.activemq.ActiveMQConnectionFactory">

    <property name="brokerURL" value="tcp://192.168.25.135:61616"/>

</bean>

<!-- Spring 用于管理真正的 ConnectionFactory 的 ConnectionFactory -->

<bean id="connectionFactory"
class="org.springframework.jms.connection.SingleConnectionFactory">

    <!-- 目标 ConnectionFactory 对应真实的可以产生 JMS Connection 的 ConnectionFactory -->

    <property name="targetConnectionFactory" ref="targetConnectionFactory"/>

</bean>

<!--这个是队列目的地，点对点的 文本信息-->

<bean id="queueTextDestination"
class="org.apache.activemq.command.ActiveMQQueue">

    <constructor-arg value="queue_text"/>

</bean>

<!-- 我的监听类 -->

<bean id="myMessageListener" class="cn.itcast.demo.MyMessageListener"></bean>

<!-- 消息监听容器 -->

<bean class="org.springframework.jms.listener.DefaultMessageListenerContainer">

    <property name="connectionFactory" ref="connectionFactory" />

    <property name="destination" ref="queueTextDestination" />

    <property name="messageListener" ref="myMessageListener" />

</bean>
```

(3) 编写监听类

```
public class MyMessageListener implements MessageListener {
```



```
public void onMessage(Message message) {  
  
    TextMessage textMessage=(TextMessage)message;  
  
    try {  
  
        System.out.println("接收到消息: "+textMessage.getText());  
  
    } catch (JMSException e) {  
  
        e.printStackTrace();  
  
    }  
  
}  
  
}
```

(4) 创建测试类

```
@RunWith(SpringJUnit4ClassRunner.class)  
  
@ContextConfiguration(locations="classpath:applicationContext-jms-consumer-queue.xml")  
  
public class TestQueue {  
  
    @Test  
  
    public void testQueue(){  
  
        try {  
  
            System.in.read();  
  
        } catch (IOException e) {  
  
            e.printStackTrace();  
  
        }  
  
    }  
  
}
```



3.2 发布/订阅模式

3.2.1 消息生产者

(1) 在工程 springjms_producer 的 applicationContext-jms-producer.xml 增加配置

```
<!--这个是订阅模式 文本信息-->

<bean id="topicTextDestination"
class="org.apache.activemq.command.ActiveMQTopic">

    <constructor-arg value="topic_text"/>

</bean>
```

(2) 创建生产者类

```
@Component

public class TopicProducer {

    @Autowired

    private JmsTemplate jmsTemplate;

    @Autowired

    private Destination topicTextDestination;

    /**
     * 发送文本消息
     * @param text
     */

    public void sendTextMessage(final String text){

        jmsTemplate.send(topicTextDestination, new MessageCreator() {
```



```
        public Message createMessage(Session session) throws JMSException {

            return session.createTextMessage(text);

        }

    });

}
```

(3) 编写测试类

```
import org.junit.Test;

import org.junit.runner.RunWith;

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

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import cn.itcast.demo.TopicProducer;

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations="classpath:applicationContext-activemq-producer.xml")

public class TestTopic {

    @Autowired

    private TopicProducer topicProducer;

    @Test

    public void sendTextQueue(){

        topicProducer.sendTextMessage();

    }

}
```



3.2.2 消息消费者

(1) 在 activemq-spring-consumer 工程中创建配置文件 applicationContext-jms-consumer-topic.xml

```
<!-- 真正可以产生 Connection 的 ConnectionFactory，由对应的 JMS 服务厂商提供-->

<bean id="targetConnectionFactory"
class="org.apache.activemq.ActiveMQConnectionFactory">

    <property name="brokerURL" value="tcp://192.168.25.135:61616"/>

</bean>

<!-- Spring 用于管理真正的 ConnectionFactory 的 ConnectionFactory -->

<bean id="connectionFactory"
class="org.springframework.jms.connection.SingleConnectionFactory">

    <!-- 目标 ConnectionFactory 对应真实的可以产生 JMS Connection 的 ConnectionFactory -->

    <property name="targetConnectionFactory" ref="targetConnectionFactory"/>

</bean>

<!-- 这个是队列目的地，点对点的 文本信息-->

<bean id="topicTextDestination"
class="org.apache.activemq.command.ActiveMQTopic">

    <constructor-arg value="topic_text"/>

</bean>

<!-- 我的监听类 -->

<bean id="myMessageListener" class="cn.itcast.demo.MyMessageListener"></bean>

<!-- 消息监听容器 -->

<bean class="org.springframework.jms.listener.DefaultMessageListenerContainer">

    <property name="connectionFactory" ref="connectionFactory" />

    <property name="destination" ref="topicTextDestination" />

</bean>
```




```
<property name="messageListener" ref="myMessageListener" />

</bean>
```

（2）编写测试类

```
@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations="classpath:applicationContext-jms-consumer-topic.xml")

public class TestTopic {

    @Test

    public void testTopic(){

        try {

            System.in.read();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}
```

测试：同时运行三个消费者工程，在运行生产者工程，查看三个消费者工程的控制台输出。

4.商品审核-导入 Solr 索引库

4.1 需求分析

运用消息中间件 activeMQ 实现运营商后台与搜索服务的零耦合。运营商执行商品审核后，向 activeMQ 发送消息（SKU 列表），搜索服务从 activeMQ 接收到消息并导入到 solr 索引库。



4.2 消息生产者（运营商后台）

4.2.1 解除耦合

修改 pinyougou-manager-web，移除搜索服务接口依赖：

```
<dependency>

    <groupId>com.pinyougou</groupId>

    <artifactId>pinyougou-search-interface</artifactId>

    <version>0.0.3-SNAPSHOT</version>

</dependency>
```

GoodsController.java 中删除调用搜索服务接口的相关代码

```
//itemSearchService.deleteByGoodsIds(Arrays.asList(ids));

//@Reference

//private ItemSearchService itemSearchService;

//itemSearchService.importList(itemList);
```

4.2.2 准备工作

（1）修改 pinyougou-manager-web 的 pom.xml,引入依赖

```
<dependency>

    <groupId>org.apache.activemq</groupId>

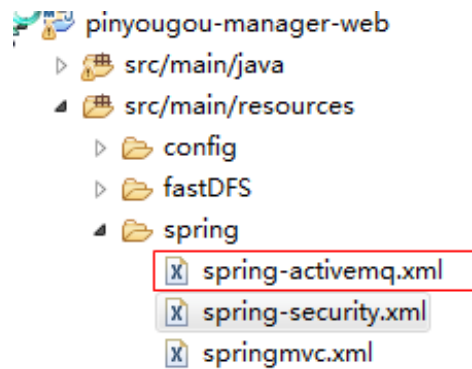
    <artifactId>activemq-client</artifactId>

    <version>5.13.4</version>

</dependency>
```



(2) pinyougou-sellergoods-service 工程添加 applicationContext-jms-producer.xml



改名为 spring-activemq.xml 内容如下：

```
<!-- 真正可以产生 Connection 的 ConnectionFactory，由对应的 JMS 服务厂商提供-->

<bean id="targetConnectionFactory"
class="org.apache.activemq.ActiveMQConnectionFactory">

    <property name="brokerURL" value="tcp://192.168.25.135:61616"/>

</bean>

<!-- Spring 用于管理真正的 ConnectionFactory 的 ConnectionFactory -->

<bean id="connectionFactory"
class="org.springframework.jms.connection.SingleConnectionFactory">

    <!-- 目标 ConnectionFactory 对应真实的可以产生 JMS Connection 的 ConnectionFactory -->

    <property name="targetConnectionFactory" ref="targetConnectionFactory"/>

</bean>

<!-- Spring 提供的 JMS 工具类，它可以进行消息发送、接收等 -->

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">

    <!-- 这个 connectionFactory 对应的是我们定义的 Spring 提供的那个 ConnectionFactory
对象 -->

    <property name="connectionFactory" ref="connectionFactory"/>

</bean>
```



```
<!--这个是队列目的地，点对点-->

<bean id="queueSolrDestination"
class="org.apache.activemq.command.ActiveMQQueue">

    <constructor-arg value="pinyougou_queue_solr"/>

</bean>
```

(3) 修改 web.xml

```
<context-param>

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

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

</context-param>
```

4.2.3 代码实现

修改 pinyougou-manager-web 工程的 GoodsController.java

```
@Autowired

private Destination queueSolrDestination;//用于发送 solr 导入的消息


@Autowired

private JmsTemplate jmsTemplate;


@RequestMapping("/updateStatus")

public Result updateStatus(Long[] ids,String status){

    try {

        goodsService.updateStatus(ids, status);

        //按照 SPU ID 查询 SKU 列表(状态为1)
```



```
        if(status.equals("1")){//审核通过

            List< TbItem> itemList =
goodsService.findItemListByGoodsIdandStatus(ids, status);

            //调用搜索接口实现数据批量导入

            if(itemList.size()>0){

                final String jsonString = JSON.toJsonString(itemList);

                jmsTemplate.send(queueSolrDestination, new MessageCreator() {

                    @Override

                    public Message createMessage(Session session) throws JMSException

                {

                    return session.createTextMessage(jsonString);

                }

            });

            }else{

                System.out.println("没有明细数据");

            }

        }

        return new Result(true, "修改状态成功");

    } catch (Exception e) {

        e.printStackTrace();

        return new Result(false, "修改状态失败");

    }

}
```



4.3 消息消费者（搜索服务）

4.3.1 准备工作

(1) 修改 pinyougou-search-service ，在 pom.xml 中添加 activemq 依赖

```
<dependency>

    <groupId>org.apache.activemq</groupId>

    <artifactId>activemq-client</artifactId>

    <version>5.13.4</version>

</dependency>
```

(2) 添加 spring 配置文件 applicationContext-jms-consumer.xml

```
<!-- 真正可以产生 Connection 的 ConnectionFactory，由对应的 JMS 服务厂商提供-->

<bean id="targetConnectionFactory"
class="org.apache.activemq.ActiveMQConnectionFactory">

    <property name="brokerURL" value="tcp://192.168.25.135:61616"/>

</bean>

<!-- Spring 用于管理真正的 ConnectionFactory 的 ConnectionFactory -->

<bean id="connectionFactory"
class="org.springframework.jms.connection.SingleConnectionFactory">

    <property name="targetConnectionFactory" ref="targetConnectionFactory"/>

</bean>

<!-- 这个是队列目的地，点对点的 文本信息-->

<bean id="queueSolrDestination"
class="org.apache.activemq.command.ActiveMQQueue">

    <constructor-arg value="pinyougou_queue_solr"/>

</bean>
```



```
<!-- 消息监听容器 -->

<bean class="org.springframework.jms.listener.DefaultMessageListenerContainer">

    <property name="connectionFactory" ref="connectionFactory" />

    <property name="destination" ref="queueSolrDestination" />

    <property name="messageListener" ref="itemSearchListener" />

</bean>
```

4.3.2 代码实现

在 pinyougou-search-service 的 com.pinyougou.search.service.impl 新增监听类

```
@Component

public class ItemSearchListener implements MessageListener{

    @Autowired

    private ItemSearchService itemSearchService;

    @Override

    public void onMessage(Message message) {

        System.out.println("监听接收到消息...");

        try {

            TextMessage textMessage=(TextMessage)message;

            String text = textMessage.getText();

            List<TbItem> list = JSON.parseArray(text,TbItem.class);

            for(TbItem item:list){

                System.out.println(item.getId()+" "+item.getTitle());

                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

    }

    itemSearchService.importList(list); //导入

    System.out.println("成功导入到索引库");

} catch (Exception e) {

    e.printStackTrace();

}

}

}
```

5.商品删除-移除 Solr 索引库记录

5.1 需求分析

通过消息中间件实现在商品删除时也同时移除索引库记录的功能。

5.2 消息生产者（运营商后台）

5.2.1 配置文件

修改 pinyougou-manager-web 工程的 spring-activemq.xml，添加 bean 配置

```
<!--这个是队列目的地，点对点-->

<bean id="queueSolrDeleteDestination"
class="org.apache.activemq.command.ActiveMQQueue">

    <constructor-arg value="pinyougou_queue_solr_delete"/>

</bean>
```




5.2.2 代码实现

修改 GoodsController.java

```
@Autowired

private Destination queueSolrDeleteDestination;//用户在索引库中删除记录

/**
 * 批量删除
 * @param ids
 * @return
 */

@RequestMapping("/delete")

public Result delete(final Long [] ids){

    try {

        goodsService.delete(ids);

        jmsTemplate.send(queueSolrDeleteDestination, new MessageCreator() {

            @Override

            public Message createMessage(Session session) throws JMSException {

                return session.createObjectMessage(ids);

            }

        });

        return new Result(true, "删除成功");

    } catch (Exception e) {

        e.printStackTrace();

    }

}
```



```
        return new Result(false, "删除失败");  
  
    }  
  
}
```

5.3 消息消费者（搜索服务）

5.3.1 配置文件

修改 pinyougou-search-service 的 applicationContext-activemq-consumer.xml

```
<!-- 这个是队列目的地，点对点的 文本信息 （删除索引库中记录） -->  
  
<bean id="queueSolrDeleteDestination"  
class="org.apache.activemq.command.ActiveMQQueue">  
  
    <constructor-arg value="pinyougou_queue_solr_delete"/>  
  
</bean>  
  
<!-- 消息监听容器(删除索引库中记录) -->  
  
<bean class="org.springframework.jms.listener.DefaultMessageListenerContainer">  
  
    <property name="connectionFactory" ref="connectionFactory" />  
  
    <property name="destination" ref="queueSolrDeleteDestination" />  
  
    <property name="messageListener" ref="itemDeleteListener" />  
  
</bean>
```

5.3.2 代码实现

com.pinyougou.search.service.impl 包下创建监听类 ItemDeleteListener

```
/**  
  
 * 监听：用于删除索引库中记录  
  
 * @author Administrator
```



```
*  
  
*/  
  
@Component  
  
public class ItemDeleteListener implements MessageListener{  
  
    @Autowired  
  
    private ItemSearchService itemSearchService;  
  
    @Override  
  
    public void onMessage(Message message) {  
  
        try {  
  
            ObjectMessage objectMessage= (ObjectMessage)message;  
  
            Long[] goodsIds = (Long[]) objectMessage.getObject();  
  
            System.out.println("ItemDeleteListener 监听接收到消息..." + goodsIds);  
  
            itemSearchService.deleteByGoodsIds(Arrays.asList(goodsIds));  
  
            System.out.println("成功删除索引库中的记录");  
  
        } catch (Exception e) {  
  
            e.printStackTrace();  
  
        }  
  
    }  
  
}
```