第8章 RocketMQ消息中间件

广东财经大学信息学院
罗 东 俊 博士
ZSUJONE@126.COM

目的和要求

- 了解为什么要使用消息中间件。
- 熟悉RocketMQ消息中间件的基本概念和工作原理。
- 掌握Spring Boot与RocketMQ的整合实现与应用。

主要内容

8.3 RocketMQ整合案例

- 8.3.1 消息生产和消费案例
- 8.3.2 电商秒杀应用案例

8.3.1 消息生产和消费案例

- 本案例基于Spring Boot整合RocketMQ消息中间件，实现单向消息、同步消息、异步消息、顺序消息、批量消息、延迟消息、事务消息的生产和消费。

相关注解

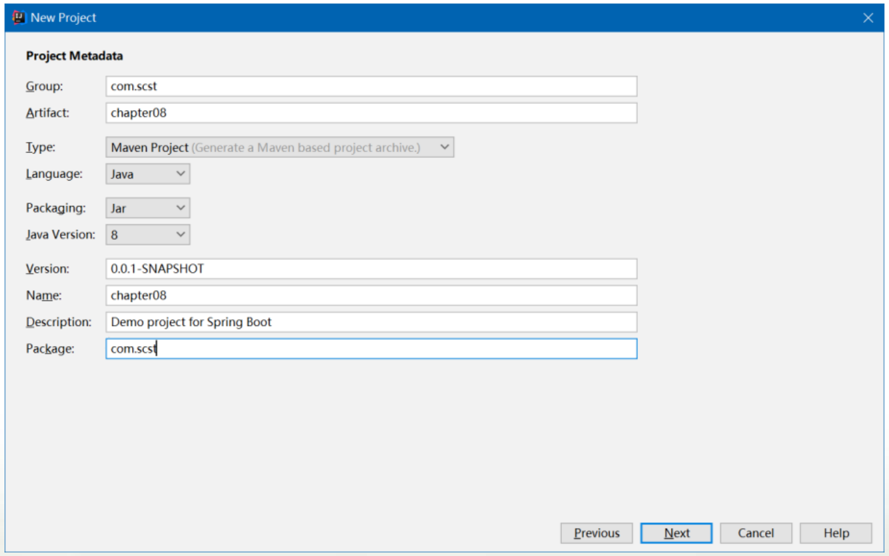
- @RocketMQTransactionListener(rocketMQTemplateBeanName)：声明一个本地事务监听器
  - ✧rocketMQTemplateBeanName为消息生产者发送消息的"RocketMQTemplate"Bean实例名
- @RocketMQMessageListener(nameServer, topic, consumerGroup, messageModel)：声明一个消息消费者
  - ✧nameServer为RocketMQ NameServer；
  - ✧topic为消费的消息主题；
  - ✧consumerGroup为消费者所在组名；
  - ✧messageModel为消息模式，默认为CLUSTERING；
  - ✧consumeMode为消费模式，默认为CONCURRENTLY

整合步骤

- ■1）创建Spring Boot父项目
- ■2）创建公共模块
- ■3）创建消息生产者模块
- ■4）创建消息消费者模块
- ■5）效果测试

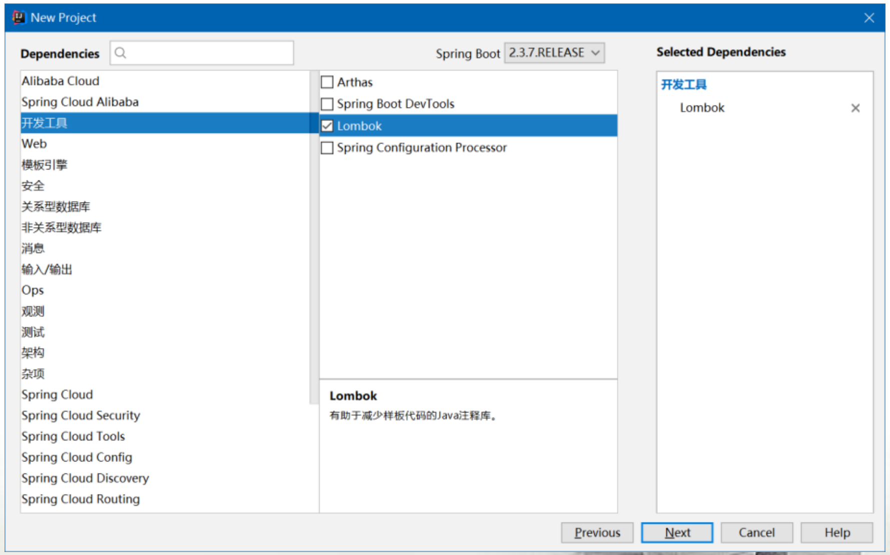
1）创建Spring Boot父项目

- 使用Spring Initializr方式创建一个Spring Boot项目chapter08，在Dependencies依赖选择中选择开发工具模块中的Lombok依赖。



Project Metadata

Group:	com.scst
Artifact:	chapter08
Type:	Maven Project (Generate a Maven based project archive.)
Language:	Java
Packaging:	Jar
Java Version:	8
Version:	0.0.1-SNAPSHOT
Name:	chapter08
Description:	Demo project for Spring Boot
Package:	com.scst



Spring Boot 2.3.7.RELEASE

☑ Lombok

Lombok
有助于减少样板代码的Java注释库。

Selected Dependencies
开发工具
Lombok

父项目chapter08

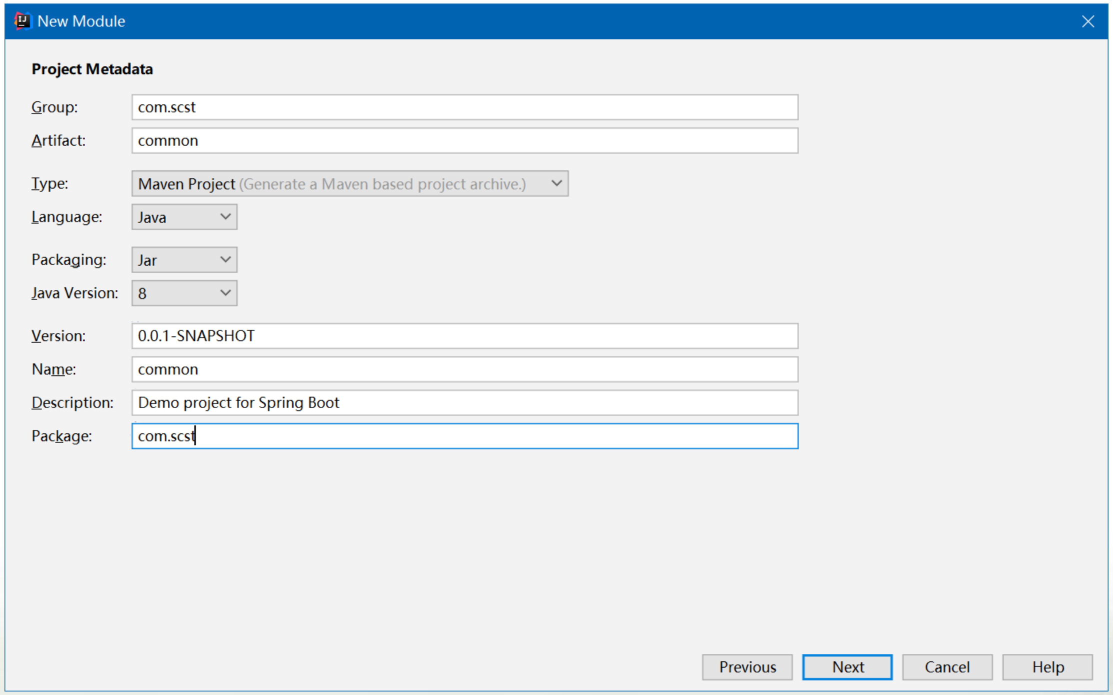
```
<groupId>com.scst</groupId>
<artifactId>chapter08</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>pom</packaging>
<name>chapter08</name>
<description>Demo project for Spring Boot</description>
```

2）创建公共模块

- 该公共模块用于组织实体类和工具类。
- 搭建步骤：
  - ✧①创建Spring Boot模块
  - ✧②创建消息实体类

①创建Spring Boot模块

- 在chapter08项目中，使用Spring Initializr方式创建一个Spring Boot模块common。



Group:	com.scst
Artifact:	common
Type:	Maven Project (Generate a Maven based project archive.)
Language:	Java
Packaging:	Jar
Java Version:	8
Version:	0.0.1-SNAPSHOT
Name:	common
Description:	Demo project for Spring Boot
Package:	com.scst

建立父项目依赖

■ 在common模块的pom.xml文件中引入对父项目chapter08依赖，示例代码如下。

```
<parent>
    <artifactId>chapter08</artifactId>
    <groupId>com.scst</groupId>
    <version>0.0.1-SNAPSHOT</version>
</parent>
```

②创建消息实体类

- 在common模块中新建一个com.scst.domain包，并在包中创建一个实体类Order，作为消息的消息体类型。

Order类

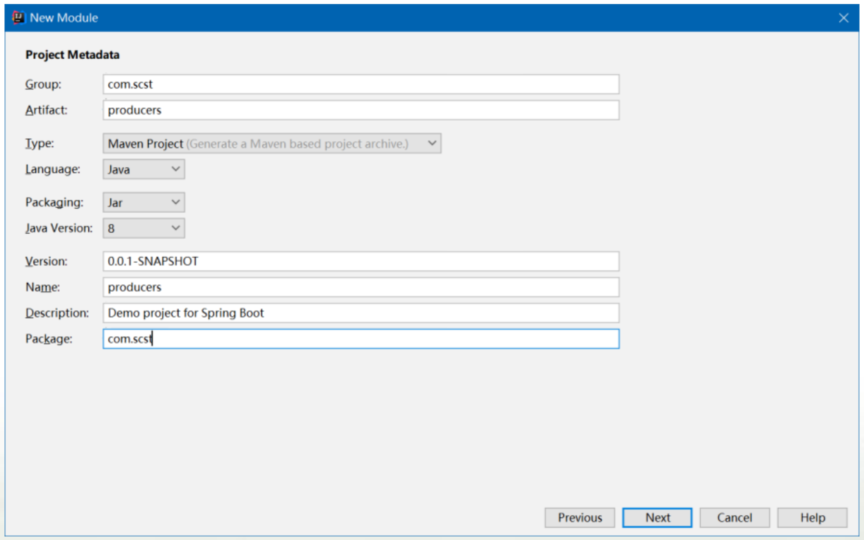
```
package com.scst.domain;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Date;
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Order {
    private String orderId;            //唯一订单号（由雪花算法生成），据此实现消费幂等性
    private String userId;             //用户ID
    private String productId;          //商品ID
    private Integer purchaseNum;       //购买数量
    private boolean status;            //是否支付
    private Date createTime;           //下单时间
}
```

3）创建消息生产者模块

- 搭建步骤：
  - ✧①创建Spring Boot模块
  - ✧②配置消息中间件连接
  - ✧③创建消息生产服务类
  - ✧④创建Web控制器类

①创建Spring Boot模块

- 在chapter08项目中，使用Spring Initializr方式创建一个Spring Boot模块producers，在Dependencies依赖选择中选择Web模块中的Spring Web依赖。

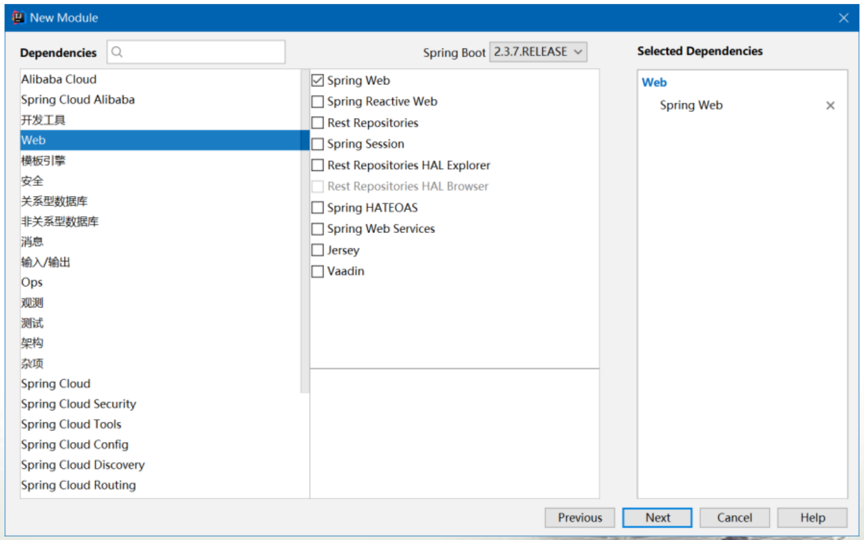


New Module

Project Metadata

Group:	com.scst
Artifact:	producers
Type:	Maven Project (Generate a Maven based project archive.)
Language:	Java
Packaging:	Jar
Java Version:	8
Version:	0.0.1-SNAPSHOT
Name:	producers
Description:	Demo project for Spring Boot
Package:	com.scst

Previous | Next | Cancel | Help



New Module

Dependencies — Spring Boot 2.3.7.RELEASE

Alibaba Cloud
Spring Cloud Alibaba
开发工具
Web
模板引擎
安全
关系型数据库
非关系型数据库
消息
输入/输出
Ops
观测
测试
架构
杂项
Spring Cloud
Spring Cloud Security
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing

- [✓] Spring Web
- [ ] Spring Reactive Web
- [ ] Rest Repositories
- [ ] Spring Session
- [ ] Rest Repositories HAL Explorer
- [ ] Rest Repositories HAL Browser
- [ ] Spring HATEOAS
- [ ] Spring Web Services
- [ ] Jersey
- [ ] Vaadin

Selected Dependencies

Web
Spring Web ×

Previous | Next | Cancel | Help

建立父项目和公共模块依赖

- 在producers模块的pom.xml文件中引入对父项目chapter08和公共模块common的依赖，示例代码如下。

```
<parent>
    <artifactId>chapter08</artifactId>
    <groupId>com.scst</groupId>
    <version>0.0.1-SNAPSHOT</version>
</parent>
<dependencies>
    <dependency>
        <groupId>com.scst</groupId>
        <artifactId>common</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
</dependencies>
```

引入RocketMQ依赖启动器

```
<!-- RocketMQ依赖启动器 -->
 <dependency>
     <groupId>org.apache.rocketmq</groupId>
     <artifactId>rocketmq-spring-boot-starter</artifactId>
     <version>2.1.1</version>
 </dependency>
```

②配置消息中间件连接

- 在producers模块的全局配置文件application.properties中添加RocketMQ消息中间件的连接配置，示例代码如下。

```
#RocketMQ NameServer地址，多个nameserver地址采用";"分隔
rocketmq.name-server=127.0.0.1:9876
#rocketmq.name-server=127.0.0.1:9876;192.168.31.173:9876
#消息生产者组名
rocketmq.producer.group=producer-group
```

③创建消息生产服务类

- 在producers模块中新建一个com.scst.service包，并在包中新建一个业务类ProduceService，实现单向消息、同步消息、异步消息、顺序消息、批量消息、延迟消息、事务消息的生产和发送。

ProduceService类

```
package com.scst.service;
import com.scst.domain.Order;
import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.client.producer.SendCallback;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.client.producer.TransactionSendResult;
import org.apache.rocketmq.spring.annotation.RocketMQTransactionListener;
import org.apache.rocketmq.spring.core.RocketMQLocalTransactionListener;
import org.apache.rocketmq.spring.core.RocketMQLocalTransactionState;
import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.Message;
import org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.concurrent.ConcurrentHashMap;
```

ProduceService类

```
@Slf4j
@Service
//消息生产服务
public class ProduceService {
    @Autowired
    private RocketMQTemplate rocketMQTemplate;
    private String destination;
    private Object payload;
    //1.发送单向消息
    public void sendOneWayMessage(String topic, String tag) {
        this.destination = topic;
        if (tag != null) {
            this.destination = topic + ":" + tag;
        }
        this.payload = new Order("O01", "U01", "P01", 2, false, new Date());
        rocketMQTemplate.convertAndSend(this.destination, this.payload);
    }
```

ProduceService类

```
//2.发送同步消息
public void sendSyncMessage(String topic, String tag) {
    this.destination = topic;
    if (tag != null) {
        this.destination = topic + ":" + tag;
    }
    this.payload = new Order("O01", "U01", "P01", 2, false, new Date());
    SendResult sendResult = rocketMQTemplate.syncSend(this.destination, this.payload);
    log.info("【同步发送结果】{}", sendResult);
}
```

ProduceService类

```
//3.发送异步消息
public void sendAsyncMessage(String topic, String tag) {
    this.destination = topic;
    if (tag != null) {
        this.destination = topic + ":" + tag;
    }
    this.payload = new Order("O01", "U01", "P01", 2, false, new Date());
    rocketMQTemplate.asyncSend(this.destination, this.payload, new SendCallback() {
        @Override
        public void onSuccess(SendResult sendResult) {
            log.info("【异步发送结果】{}", sendResult);
        }
        @Override
        public void onException(Throwable throwable) {
            log.error("【异步发送异常】{}", throwable.getMessage());
        }
    });
}
```

ProduceService类

```
//4.发送顺序消息
public void sendOrderedMessages(String topic, String tag) {
    this.destination = topic;
    if (tag != null) {
        this.destination = topic + ":" + tag;
    }
    for (int i = 0; i < 10; i++) {
        //hashkey用于选择消息队列，只有在相同队列的消息能保持顺序
        rocketMQTemplate.syncSendOrderly(this.destination, new Order("O0" + i, "U01", "P01", 2, false, new
Date()), "hashKey");
    }
}
```

ProduceService类

```
//5.发送批量消息
public void sendBatchMessages(String topic, String tag) {
    this.destination = topic;
    if (tag != null) {
        this.destination = topic + ":" + tag;
    }
    List<Message> messages = new ArrayList<>();
    for (int i = 0; i < 10; i++) {
        this.payload = new Order("O0" + i, "U01", "P01", 2, false, new Date());
        Message<Object> message = MessageBuilder.withPayload(this.payload).build();
        messages.add(message);
    }
    SendResult sendResult = rocketMQTemplate.syncSend(this.destination, messages, 1000);
    log.info("【批量发送结果】{}", sendResult);
}
```

ProduceService类

```
//6.发送延迟消息
public void sendDelayMessage(String topic, String tag) {
    this.destination = topic;
    if (tag != null) {
        this.destination = topic + ":" + tag;
    }
    this.payload = new Order("O01", "U01", "P01", 2, false, new Date());
    Message<Object> message = MessageBuilder.withPayload(this.payload).build();
    //设置延迟时间为10s(1s/5s/10s/30s/1m/2m/3m/4m/5m/6m/7m/8m/9m/10m/20m/30m/1h/2h)
    SendResult sendResult = rocketMQTemplate.syncSend(this.destination, message, 100000, 3);
    log.info("【延迟发送结果】{}", sendResult);
}
```

ProduceService类

```
//7.发送事务消息
public void sendTransactionMessage(String topic, String tag) {
    this.destination = topic;
    if (tag != null) {
        this.destination = topic + ":" + tag;
    }
    this.payload = new Order("O01", "U01", "P01", 2, false, new Date());
    Message<Object> message = MessageBuilder.withPayload(this.payload).build();
    log.info("【发送半消息】{}", message.getPayload());
    TransactionSendResult result = rocketMQTemplate.sendMessageInTransaction(this.destination, message,
this.payload);
    log.info("【本地事务状态】{}", result.getLocalTransactionState());
}
```

ProduceService类

```
//声明本地事务监听器
@RocketMQTransactionListener(rocketMQTemplateBeanName = "rocketMQTemplate")
class ProducerLocalTransactionListener implements RocketMQLocalTransactionListener {
   private ConcurrentHashMap<String, Object> localTrans = new ConcurrentHashMap<>();
   //半消息投递成功后执行的逻辑
   @Override
   public RocketMQLocalTransactionState executeLocalTransaction(Message message, Object o) {
      try {
         log.info("【收到半消息响应ACK，执行本地事务：】");
         log.info("Message:{}", message);
         log.info("Object:{}", o);
         localTrans.put(message.getHeaders().getId() + "", message.getPayload());
         return RocketMQLocalTransactionState.UNKNOWN;
         //return RocketMQLocalTransactionState.COMMIT;
      } catch (Exception e) {
         e.printStackTrace();
         log.error("【执行本地事务异常】 Exception:{}", e.getMessage());
         return RocketMQLocalTransactionState.ROLLBACK;
      }
   }
```

ProduceService类

```
        //回查本地事务执行状态
        @Override
        public RocketMQLocalTransactionState checkLocalTransaction(Message message) {
            log.info("【检查本地事务状态】");
            return RocketMQLocalTransactionState.COMMIT;
        }
    }
}
```

④创建Web控制器类

- 在producers模块中新建一个com.scst.controller包，并在包中新建一个Web控制类ProduceController，并在该类中实现单向消息、同步消息、异步消息、顺序消息、批量消息、延迟消息、事务消息的发送请求。

ProduceController类

```
package com.scst.controller;
import com.scst.service.ProduceService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class ProduceController {
    @Autowired
    private ProduceService produceService;
    private String topic = "msg-topic";
    private String tag = "TagA";

    @GetMapping("/transactionMessage")
    public void testTransactionMessage() {
        produceService.sendTransactionMessage(topic, tag);
    }
```

ProduceController类

```
@GetMapping("/onewayMessage")
public void testOnewayMessage() {
    produceService.sendOneWayMessage(topic, tag);
}

@GetMapping("/syncMessage")
public void testSyncMessage() {
    produceService.sendSyncMessage(topic, tag);
}

@GetMapping("/asyncMessage")
public void testAsyncMessage() {
    produceService.sendAsyncMessage(topic, tag);
}
```

ProduceController类

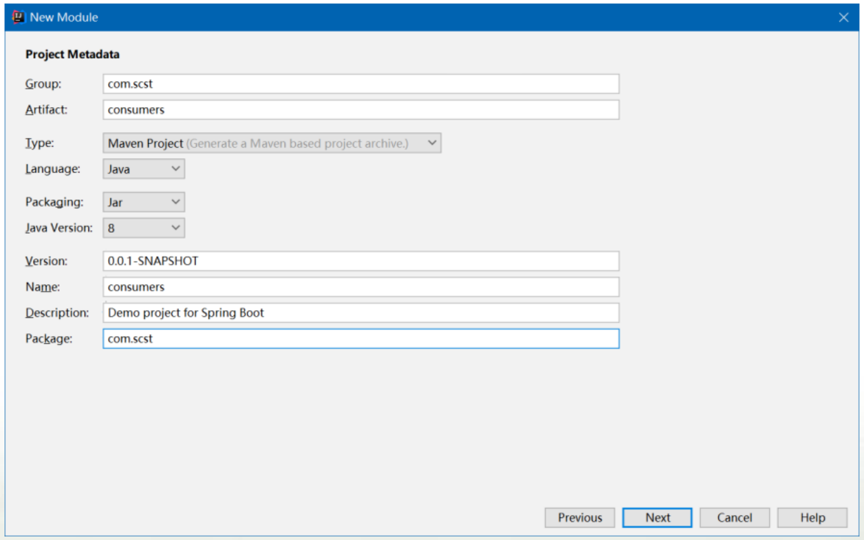
```
    @GetMapping("/delayMessage")
    public void testDelayMessage() {
        produceService.sendDelayMessage(topic, tag);
    }

    @GetMapping("/orderedMessages")
    public void testOrderedMessages() {
        produceService.sendOrderedMessages(topic, tag);
    }

    @GetMapping("/batchMessages")
    public void testBatchMessages() {
        produceService.sendBatchMessages(topic, tag);
    }
}
```

4）创建消息消费者模块

- 搭建步骤：
  - ✧①创建Spring Boot模块
  - ✧②配置消息中间件连接
  - ✧③创建消息消费服务类

①创建Spring Boot模块

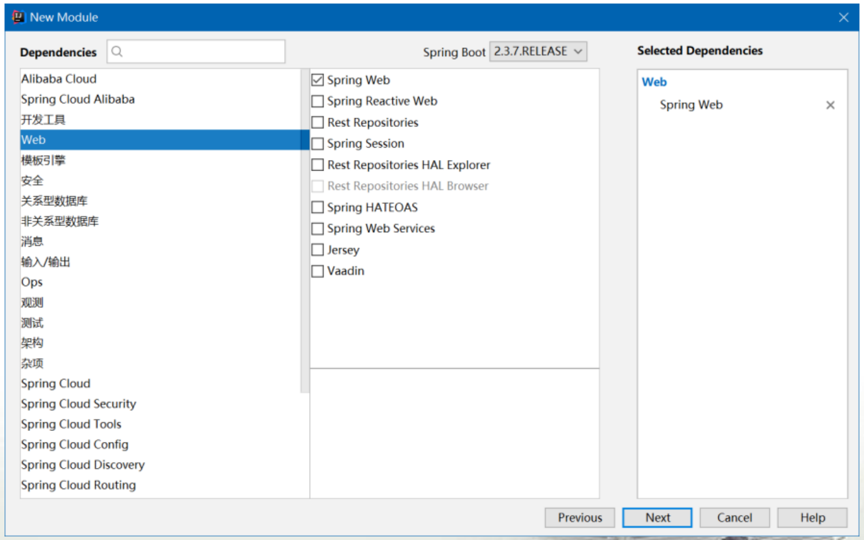
- 在chapter08项目中，使用Spring Initializr方式创建一个Spring Boot模块consumers，在Dependencies依赖选择中选择Web模块中的Spring Web依赖。



New Module

Project Metadata

Group:	com.scst
Artifact:	consumers
Type:	Maven Project (Generate a Maven based project archive.)
Language:	Java
Packaging:	Jar
Java Version:	8
Version:	0.0.1-SNAPSHOT
Name:	consumers
Description:	Demo project for Spring Boot
Package:	com.scst



New Module

Dependencies Spring Boot 2.3.7.RELEASE **Selected Dependencies**

Alibaba Cloud
Spring Cloud Alibaba
开发工具
Web
模板引擎
安全
关系型数据库
非关系型数据库
消息
输入/输出
Ops
观测
测试
架构
杂项
Spring Cloud
Spring Cloud Security
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing

- [x] Spring Web
- [ ] Spring Reactive Web
- [ ] Rest Repositories
- [ ] Spring Session
- [ ] Rest Repositories HAL Explorer
- [ ] Rest Repositories HAL Browser
- [ ] Spring HATEOAS
- [ ] Spring Web Services
- [ ] Jersey
- [ ] Vaadin

Web
Spring Web ×

建立父项目和公共模块依赖

- 在consumers模块的pom.xml文件中引入对父项目chapter08和公共模块common的依赖，示例代码如下。

```
<parent>
    <artifactId>chapter08</artifactId>
    <groupId>com.scst</groupId>
    <version>0.0.1-SNAPSHOT</version>
</parent>
<dependencies>
    <dependency>
        <groupId>com.scst</groupId>
        <artifactId>common</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
</dependencies>
```

②配置消息中间件连接

- 在consumers模块的全局配置文件application.properties中添加RocketMQ消息中间件的连接配置，示例代码如下。

```
#RocketMQ NameServer地址，多个nameserver地址采用";"分隔
rocketmq.name-server=127.0.0.1:9876
#rocketmq.name-server=127.0.0.1:9876;192.168.31.173:9876
#消息消费者组名
rocketmq.consumer.group0=consumer-group0
rocketmq.consumer.group1=consumer-group1
rocketmq.consumer.group2=consumer-group2
rocketmq.consumer.group=order-group
```

③创建消息消费服务类

- 在consumers模块中新建一个com.scst.service包，并在包中新建一个业务类ConsumeService，实现单向消息、同步消息、异步消息、顺序消息、批量消息、延迟消息、事务消息的监听和消费。

ConsumeService类

```
package com.scst.service;
import com.scst.domain.Order;
import org.apache.rocketmq.spring.annotation.ConsumeMode;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;
import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.spring.annotation.MessageModel;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
//消息消费服务
@Slf4j
@Service
public class ConsumeService {
```

ConsumeService类

```
   @Component
   //topic需要和生产者的topic一致；consumerGroup属性必须指定，内容可随意；messageModel默认为
CLUSTERING
   @RocketMQMessageListener(nameServer = "${rocketmq.name-server}", topic = "msg-topic", consumerGroup
= "${rocketmq.consumer.group0}", messageModel = MessageModel.BROADCASTING)
   class ConsumerInGroup0 implements RocketMQListener<Object> {
      @Override
      public void onMessage(Object o) {
         //消息体（Payload）o为JSON对象(字符串)
         log.info("C0开始消费消息:{}", o);
      }
   }
```

ConsumeService类

```
@Component
//topic需要和生产者的topic一致；consumerGroup属性必须指定，内容可随意；consumeMode默认为CONCURRENTLY
@RocketMQMessageListener(nameServer = "${rocketmq.name-server}", topic = "msg-topic", consumerGroup = "${rocketmq.consumer.group1}", consumeMode = ConsumeMode.ORDERLY)
class ConsumerInGroup1 implements RocketMQListener<Order> {
    @Override
    public void onMessage(Order o) {
        //消息体（Payload）o为Java对象
        log.info("C1开始消费消息:{}", o);
    }
}
```

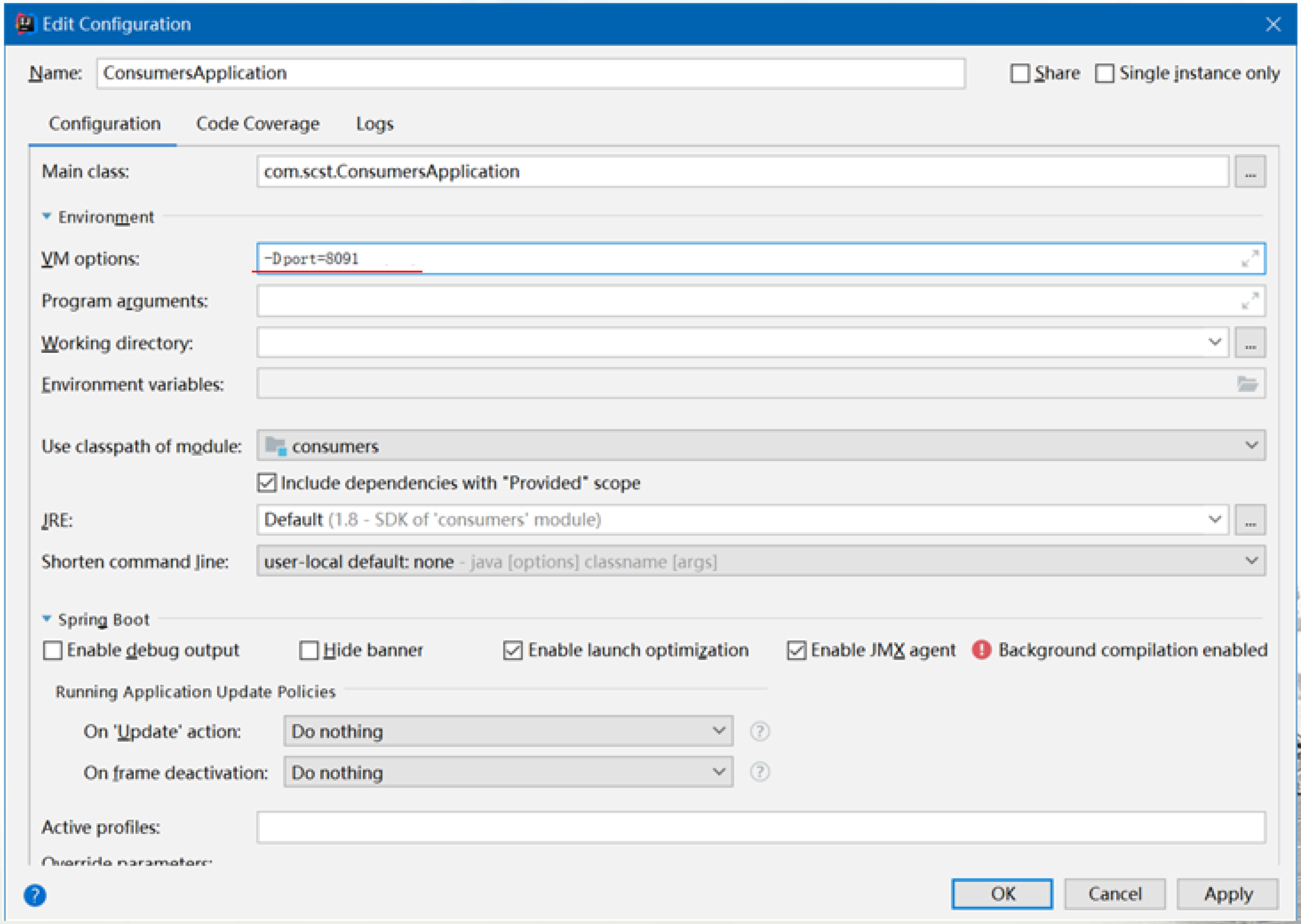
ConsumeService类

```
@Component
//topic需要和生产者的topic一致；consumerGroup属性必须指定，内容可随意；consumeMode默认为CONCURRENTLY；messageModel默认为CLUSTERING
@RocketMQMessageListener(nameServer = "${rocketmq.name-server}", topic = "msg-topic", consumerGroup = "${rocketmq.consumer.group2}")
class ConsumerInGroup2 implements RocketMQListener<Order> {
    @Override
    public void onMessage(Order o) {
        //消息体（Payload）o为Java对象
        log.info("C2开始消费消息:{}", o);
    }
}
}
```

5）效果测试

- ■ 测试步骤：
  - ✧①启动Name Server
  - ✧②启动Broker Server
  - ✧③以8080端口启动ProducersApplication类
  - ✧④先后以8090和8091端口启动ConsumersApplication类
  - ✧⑤分别访问http://localhost:8080/transactionMessage和http://localhost:8080/batchMessages测试事务消息和批量消息的生产和消费效果。

更改服务端口


Edit Configuration

Name: ConsumersApplication

☐ Share ☐ Single instance only

Configuration | Code Coverage | Logs

Main class: com.scst.ConsumersApplication

Environment

VM options: -Dport=8091

Program arguments:

Working directory:

Environment variables:

Use classpath of module: consumers

☑ Include dependencies with "Provided" scope

JRE: Default (1.8 - SDK of 'consumers' module)

Shorten command line: user-local default: none - java [options] classname [args]

Spring Boot

☐ Enable debug output ☐ Hide banner ☑ Enable launch optimization ☑ Enable JMX agent ● Background compilation enabled

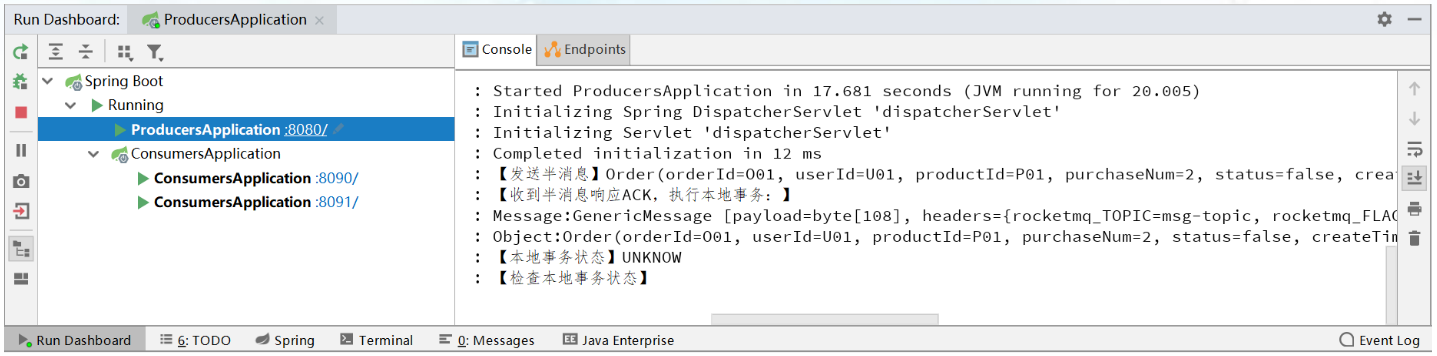
Running Application Update Policies

On 'Update' action: Do nothing

On frame deactivation: Do nothing

Active profiles:

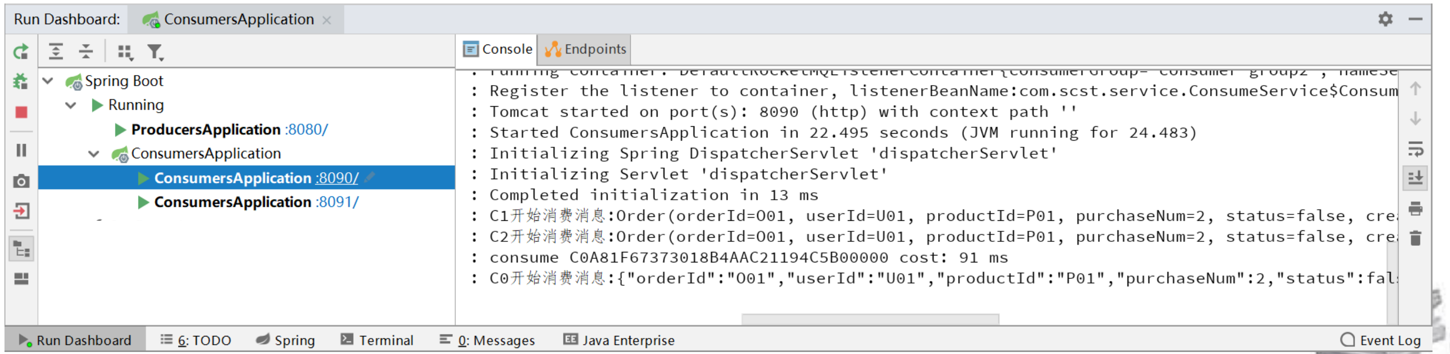
事务消息测试效果



Run Dashboard: ProducersApplication

- Spring Boot
  - Running
    - ProducersApplication :8080/
    - ConsumersApplication
      - ConsumersApplication :8090/
      - ConsumersApplication :8091/

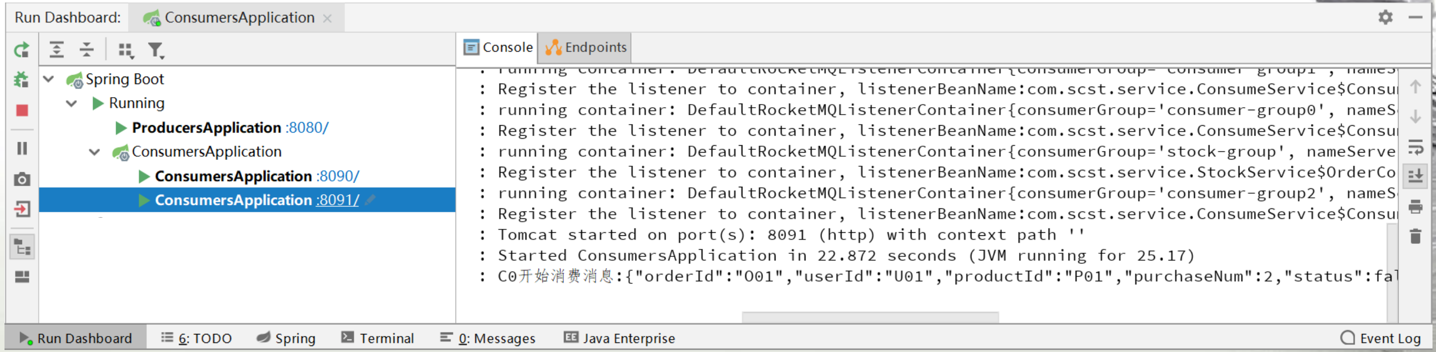
```
: Started ProducersApplication in 17.681 seconds (JVM running for 20.005)
: Initializing Spring DispatcherServlet 'dispatcherServlet'
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 12 ms
: 【发送半消息】Order(orderId=001, userId=U01, productId=P01, purchaseNum=2, status=false, crea
: 【收到半消息响应ACK, 执行本地事务: 】
: Message:GenericMessage [payload=byte[108], headers={rocketmq_TOPIC=msg-topic, rocketmq_FLAG
: Object:Order(orderId=001, userId=U01, productId=P01, purchaseNum=2, status=false, createTim
: 【本地事务状态】UNKNOW
: 【检查本地事务状态】
```



Run Dashboard: ConsumersApplication

- Spring Boot
  - Running
    - ProducersApplication :8080/
    - ConsumersApplication
      - ConsumersApplication :8090/
      - ConsumersApplication :8091/

```
: running container: DefaultRocketMQListenerContainer{consumerGroup='consumer-group2', nameSe
: Register the listener to container, listenerBeanName:com.scst.service.ConsumeService$Consum
: Tomcat started on port(s): 8090 (http) with context path ''
: Started ConsumersApplication in 22.495 seconds (JVM running for 24.483)
: Initializing Spring DispatcherServlet 'dispatcherServlet'
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 13 ms
: C1开始消费消息:Order(orderId=001, userId=U01, productId=P01, purchaseNum=2, status=false, cre
: C2开始消费消息:Order(orderId=001, userId=U01, productId=P01, purchaseNum=2, status=false, cre
: consume C0A81F67373018B4AAC21194C5B00000 cost: 91 ms
: C0开始消费消息:{"orderId":"001","userId":"U01","productId":"P01","purchaseNum":2,"status":fal
```

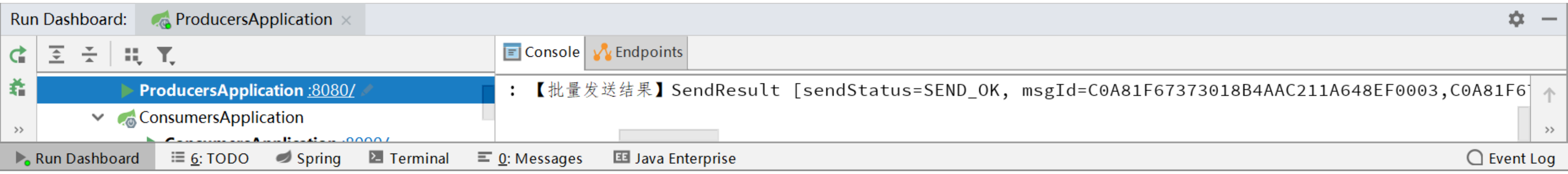


Run Dashboard: ConsumersApplication

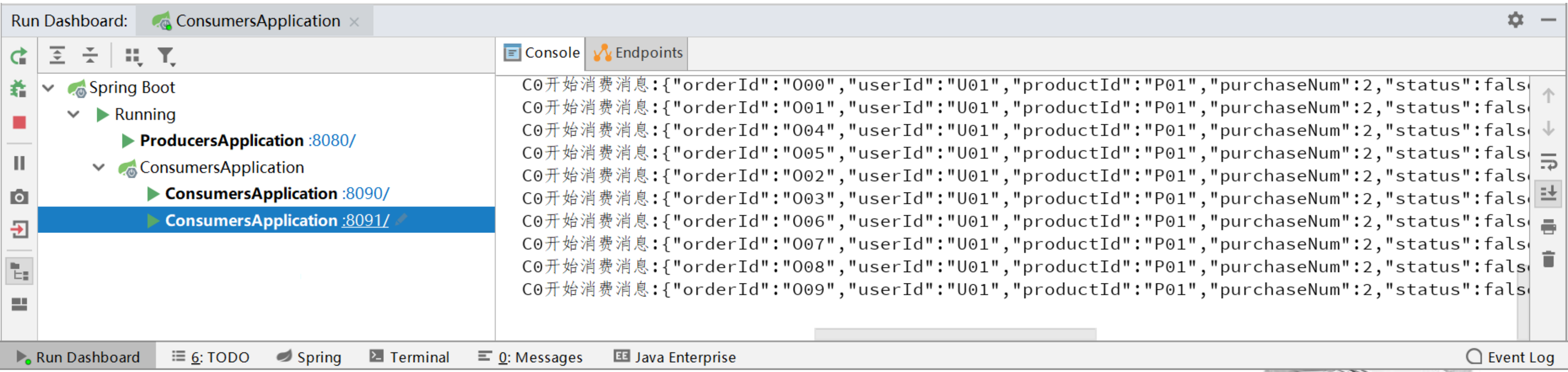
- Spring Boot
  - Running
    - ProducersApplication :8080/
    - ConsumersApplication
      - ConsumersApplication :8090/
      - ConsumersApplication :8091/

```
: running container: DefaultRocketMQListenerContainer{consumerGroup='consumer-group1', names
: Register the listener to container, listenerBeanName:com.scst.service.ConsumeService$Consu
: running container: DefaultRocketMQListenerContainer{consumerGroup='consumer-group0', nameS
: Register the listener to container, listenerBeanName:com.scst.service.ConsumeService$Consu
: running container: DefaultRocketMQListenerContainer{consumerGroup='stock-group', nameServe
: Register the listener to container, listenerBeanName:com.scst.service.StockService$OrderCo
: running container: DefaultRocketMQListenerContainer{consumerGroup='consumer-group2', nameS
: Register the listener to container, listenerBeanName:com.scst.service.ConsumeService$Consu
: Tomcat started on port(s): 8091 (http) with context path ''
: Started ConsumersApplication in 22.872 seconds (JVM running for 25.17)
: C0开始消费消息:{"orderId":"001","userId":"U01","productId":"P01","purchaseNum":2,"status":fal
```

批量消息测试效果

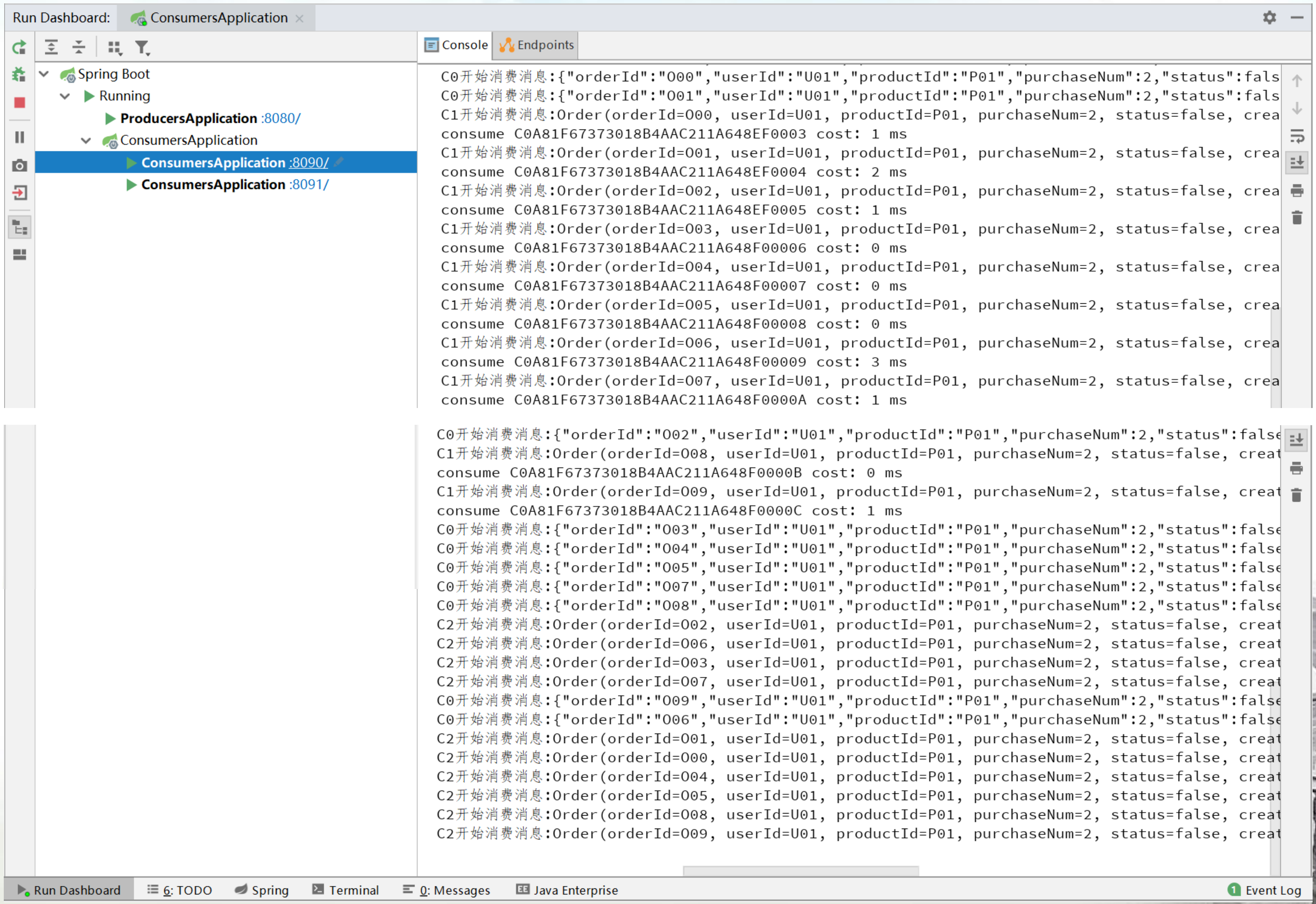


```
: 【批量发送结果】SendResult [sendStatus=SEND_OK, msgId=C0A81F67373018B4AAC211A648EF0003,C0A81F67
```



```
C0开始消费消息:{"orderId":"000","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"001","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"004","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"005","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"002","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"003","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"006","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"007","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"008","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"009","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
```

批量消息测试效果



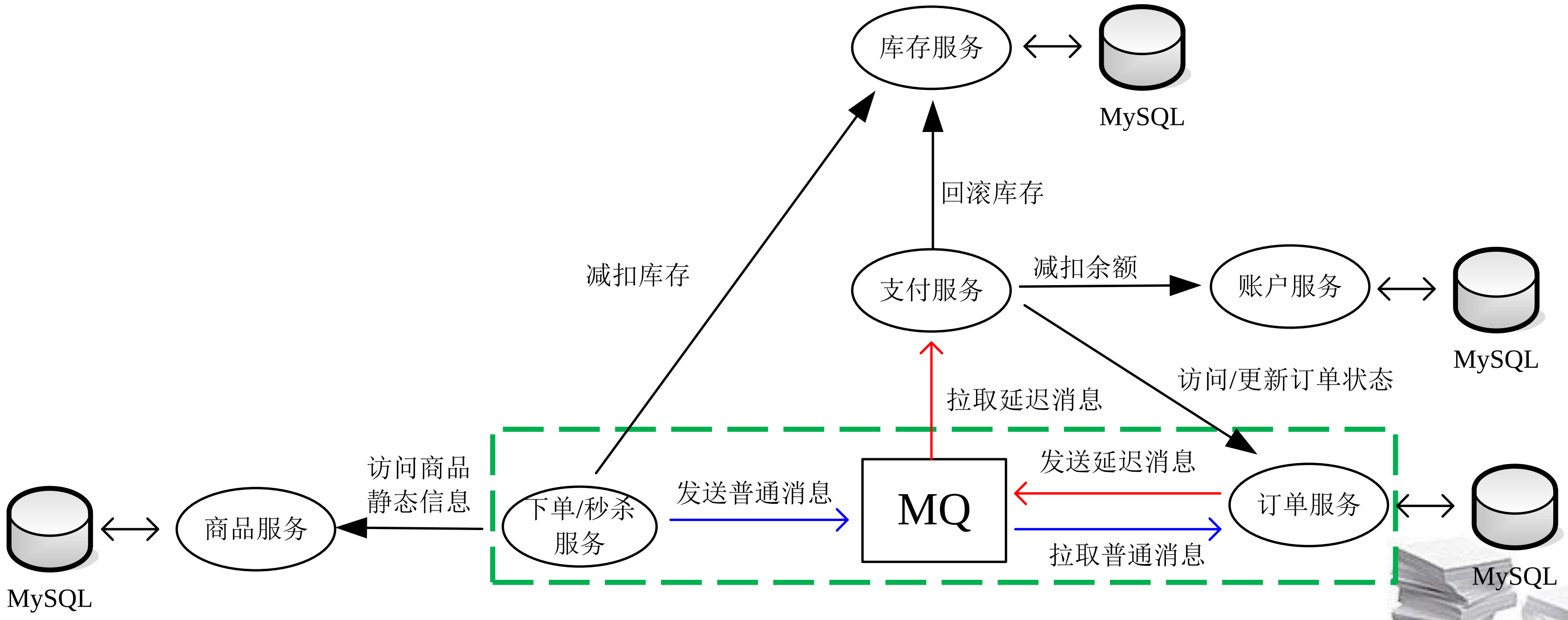
```
C0开始消费消息:{"orderId":"000","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C0开始消费消息:{"orderId":"001","userId":"U01","productId":"P01","purchaseNum":2,"status":fals
C1开始消费消息:Order(orderId=000, userId=U01, productId=P01, purchaseNum=2, status=false, crea
consume C0A81F67373018B4AAC211A648EF0003 cost: 1 ms
C1开始消费消息:Order(orderId=001, userId=U01, productId=P01, purchaseNum=2, status=false, crea
consume C0A81F67373018B4AAC211A648EF0004 cost: 2 ms
C1开始消费消息:Order(orderId=002, userId=U01, productId=P01, purchaseNum=2, status=false, crea
consume C0A81F67373018B4AAC211A648EF0005 cost: 1 ms
C1开始消费消息:Order(orderId=003, userId=U01, productId=P01, purchaseNum=2, status=false, crea
consume C0A81F67373018B4AAC211A648F00006 cost: 0 ms
C1开始消费消息:Order(orderId=004, userId=U01, productId=P01, purchaseNum=2, status=false, crea
consume C0A81F67373018B4AAC211A648F00007 cost: 0 ms
C1开始消费消息:Order(orderId=005, userId=U01, productId=P01, purchaseNum=2, status=false, crea
consume C0A81F67373018B4AAC211A648F00008 cost: 0 ms
C1开始消费消息:Order(orderId=006, userId=U01, productId=P01, purchaseNum=2, status=false, crea
consume C0A81F67373018B4AAC211A648F00009 cost: 3 ms
C1开始消费消息:Order(orderId=007, userId=U01, productId=P01, purchaseNum=2, status=false, crea
consume C0A81F67373018B4AAC211A648F0000A cost: 1 ms
```

```
C0开始消费消息:{"orderId":"002","userId":"U01","productId":"P01","purchaseNum":2,"status":false
C1开始消费消息:Order(orderId=008, userId=U01, productId=P01, purchaseNum=2, status=false, creat
consume C0A81F67373018B4AAC211A648F0000B cost: 0 ms
C1开始消费消息:Order(orderId=009, userId=U01, productId=P01, purchaseNum=2, status=false, creat
consume C0A81F67373018B4AAC211A648F0000C cost: 1 ms
C0开始消费消息:{"orderId":"003","userId":"U01","productId":"P01","purchaseNum":2,"status":false
C0开始消费消息:{"orderId":"004","userId":"U01","productId":"P01","purchaseNum":2,"status":false
C0开始消费消息:{"orderId":"005","userId":"U01","productId":"P01","purchaseNum":2,"status":false
C0开始消费消息:{"orderId":"007","userId":"U01","productId":"P01","purchaseNum":2,"status":false
C0开始消费消息:{"orderId":"008","userId":"U01","productId":"P01","purchaseNum":2,"status":false
C2开始消费消息:Order(orderId=002, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C2开始消费消息:Order(orderId=006, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C2开始消费消息:Order(orderId=003, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C2开始消费消息:Order(orderId=007, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C0开始消费消息:{"orderId":"009","userId":"U01","productId":"P01","purchaseNum":2,"status":false
C0开始消费消息:{"orderId":"006","userId":"U01","productId":"P01","purchaseNum":2,"status":false
C2开始消费消息:Order(orderId=001, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C2开始消费消息:Order(orderId=000, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C2开始消费消息:Order(orderId=004, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C2开始消费消息:Order(orderId=005, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C2开始消费消息:Order(orderId=008, userId=U01, productId=P01, purchaseNum=2, status=false, creat
C2开始消费消息:Order(orderId=009, userId=U01, productId=P01, purchaseNum=2, status=false, creat
```

8.3.2 电商秒杀应用案例

- 本案例实现基于**Redis+RocketMQ**的电商秒杀系统，由秒杀服务和订单服务来执行订单的生产和消费，通过**Redis**的限流和**RocketMQ**的消峰作用，防止超卖发生。

电商项目典型场景


库存服务
MySQL
回滚库存
减扣库存
支付服务
减扣余额
账户服务
MySQL
访问/更新订单状态
拉取延迟消息
访问商品
静态信息
发送普通消息
MQ
发送延迟消息
订单服务
MySQL
下单/秒杀
服务
拉取普通消息
商品服务
MySQL

搭建步骤

- 1）创建实体类
- 2）创建秒杀服务
- 3）创建订单服务
- 4）效果测试

1）创建实体类

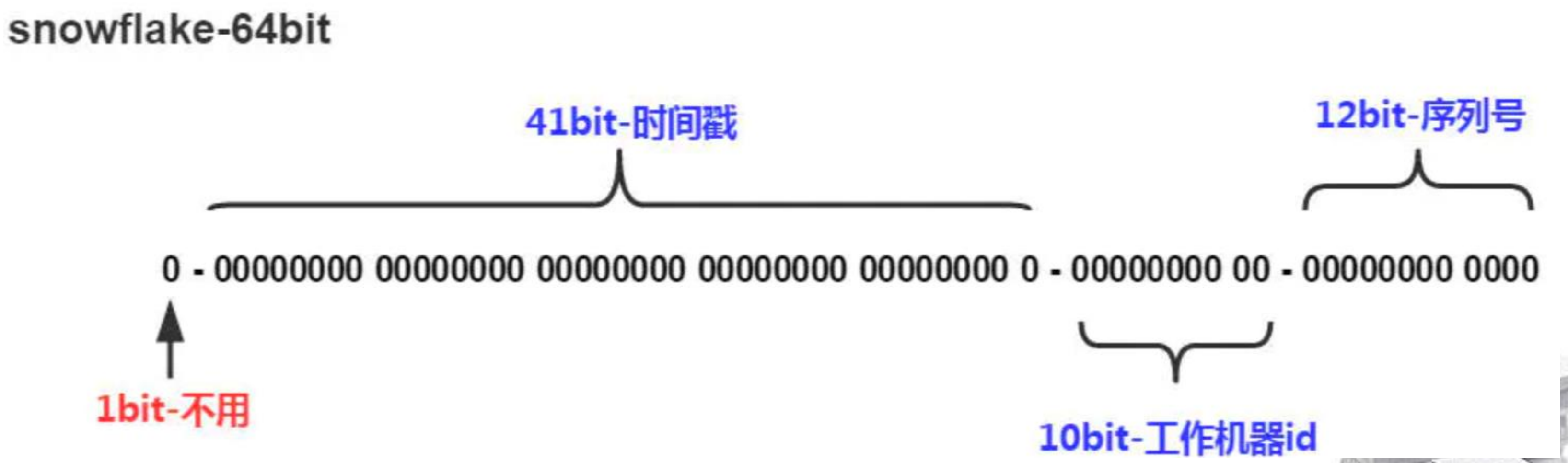
- 在common模块的com.scst.domain包中创建一个实体类Stock，模拟商品库存；以包中的Order类，作为订单实体类。
- Order类中的orderId属性值要求由雪花算法生成，为此，在common模块中新建一个com.scst.utils包，并在包中创建一个IdWorker类，以实现雪花算法。

Stock类

```
package com.scst.domain;
import lombok.Data;
import org.springframework.stereotype.Component;

@Component
@Data
public class Stock {
    private String productId = "NO005";
    private String productName = "苹果";
    private Integer stockNum = 100;        //初始库存量
}
```

雪花算法（SnowFlake）

- SnowFlake算法生成id的结果是一个64bit大小的整数。



2）创建秒杀服务

- 搭建步骤：
  - ✧①配置缓存中间件连接
  - ✧②引入依赖启动器
  - ✧③配置序列化方式和分布式锁
  - ✧④创建秒杀服务类
  - ✧⑤创建**Web**控制器类

①配置缓存中间件连接

- 在producers模块的全局配置文件application.properties中添加Redis缓存中间件的连接配置，示例代码如下。

```
#数据库名db0
spring.redis.database=0
# Redis服务器地址
spring.redis.host=192.168.31.173
# Redis服务器连接端口
spring.redis.port=6379
# Redis服务器连接密码（默认为空）
spring.redis.password=
```

②引入依赖启动器

- 在producers模块的pom.xml文件中引入Redis和Redisson依赖，示例代码如下。

```
<!-- Spring Data Redis依赖启动器 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<!-- 使用redisson作为所有分布式锁，分布式对象等功能框架-->
<dependency>
    <groupId>org.redisson</groupId>
    <artifactId>redisson</artifactId>
    <version>3.12.0</version>
</dependency>
```

③配置序列化方式和分布式锁

- 在producers模块新建一个com.scst.config包，并在包中创建一个自定义配置类RedisConfig
  - ✧为RedisTemplate设置序列化方式，分别对缓存数据的key和value进行序列化方式定制，其中key定制为StringRedisSerializer（即String格式），value定制为Jackson2JsonRedisSerializer（即JSON格式）。
  - ✧为Redisson设置服务器地址为：redis://192.168.31.173:6379

RedisConfig类

```
package com.scst.config;
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.config.Config;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.*;
@Configuration   // 定义一个配置类
public class RedisConfig {
    @Value("redis://"+"${spring.redis.host}"+":"+"${spring.redis.port}")
    private String redissonserver; //redis://192.168.31.173:6379
```

RedisConfig类

```
@Bean
 RedissonClient redisson(){
     Config config = new Config();
     config.useSingleServer().setAddress(redissonserver);
     return Redisson.create(config);
 }
```

RedisConfig类

```
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();
        template.setConnectionFactory(factory);
        // key采用String的序列化方式
        template.setKeySerializer(new StringRedisSerializer());
        // hash的key也采用String的序列化方式
        template.setHashKeySerializer(new StringRedisSerializer());
        // value序列化方式采用jackson
        template.setValueSerializer(new GenericJackson2JsonRedisSerializer());
        // hash的value序列化方式采用jackson
        template.setHashValueSerializer(new GenericJackson2JsonRedisSerializer());
        template.afterPropertiesSet();
        return template;
    }
}
```

④创建秒杀服务类

- 在producers模块的com.scst.service包中新建一个业务类SecKillService，实现一旦模块启动就加载秒杀商品信息到Redis，以及秒杀成功订单的生成和发送。

SecKillService类

```
package com.scst.service;
import com.scst.domain.Order;
import com.scst.domain.Stock;
import com.scst.utils.IdWorker;
import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.redisson.api.RLock;
import org.redisson.api.RedissonClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.context.event.EventListener;
import org.springframework.data.redis.core.BoundHashOperations;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;
import java.util.Date;
import java.util.concurrent.ThreadLocalRandom;
import java.util.concurrent.TimeUnit;
```

SecKillService类

```
//秒杀服务
@Slf4j
@Service
public class SecKillService {
    @Autowired
    private RedisTemplate redisTemplate;
    @Autowired
    private RocketMQTemplate rocketMQTemplate;
    @Autowired
    private IdWorker idWorker;
    @Autowired
    private RedissonClient redisson;
    @Autowired
    private Stock stock;
    private String destination;
    private Object payload;
    public static final String SEC_KILL_GOODS_KEY = "secondsKillGoods";
```

SecKillService类

```
//商品秒杀
public void secKill(String topic, String tag) throws InterruptedException{
    String userId="User"+buildRandomUserId();
    String productId=stock.getProductId();
    this.destination = topic;
    if (tag != null) {
        this.destination = topic + ":" + tag;
    }
    RLock lock = redisson.getLock("seckill:" + productId);                    //定义分布式锁
    boolean res = lock.tryLock(10, 30, TimeUnit.SECONDS);                    //尝试加锁，最多等待10s，上锁以后30s自动解锁
    //lock.lock();
    //Future<Boolean> res = lock.tryLockAsync(100, 10, TimeUnit.SECONDS);
    if(res) {
        this.payload = createOrder(productId, userId);
        if (this.payload != null) {
            //rocketMQTemplate.syncSendOrderly(this.destination, this.payload,"hashKey");
            rocketMQTemplate.syncSend(this.destination, this.payload);
        }
        if (lock.isLocked() && lock.isHeldByCurrentThread()) {            //只有加锁成功才需要解锁，且自己加的锁自己解，不能解别人的锁
            lock.unlock();
        }
    }
}
```

SecKillService类

```
//生成订单
private Order createOrder(String productId, String userId) {
    //获取操作redis hash类型的操作类
    BoundHashOperations<String, Object, Object> hashOperations = redisTemplate.boundHashOps(SEC_KILL_GOODS_KEY);
    Integer amount = (Integer) hashOperations.get(productId);        //从Redis获得秒杀商品信息
    if (amount == null || amount <= 0) {
        return null;
    }
    Long value = hashOperations.increment(productId, -1);          //redis预扣库存
    if (value <= 0) {
        hashOperations.delete(productId);                          // 如果扣完后库存为0，则删除当前商品
    }
    Order order = new Order();
    order.setOrderId(String.valueOf(idWorker.nextId()));
    order.setUserId(userId);
    order.setProductId(productId);
    order.setPurchaseNum(1);
    order.setCreateTime(new Date());
    order.setStatus(false);
    return order;
}
```

SecKillService类

```
    private Integer buildRandomUserId(){
        return ThreadLocalRandom.current().nextInt(100000) + 50;
    }

    //模块一旦启动就往redis中加载秒杀商品
    @EventListener
    public void contextRefreshedEventListener(ContextRefreshedEvent contextRefreshedEvent) {
        if(redisTemplate == null){
            return;
        }
        System.out.println("往redis中加载秒杀商品开始！");
        //获取操作redis hash类型的操作类
        BoundHashOperations<String, Object, Object> hashOperations=redisTemplate.boundHashOps(SEC_KILL_GOODS_KEY);
        hashOperations.put(stock.getProductId(),stock.getStockNum());            //往redis中加载数据
        System.out.println("往redis中加载秒杀商品成功！");
    }
}
```

⑤创建Web控制器类

■ 在producers模块的com.scst.controller包中新建一个Web控制类SecKillController，并在该类中实现商品秒杀请求。

SecKillController类

```
package com.scst.controller;
import com.scst.service.SecKillService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class SecKillController {
    @Autowired
    private SecKillService secKillService;
    private String topic = "order-topic";
    private String tag = "TagA";
    @GetMapping("/seckill")
    public void seckill() throws Exception{
        secKillService.secKill(topic, tag);
    }
}
```

3）创建订单服务

- 搭建步骤：
  - ✧①创建订单服务类
  - ✧②创建Web控制器类

①创建库存服务类

- 在consumers模块的com.scst.service包中新建一个业务类OrderService，实现秒杀订单的监听和消费。

OrderService类

```
package com.scst.service;
import com.scst.domain.Order;
import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.List;

//库存服务
@Slf4j
@Service
public class OrderService {
    private List<Order> consumeRecds = new ArrayList<>();
```

OrderService类

```
    @Component
    //@RocketMQMessageListener(nameServer = "${rocketmq.name-server}", topic = "order-topic",
consumerGroup = "${rocketmq.consumer.group}",consumeMode=ConsumeMode.ORDERLY)
    @RocketMQMessageListener(nameServer = "${rocketmq.name-server}", topic = "order-topic", consumerGroup
= "${rocketmq.consumer.group}")
    class OrderConsumer implements RocketMQListener<Order> {
        @Override
        public void onMessage(Order order) {
            //消息体（Payload）o为Java对象
            log.info("开始记载订单:{}", order);
            consumeRecds.add(order);
        }
    }

    public String secKillSuccesses(){ return "当前服务记载的秒杀成功订单数："+this.consumeRecds.size();}
    public List<Order> getOrders(){ return consumeRecds;}
}
```

②创建Web控制器类

- 在consumers模块中新建一个com.scst.controller包，并在包中新建一个Web控制类OrderController，并在该类中实现查看秒杀结果请求。

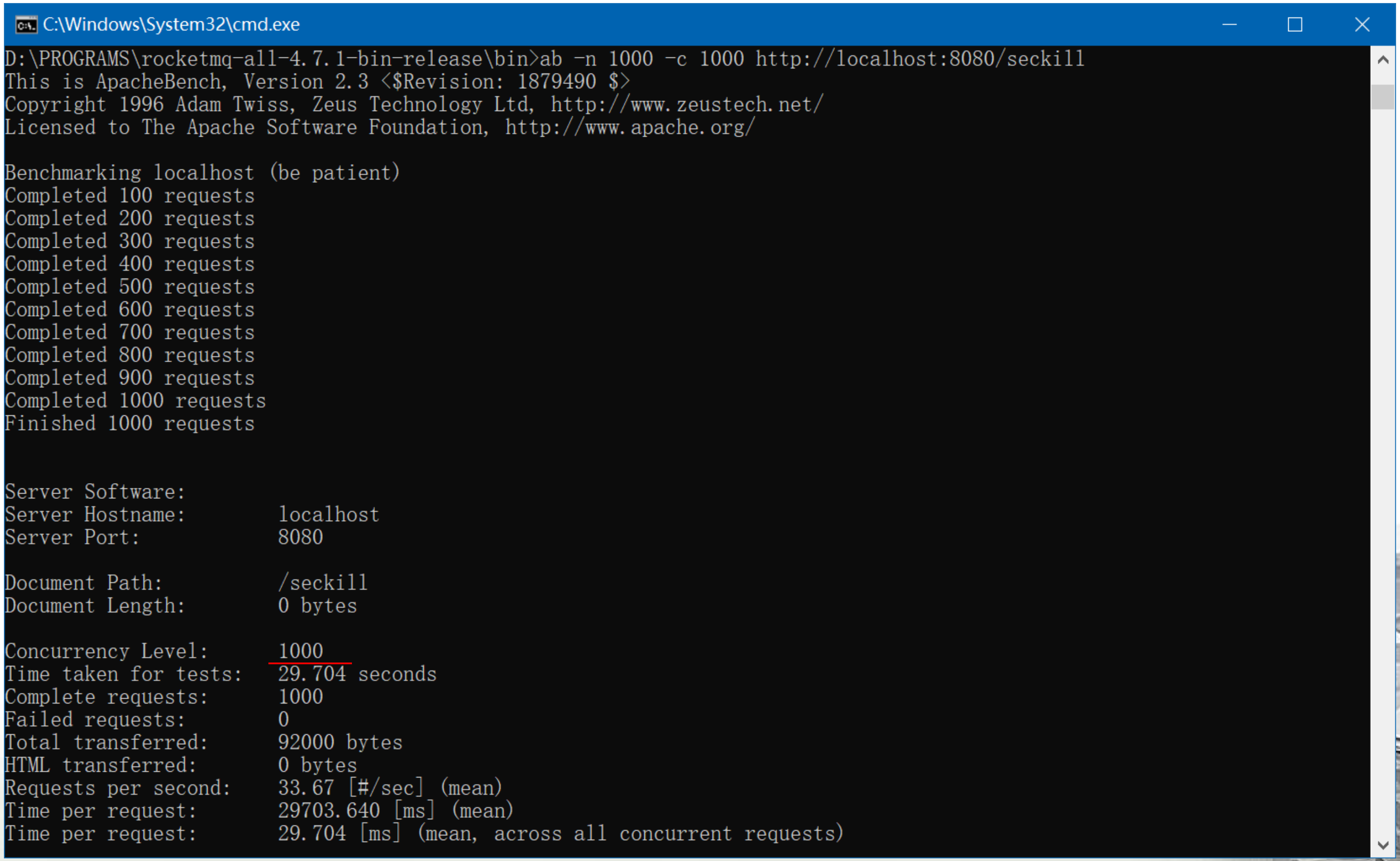
OrderController类

```
package com.scst.controller;
import com.scst.domain.Order;
import com.scst.service.OrderService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;
@RestController
public class OrderController {
    @Autowired
    private OrderService orderService;
    @GetMapping("/successes")
    public String successes() {  return orderService.secKillSuccesses(); }
    @GetMapping("/orders")
    public List<Order> orders() { return orderService.getOrders();}
}
```

4）效果测试

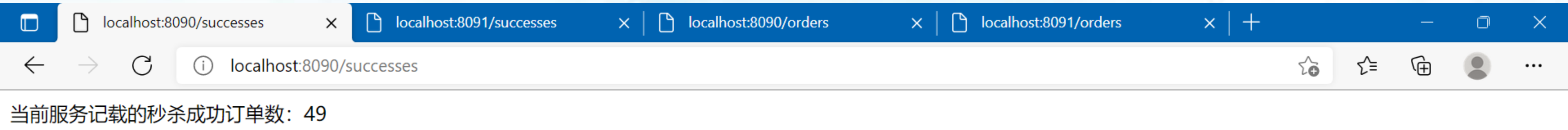
- 测试步骤：
  - ✧ ①启动Name Server
  - ✧ ②启动Broker Server
  - ✧ ③启动Redis Server
  - ✧ ④以8080端口启动ProducersApplication类
  - ✧ ⑤先后以8090和8091端口启动ConsumersApplication类
  - ✧ ⑥使用AB工具通过执行以下命令模拟大量并发秒杀用户：
    - >ab -n 1000 -c 1000 http://localhost:8080/seckill
  - ✧ ⑦分别访问以下链接，观察两个库存服务实例记载的秒杀成功订单及其总数：
    - http://localhost:8090/successes
    - http://localhost:8091/successes
    - http://localhost:8090/orders
    - http://localhost:8091/orders

测试结果

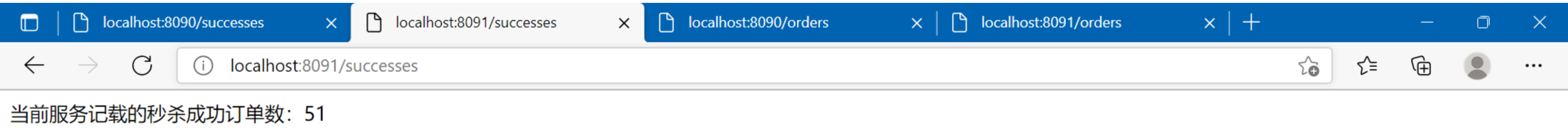


```
D:\PROGRAMS\rocketmq-all-4.7.1-bin-release\bin>ab -n 1000 -c 1000 http://localhost:8080/seckill
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:
Server Hostname:        localhost
Server Port:            8080

Document Path:          /seckill
Document Length:        0 bytes

Concurrency Level:      1000
Time taken for tests:   29.704 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      92000 bytes
HTML transferred:       0 bytes
Requests per second:    33.67 [#/sec] (mean)
Time per request:       29703.640 [ms] (mean)
Time per request:       29.704 [ms] (mean, across all concurrent requests)
```

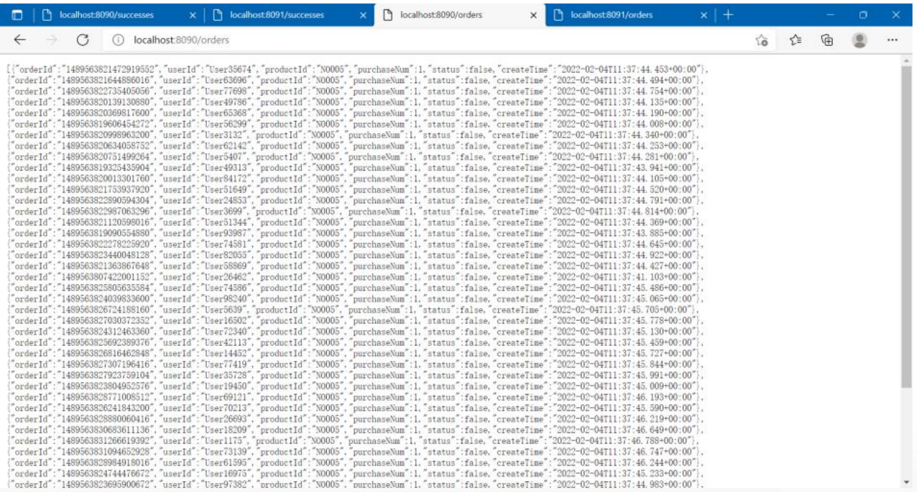
测试结果



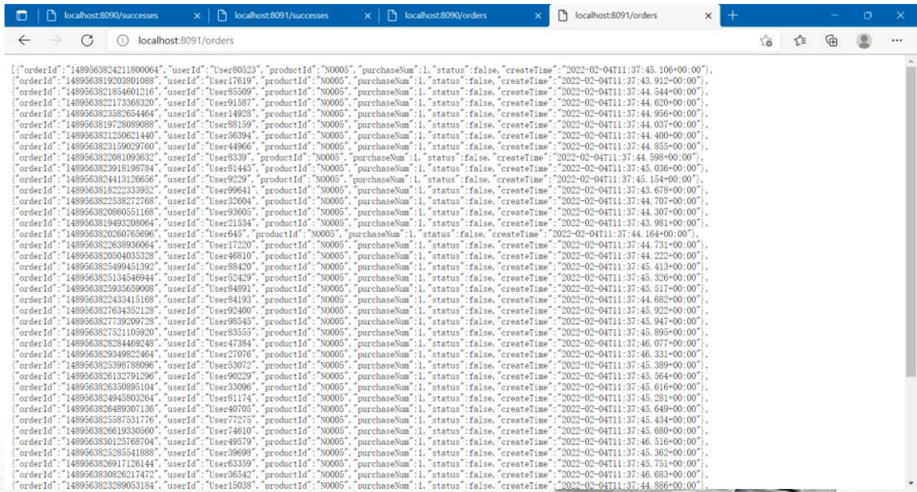
当前服务记载的秒杀成功订单数：49



当前服务记载的秒杀成功订单数：51



```
[{"orderId":"1489563821472919552","userId":"User35674","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.453+00:00"},
{"orderId":"1489563821644886016","userId":"User63696","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.494+00:00"},
{"orderId":"1489563822735405056","userId":"User77698","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.754+00:00"},
{"orderId":"1489563820139130880","userId":"User49786","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.135+00:00"},
{"orderId":"1489563820369817600","userId":"User65368","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.190+00:00"},
{"orderId":"1489563819606454272","userId":"User56299","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.008+00:00"},
{"orderId":"1489563820998963200","userId":"User3132","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.340+00:00"},
{"orderId":"1489563820634058752","userId":"User62142","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.253+00:00"},
{"orderId":"1489563820751499264","userId":"User5407","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.281+00:00"},
{"orderId":"1489563819325435904","userId":"User49313","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:43.941+00:00"},
{"orderId":"1489563820013301760","userId":"User84172","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.105+00:00"},
{"orderId":"1489563821753937920","userId":"User51649","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.520+00:00"},
{"orderId":"1489563822890594304","userId":"User24853","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.791+00:00"},
{"orderId":"1489563822987063296","userId":"User3699","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.814+00:00"},
{"orderId":"1489563821120598016","userId":"User51344","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.369+00:00"},
{"orderId":"1489563819090554880","userId":"User93987","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:43.885+00:00"},
{"orderId":"1489563822278225920","userId":"User74581","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.645+00:00"},
{"orderId":"1489563823440048128","userId":"User82055","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.922+00:00"},
{"orderId":"1489563821363867648","userId":"User58869","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.427+00:00"},
{"orderId":"1489563807422001152","userId":"User26462","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:41.103+00:00"},
{"orderId":"1489563825805635584","userId":"User74586","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.486+00:00"},
{"orderId":"1489563824039833600","userId":"User98240","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.065+00:00"},
{"orderId":"1489563826724188160","userId":"User5639","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.705+00:00"},
{"orderId":"1489563827030372352","userId":"User16502","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.778+00:00"},
{"orderId":"1489563824312463360","userId":"User72340","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.130+00:00"},
{"orderId":"1489563825692389376","userId":"User42113","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.459+00:00"},
{"orderId":"1489563826816462848","userId":"User14452","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.727+00:00"},
{"orderId":"1489563827307196416","userId":"User77419","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.844+00:00"},
{"orderId":"1489563827923759104","userId":"User35728","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.991+00:00"},
{"orderId":"1489563823804952576","userId":"User19450","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.009+00:00"},
{"orderId":"1489563828771008512","userId":"User69121","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.193+00:00"},
{"orderId":"1489563826241843200","userId":"User70213","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.590+00:00"},
{"orderId":"1489563828880060416","userId":"User26693","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.219+00:00"},
{"orderId":"1489563830683611136","userId":"User18209","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.649+00:00"},
{"orderId":"1489563831266619392","userId":"User1175","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.788+00:00"},
{"orderId":"1489563831094652928","userId":"User73139","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.747+00:00"},
{"orderId":"1489563828984918016","userId":"User61595","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.244+00:00"},
{"orderId":"1489563824744476672","userId":"User16975","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.233+00:00"},
{"orderId":"1489563823695900672","userId":"User97382","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.983+00:00"},
```



```
[{"orderId":"1489563824211800064","userId":"User80523","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.106+00:00"},
{"orderId":"1489563819203801088","userId":"User17619","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:43.912+00:00"},
{"orderId":"1489563821854601216","userId":"User85509","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.544+00:00"},
{"orderId":"1489563822173368320","userId":"User91587","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.620+00:00"},
{"orderId":"1489563823582654464","userId":"User14928","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.956+00:00"},
{"orderId":"1489563819728089088","userId":"User88159","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.037+00:00"},
{"orderId":"1489563821250621440","userId":"User56394","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.400+00:00"},
{"orderId":"1489563823159029760","userId":"User44966","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.855+00:00"},
{"orderId":"1489563822081093632","userId":"User8339","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.598+00:00"},
{"orderId":"1489563823918198784","userId":"User81445","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.036+00:00"},
{"orderId":"1489563824413126656","userId":"User9229","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.154+00:00"},
{"orderId":"1489563818222333952","userId":"User99641","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:43.678+00:00"},
{"orderId":"1489563822538272768","userId":"User32604","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.707+00:00"},
{"orderId":"1489563820860551168","userId":"User93605","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.307+00:00"},
{"orderId":"1489563819493208064","userId":"User21534","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:43.981+00:00"},
{"orderId":"1489563820260765696","userId":"User645","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.164+00:00"},
{"orderId":"1489563822638936064","userId":"User17220","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.731+00:00"},
{"orderId":"1489563820504035328","userId":"User46810","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.222+00:00"},
{"orderId":"1489563825499451392","userId":"User88420","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.413+00:00"},
{"orderId":"1489563825134546944","userId":"User52429","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.326+00:00"},
{"orderId":"1489563825935659008","userId":"User84891","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.517+00:00"},
{"orderId":"1489563822433415168","userId":"User84193","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.682+00:00"},
{"orderId":"1489563827634352128","userId":"User92400","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.922+00:00"},
{"orderId":"1489563827739209728","userId":"User98545","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.947+00:00"},
{"orderId":"1489563827521105920","userId":"User83555","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.895+00:00"},
{"orderId":"1489563828284469248","userId":"User47384","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.077+00:00"},
{"orderId":"1489563829349822464","userId":"User27076","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.331+00:00"},
{"orderId":"1489563825398788096","userId":"User53072","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.389+00:00"},
{"orderId":"1489563826132791296","userId":"User90229","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.564+00:00"},
{"orderId":"1489563826350895104","userId":"User33096","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.616+00:00"},
{"orderId":"1489563824945803264","userId":"User81174","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.281+00:00"},
{"orderId":"1489563826489307136","userId":"User40705","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.649+00:00"},
{"orderId":"1489563825587531776","userId":"User77275","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.434+00:00"},
{"orderId":"1489563826619330560","userId":"User74610","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.680+00:00"},
{"orderId":"1489563830125768704","userId":"User49579","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.516+00:00"},
{"orderId":"1489563825285541888","userId":"User39698","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.362+00:00"},
{"orderId":"1489563826917126144","userId":"User63359","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:45.751+00:00"},
{"orderId":"1489563830826217472","userId":"User36542","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:46.683+00:00"},
{"orderId":"1489563823289053184","userId":"User15038","productId":"N0005","purchaseNum":1,"status":false,"createTime":"2022-02-04T11:37:44.886+00:00"},
```

本章小结

- 本章具体讲解了：
  - ✧8.1 消息服务概述
  - ✧8.2 RocketMQ基础
  - ✧8.3 RocketMQ整合案例

聪明等于勤奋
努力就是幸福