# 复习

1.baidu dubbo有没有负载均衡策略

## 动态代理

解决问题：代码重用

1.动态代理

2.组合 Tools.

3.继承

继承是为了多态，回调，重用代码用动态代理或组合

MyInvocationHandler implements InvocationHandler

{

Invoke()

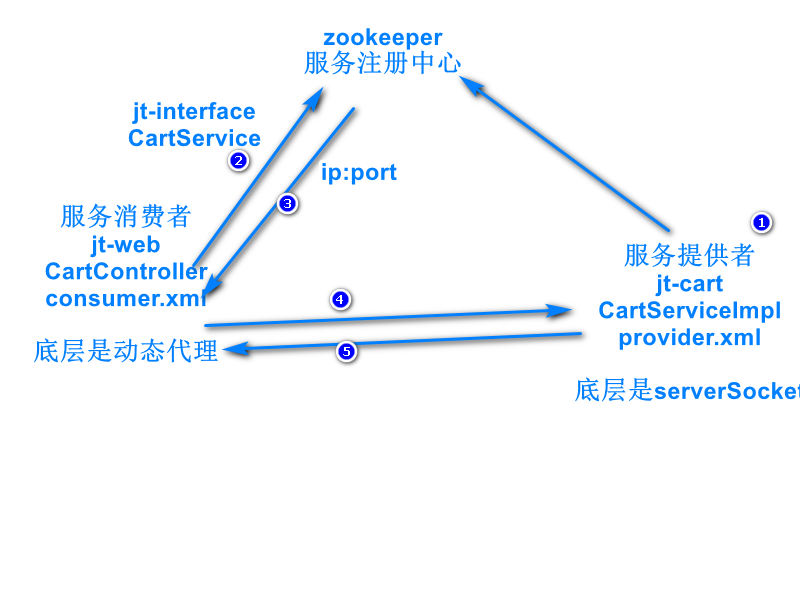
}

动态代理对象$proxy0=Proxy.newProxyInstance(类加载器,Class[],myInvocationHandler) 1,创建动态类，2，根据动态类创建代理对象

cartService.findCardByUserId() 先执行invoke()

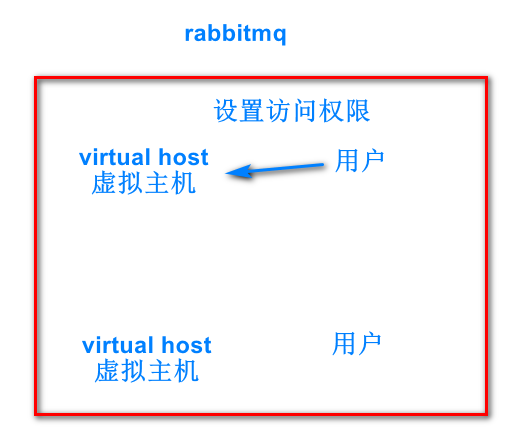
1. method.invoke()执行目标方法

## Dubbo



# 消息队列

## 后台管理配置



### 开三个服务器

[root@localhost ~]# service iptables stop

[root@localhost ~]# cd /usr/local/src

[root@localhost src]# ls

[root@localhost src]# cd redis-3.2.8

[root@localhost redis-3.2.8]# cd cluster

[root@localhost cluster]# sh start.sh

[root@localhost cluster]# cd /usr/local/src

[root@localhost src]# cd zookeeper-3.4.8

[root@localhost zookeeper-3.4.8]# cd bin

[root@localhost bin]# ls

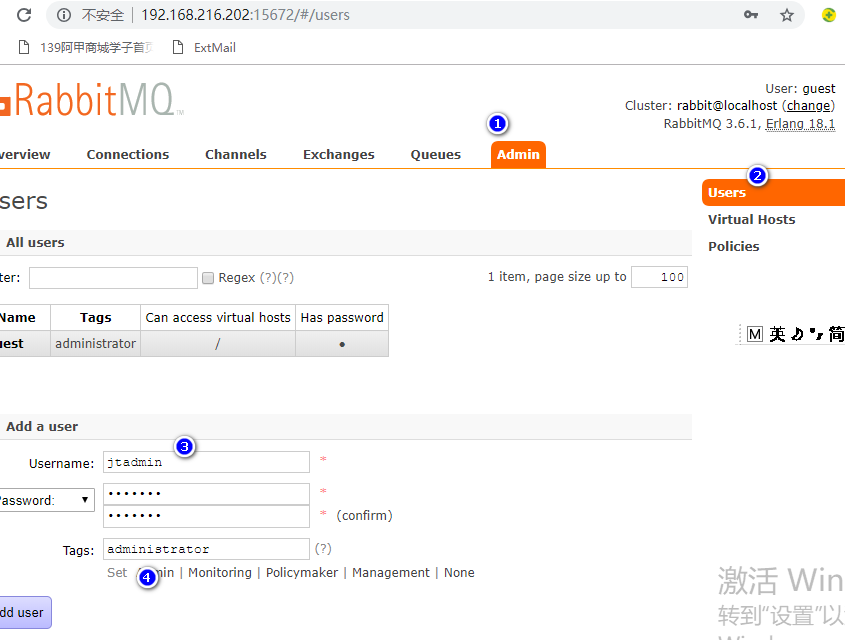
[root@localhost bin]# sh zkServer.sh start

[root@localhost bin]# cd /usr/sbin

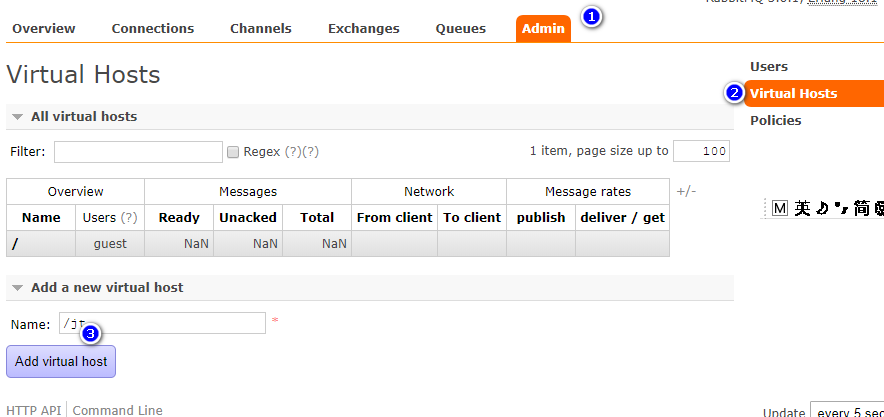
[root@localhost sbin]# ls ra\*

[root@localhost sbin]# service rabbitmq-server start

### 创建用户



### 创建虚拟主机



### 设置权限

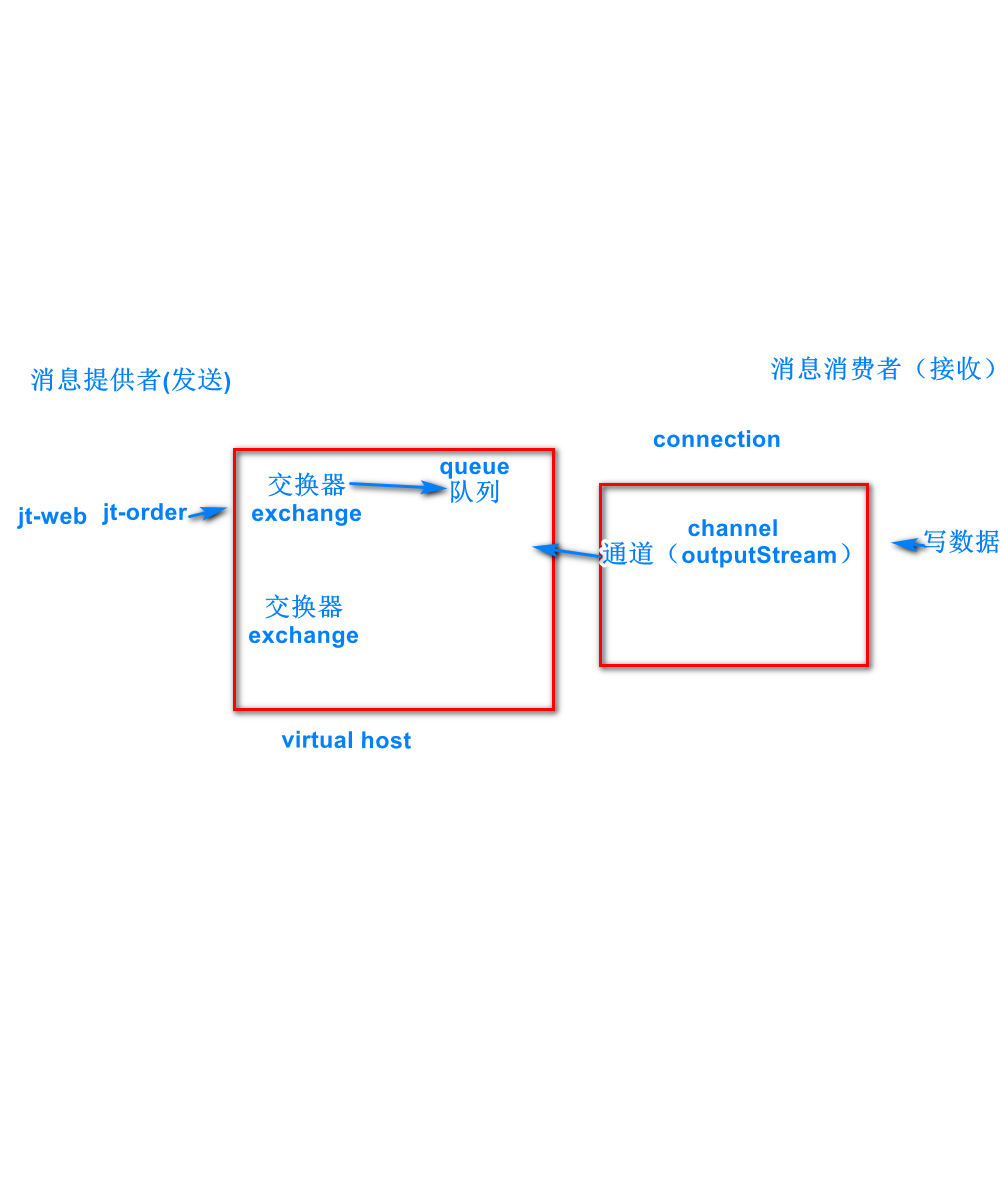
在虚拟主机管理界面选 中/jt,单击/jt



## 五种模式

### 简单模式

#### 需求



#### 分析

Test\_1\_simple\_provider发消息

Test\_1\_simple\_consumer接收消息

#### 消息发送（提供者）

建立连接

ConnectionFactory

Connection

指定队列

createChannel

channel.queueDeclare

发数据

Channel.basicPublish

关闭

Close

**public** **class** Test\_1\_simple\_provider {

@Test

**public** **void** provider() **throws** Throwable

{

//1,建立连接

ConnectionFactory factory

=**new** ConnectionFactory();

factory.setHost("192.168.216.201");

factory.setPort(5672);

factory.setUsername("jtadmin");

factory.setPassword("jtadmin");

factory.setVirtualHost("/jt");

//2,创建通道

//com.rabbitmq.client.Connection

Connection connection=factory.newConnection();

//com.rabbitmq.client

Channel channel=connection.createChannel();

//3,创建队列

String queueName="order";

//p1:队列名

//p2:durable true 保存到硬盘上，重启后还有

//p3:exclusive true 消息只能由connection消费 false别人能消费

//p4:autoDelete true 消息处理完了,删除队列

//p5:arguments配置

channel.queueDeclare

(queueName, **true**, **false**, **false**, **null**);

//4,发消息

//p1:是交换机，给"",发到default exchange

//p2:routingKey是数据的标识

channel.basicPublish

("", queueName, **null**, "msg1".getBytes());

//5,关闭

channel.close();

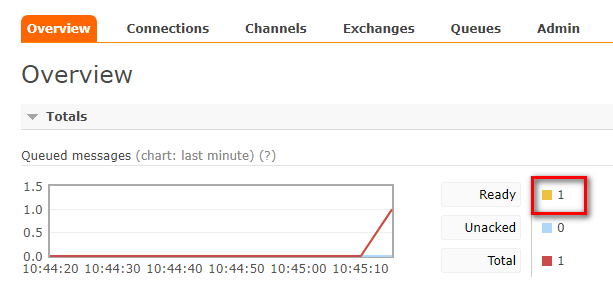
connection.close();

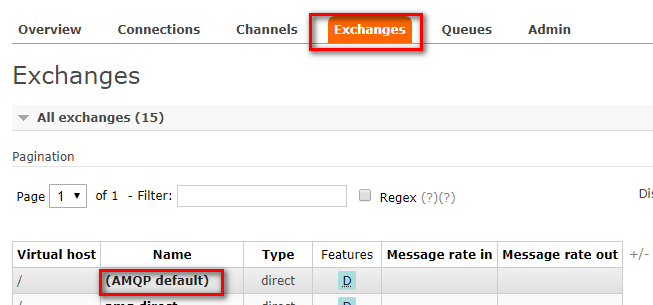
//在后台管理overview,看到队列中消息的个数

}

}

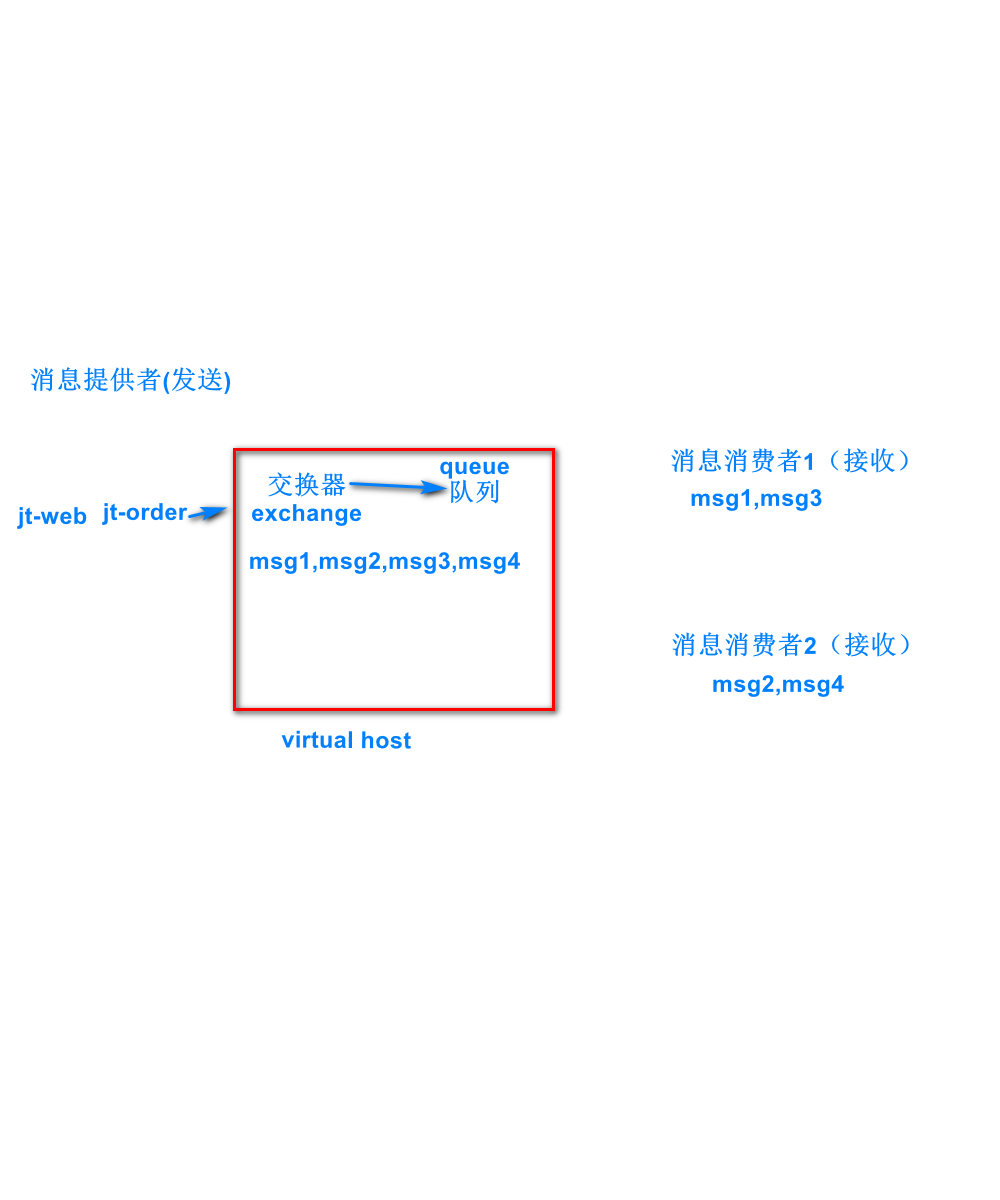
#### 后台查看数据





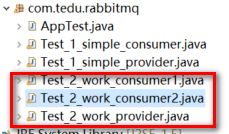
#### 消息接收(消费者)

### 工作模式



拷贝一个provider

拷贝一个consumer改成consumer1,consumer2

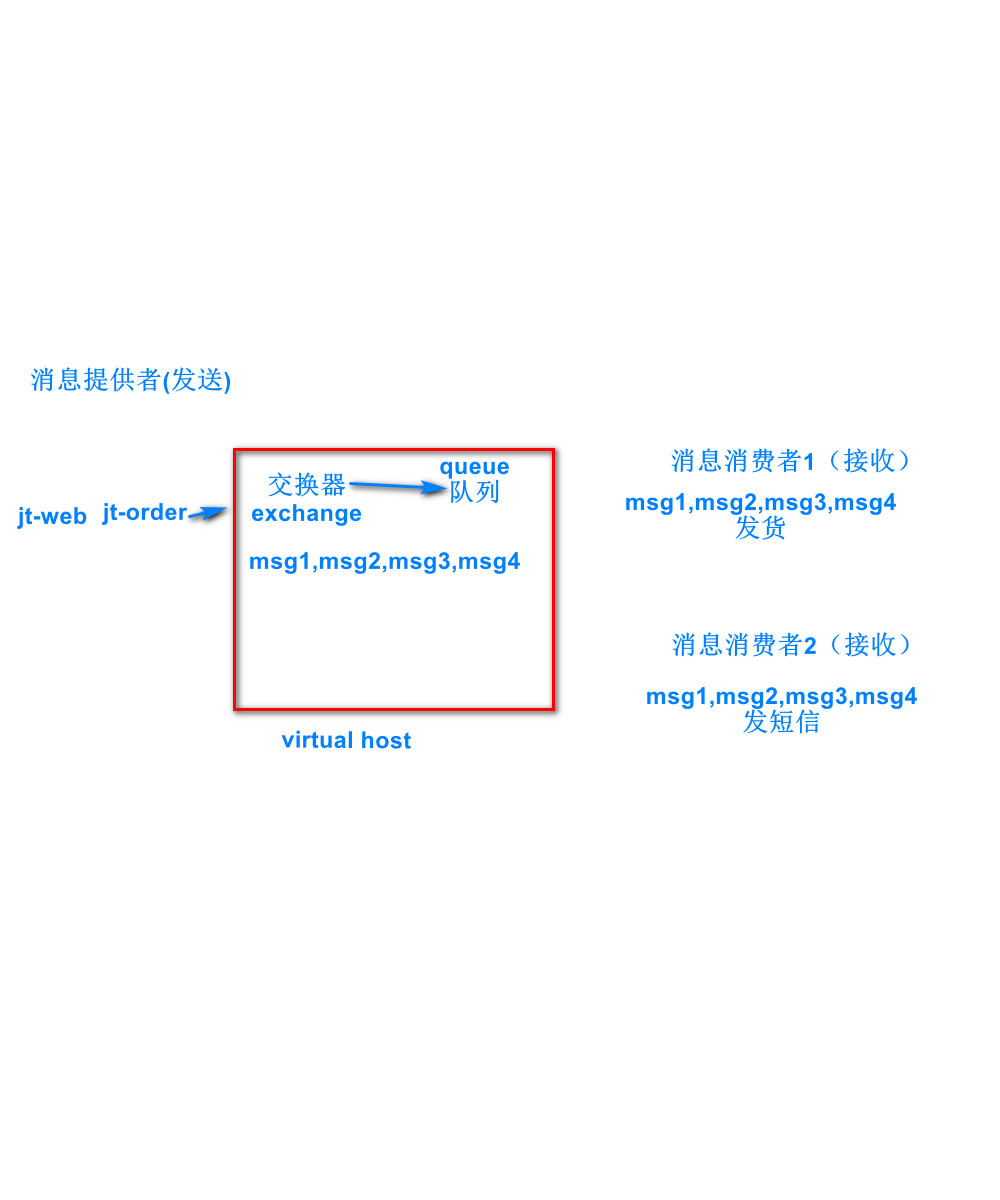


先启动两个消息者、

多次启动提供者

控制台看到消费者1取到信息，消费者2也能取到信息

### 订阅模式



拷贝test2\_work\_provider改成test3\_publish\_provider

拷贝消费者1,消费者2

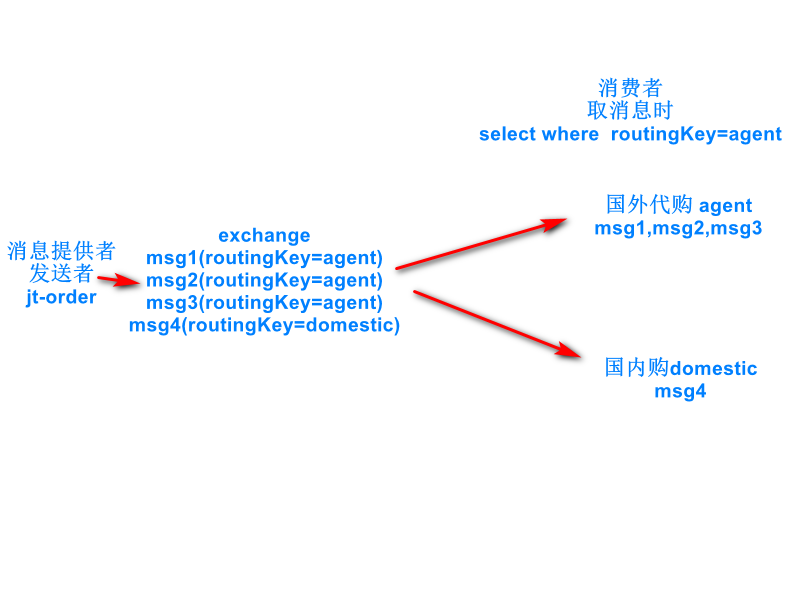
### 总结

简单模式只有一个消费者

工作模式有多个消费者

订阅模式，这个消息会被多个消费者处理，案例微信公众号，聊天

### 路由模式

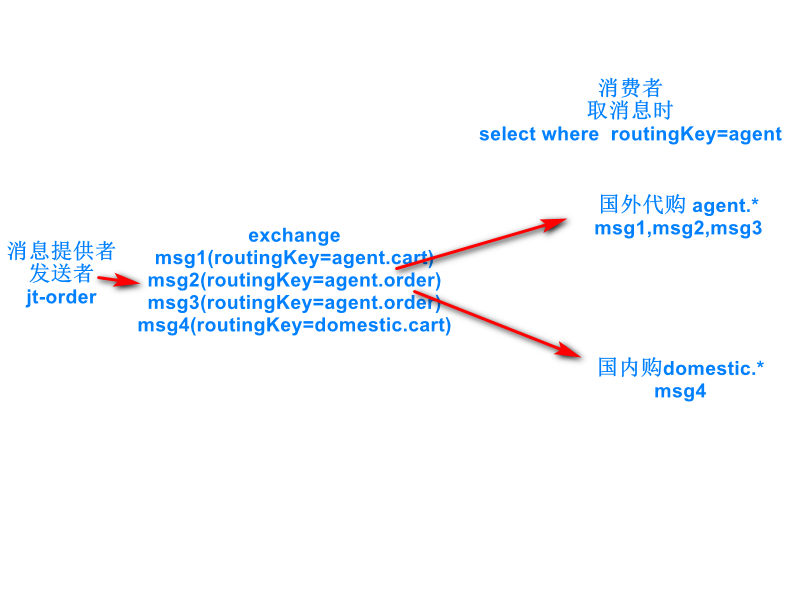


Direct

路由模式，每个消费者取到的数据不一样

订阅模式，每个消费者取到的数据是一样的

### 主题模式



## 五种模式总结

简单simple：一个消费者

工作work:多个消费者，每个消费者处理的信息是不一样的，用在高并发

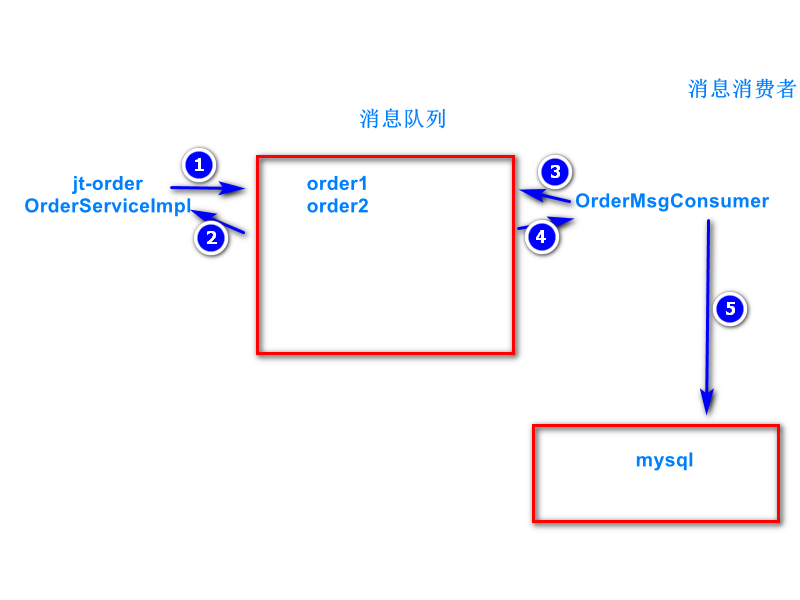
订阅fanout:每个消费者得到信息一样的，

路由direct: 每个消费者得到信息不一样的,信息有routingkey,消费者执行select where routingkey=agent

主题topic:消息分类，消费者取消息时用\*

## 京淘项目中使用rabbitmq

### 需求



### 消息提供者

1. 添加rabbitmq依赖

Jt-parent已经添加过了

<!-- 消息队列 -->

<dependency>

<groupId>com.rabbitmq</groupId>

<artifactId>amqp-client</artifactId>

<version>3.5.1</version>

</dependency>

<dependency>

<groupId>org.springframework.amqp</groupId>

<artifactId>spring-rabbit</artifactId>

<version>1.4.0.RELEASE</version>

</dependency>

1. 在property中创建rabbitmq.properties

Jt-order/main/resources/properties/rabbitmq.properties

rabbit.ip=192.168.216.202

rabbit.port=5672

rabbit.username=jtadmin

rabbit.password=jtadmin

rabbit.vhost=/jt

1. applicationContext.xml加载properties

/resources/spring/applicationContext.xml 中增加一个value

<list>

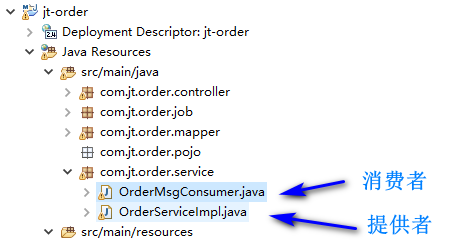
<value>classpath:/property/jdbc.properties</value>

<value>classpath:/property/redis.properties</value>

<value>classpath:/property/rabbitmq.properties</value> </list>

4.添加applicationContext-rabbitmq-send

4.修改OrderServiceImpl从spring框架取出rabbitmqTemplate,发数据



@Autowired

RabbitTemplate rabbitTemplate;

@Override

**public** String saveOrder(Order order) {

//拼接orderId号

String orderId = order.getUserId() + "" + System.*currentTimeMillis*();

order.setOrderId(orderId);

//原先直接入库，数据很多，入库很慢

//把订单发到消息队列

String routingKey="saveOrder";

//把order对象序列化，变成字符串，发到rabbitmq服务器上

rabbitTemplate.convertAndSend(routingKey, order);}

### 消息消费者

#### 修改消费者实现类OrderMsgConsumer

不实现接口，删除@override

saveOrder返回void

去掉@service,删除别的方法，

**public** **class** OrderMsgConsumer {

@Autowired

**private** OrderMapper orderMapper;

@Autowired

**private** OrderItemMapper orderItemMapper;

@Autowired

**private** OrderShippingMapper orderShippingMapper;

**public** **void** saveOrder(Order order) {

//拼接orderId号

//String orderId = order.getUserId() + "" + System.currentTimeMillis();

String orderId = order.getOrderId();

Date date = **new** Date();

//入库订单

order.setOrderId(orderId);

order.setStatus(1);

order.setCreated(date);

order.setUpdated(date);

orderMapper.insert(order);

System.***out***.println("订单入库成功!!!!!");

//获取订单物流信息

OrderShipping orderShipping = order.getOrderShipping();

orderShipping.setOrderId(orderId);

orderShipping.setCreated(date);

orderShipping.setUpdated(date);

orderShippingMapper.insert(orderShipping);

System.***out***.println("订单物流信息入库成功!!");

//实现订单商品入库

List<OrderItem> orderItemList = order.getOrderItems();

**for** (OrderItem orderItem : orderItemList) {

orderItem.setOrderId(orderId);

orderItem.setCreated(date);

orderItem.setUpdated(date);

orderItemMapper.insert(orderItem);

}

System.***out***.println("订单入数据库成功!!!");

//return orderId;

}

}

#### 

#### 定义消费者配置文件

<!-- 异步的线程池，线程池的最在数不能设定太小，不然<rabbit:listener/>/@RabbitListener太多的话，会出现发无法正常消费问题 -->

<task:executor id=*"taskExecutor"* pool-size=*"4-256"* queue-capacity=*"128"* />

<!-- 定义RabbitMQ的连接工厂 -->

<rabbit:connection-factory id=*"connectionFactory"*

host=*"${rabbit.ip}"* port=*"${rabbit.port}"* username=*"${rabbit.username}"* password=*"${rabbit.password}"*

virtual-host=*"${rabbit.vhost}"*

publisher-confirms=*"true"*

publisher-returns=*"true"*

channel-cache-size=*"5"*

executor=*"taskExecutor"*/>

<!-- MQ的管理，包括队列、交换器等 -->

<rabbit:admin connection-factory=*"connectionFactory"* />

<!-- 定义消息队列 -->

<rabbit:queue name=*"orderQueue"* auto-declare=*"true"*/>

<!-- 定义交换机，并且完成队列和交换机的绑定 -->

<rabbit:direct-exchange name=*"orderExchange"* auto-declare=*"true"*>

<rabbit:bindings>

<!-- 前台系统只接收商品更新的消息，key路由key -->

<rabbit:binding queue=*"orderQueue"* key=*"saveorder"*/>

</rabbit:bindings>

</rabbit:direct-exchange>

<!-- 定义监听 -->

<rabbit:listener-container connection-factory=*"connectionFactory"*>

<!-- 监听一个队列，当队列中有消息，就会自动触发类.方法，传递消息就作为方法的参数，根据方法声明的参数强转 -->

<rabbit:listener ref=*"rabbitMQService"* method=*"saveOrder"* queue-names=*"orderQueue"*/>

</rabbit:listener-container>

<bean id=*"rabbitMQService"* class=*"com.jt.order.service.RabbitMQService"*></bean>

### Jt-web中修改js

修改order.js,查找success.html 5777行

**if**(result.status == 200){

window.alert("订单提交成功")

location.href = "/order/success.html?id="+result.data;

}

修改order-cart.jsp 加version=1,让js 立刻生效

<script type="text/javascript" src="/js/order.js?version=3">

</script>

# 总结