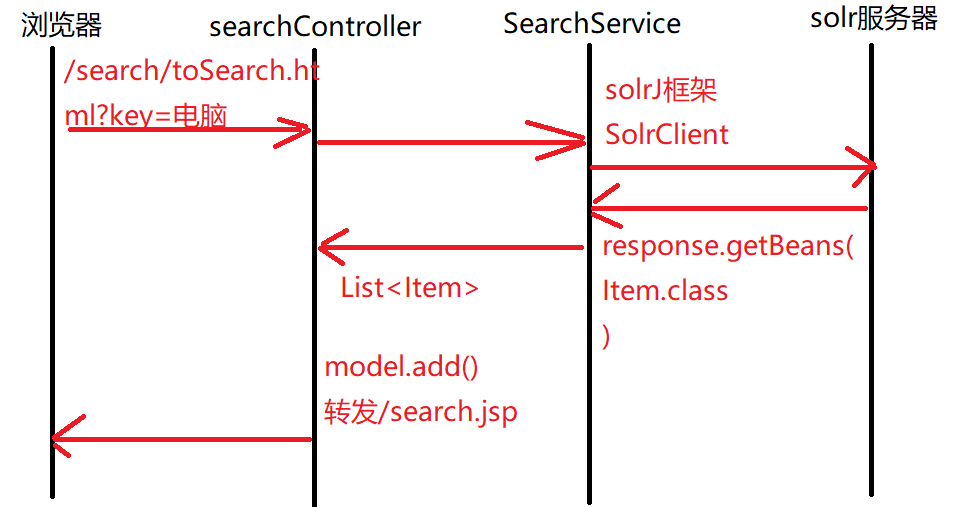
# 复习

产品有两个solr,es

Solr和es基于lucene,ikAnalyzer

连接solr服务器用的是solrJ框架 使用框架中的SolrClinet.query()



Solr比mysql查询速度快

Mysql查的时候比较每一行与key是否一致

文档表

|  |  |
| --- | --- |
| id | 文档 |
| 1 | Java是编程语言/1.html |
| 2 | Java开发工程师/2.html |

索引表

|  |  |
| --- | --- |
| 关键词 | 文档 |
| java | 1,2 |
| 编程 | 1 |

# 消息队列

## 消息队列作用

### 消息队列使用场景

#### 高并发(流量削锋)

应用场景：会因为流量过大，导致流量暴增，应用挂掉，如图- 55所示。

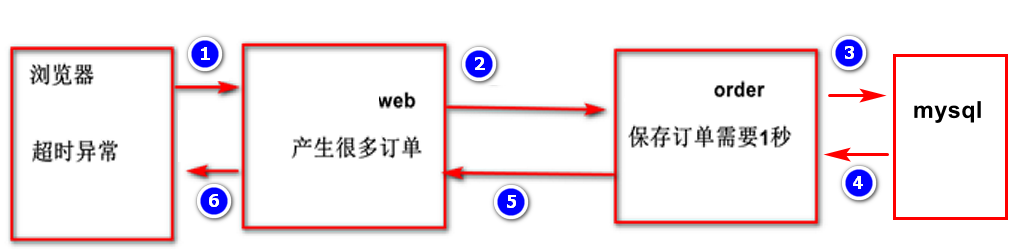


图- 55

为解决这个问题，一般需要在项目中加入消息队列，如图- 56所示。

消息提供者也叫消息发送者或消息生产者。

消息消费者也叫消息接收者。

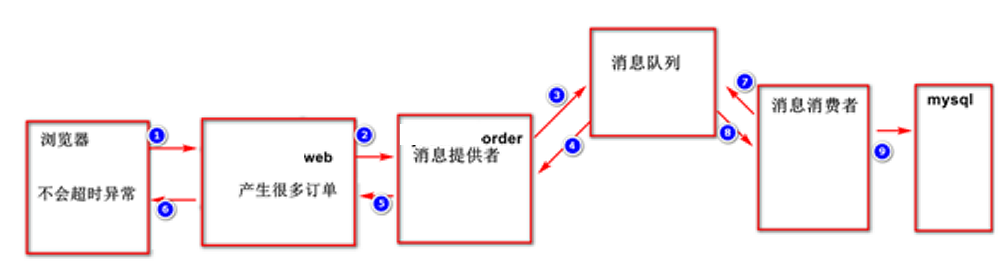
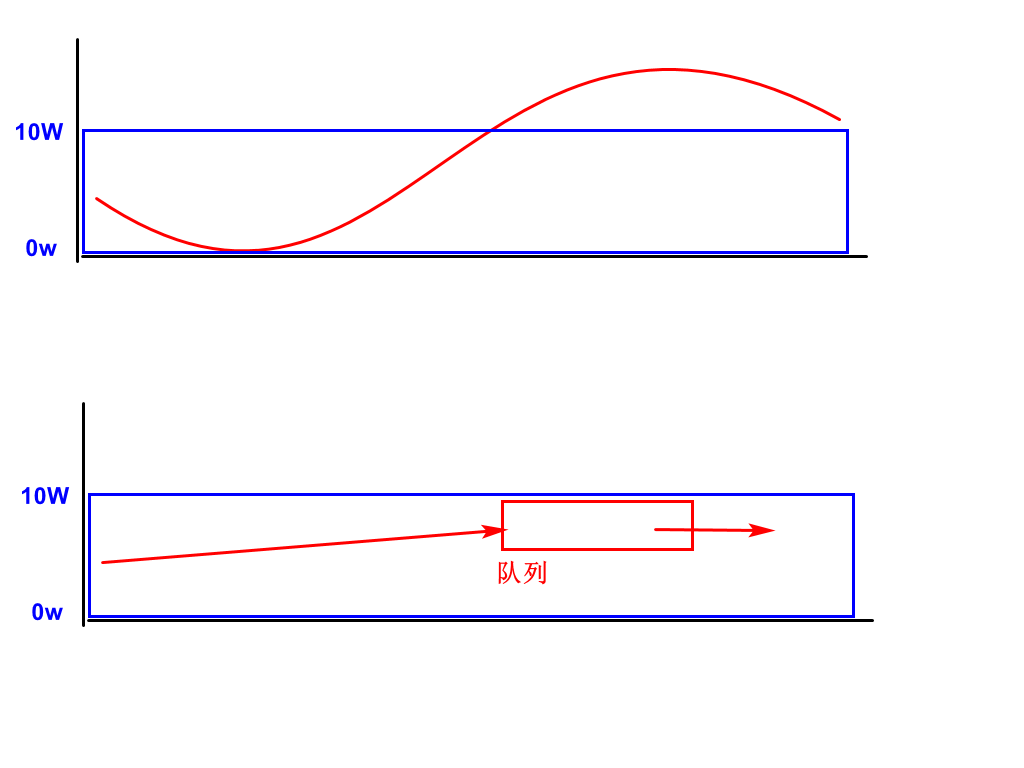


图- 56

* 用户的请求，服务器接收后，首先写入消息队列。
* 流量削峰也叫限流削峰或削峰填谷



#### 应用解耦

场景说明：用户下单后，web系统需要通知订单系统。传统的做法是，web系统调用订单系统的接口。如下图



图- 57

传统模式的缺点：

假如order无法访问，则用户下单失败。

web系统与order系统耦合高

如何解决以上问题呢？引入应用消息队列后的方案，如图-58 所示：

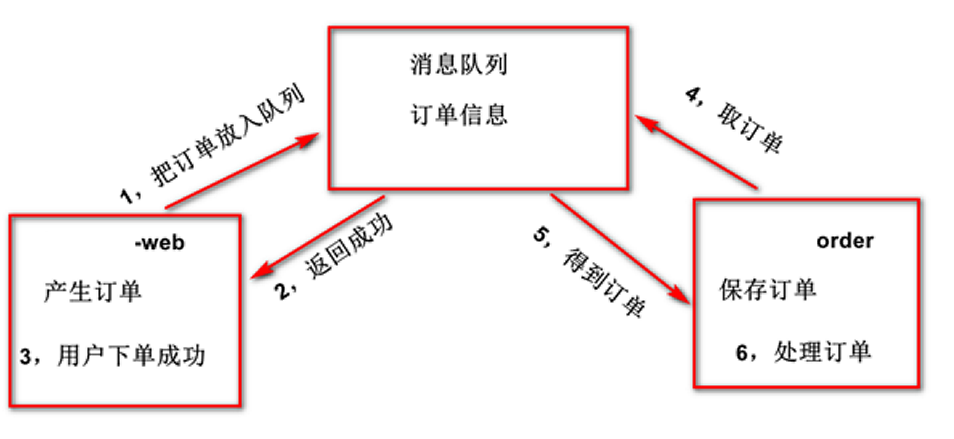


图- 58

提高了程序的可靠性

#### 异步处理

场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种 1.串行的方式；2.并行方式

（1）串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端，如图-59 所示



图- 59

（2）并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间，如图-60 所示



图- 60

假设三个业务节点每个使用50毫秒钟，不考虑网络等其他开销，则串行方式的时间是150毫秒，并行的时间可能是100毫秒。

如何解决这个问题呢？

引入消息队列，将不是必须的业务逻辑，异步处理。改造后的架构如图-61 所示：



图- 61

按照以上约定，用户的响应时间相当于是注册信息写入数据库的时间，也就是50毫秒。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是50毫秒。

### 消息队列

MQ全称为Message Queue, 消息队列（MQ）是一种应用程序对应用程序的通信方法。

消息队列是一个程序写入数据到队列中,另一个程序从队列中获取消息,并且通信通过RPC调用,可以传递java对象,方便操作.

### 缺点

增加了复杂度

网络出问题或消息队列服务器关机了，可靠性降低

### 消息队列的产品，如图-62 所示

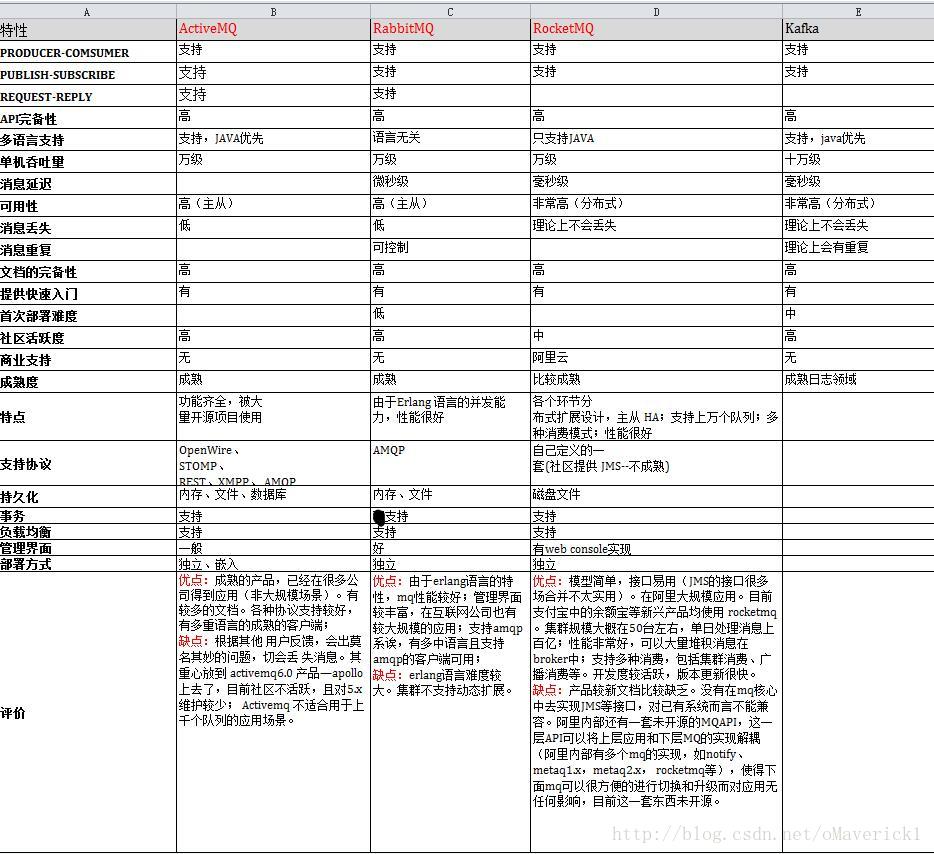


图- 62

### RabbitMQ特点

1. 高可用，能搭建集群服务器。

2.可扩展，有大量插件

3. 有消息确认机制

4.有持久化机制,可靠性高。

5. 开源

## RabbitMQ安装

### 上传文件

1.使用fz在/usr/local/src下创建文件夹rabbitmq

2.上传\亿发,移动端课前资料\rabbitMQ\ rabbitmq-server-3.6.1-1.noarch.rpm

到/usr/local/src/rabbitmq

### 安装rabbitMQ

执行rpm命令安装rabbitMQ

RPM是Red-Hat Package Manager（RPM软件包管理器）的缩写，Red-hat是一种Linux,RPM是用来安装Linux软件的。

cd rabbitmq

rpm -ivh rabbitmq-server-3.6.1-1.noarch.rpm

安装过程如图-63 所示

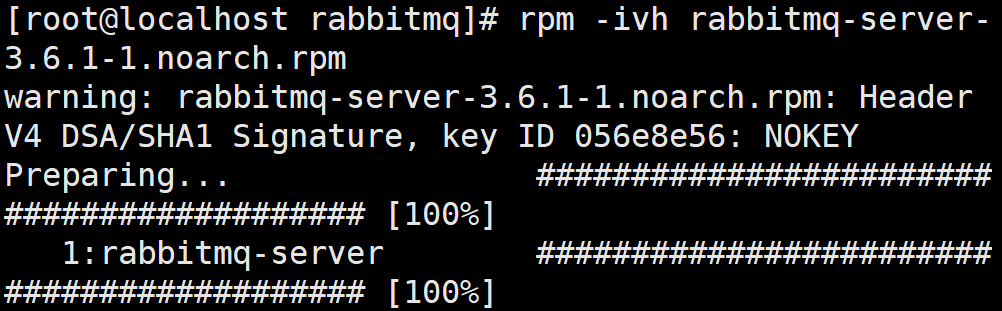


图- 63

### 安装rabbitMQ web后台管理

1.查看插件

cd /usr/lib/rabbitmq/lib/rabbitmq\_server-3.6.1/plugins

ls

#有management插件，这个插件是后台管理网页

rabbitmq\_management-3.6.1.ez

2.安装插件的命令

cd /usr/sbin

ls ra\*

rabbitmq-plugins

#用vim打开rabbitmq-plugins，是个脚本程序

vim rabbitmq-plugins

//下面命令安装后台管理插件，必须执行

[root@localhost sbin]# rabbitmq-plugins enable rabbitmq\_management

The following plugins have been enabled:

mochiweb

webmachine

rabbitmq\_web\_dispatch

amqp\_client

rabbitmq\_management\_agent

rabbitmq\_management

Applying plugin configuration to rabbit@localhost... failed.

\* Could not contact node rabbit@localhost.

Changes will take effect at broker restart.

\* Options: --online - fail if broker cannot be contacted.

--offline - do not try to contact broker.

### 启动rabbitMQ

# cd /usr/sbin

# ls ra\*

rabbitmq-server

service rabbitmq-server start

启动后如图- 64所示

计算机生成了可选文字:
[rootOlocalhost
rabbitmql逆
SeFVICe
rabbltmq一serverstart
Startlngrabbltmq一server:
rabbltmq一server.
[rootOlo。alhostrabbitmql#.

图- 64

### 添加用户，设置权限

[root@localhost sbin]# pwd

/usr/sbin

[root@localhost sbin]# ls ra\*

rabbitmqctl rabbitmq-plugins rabbitmq-server raid-check

//man查看命令的帮助文档

man ls

:q 就退出了

man cd

man pwd

//查看rabbitmqctl帮助文档

man rabbitmqctl

在最后一行直接 输入/add\_user查找add\_user字符串，如图-68 所示

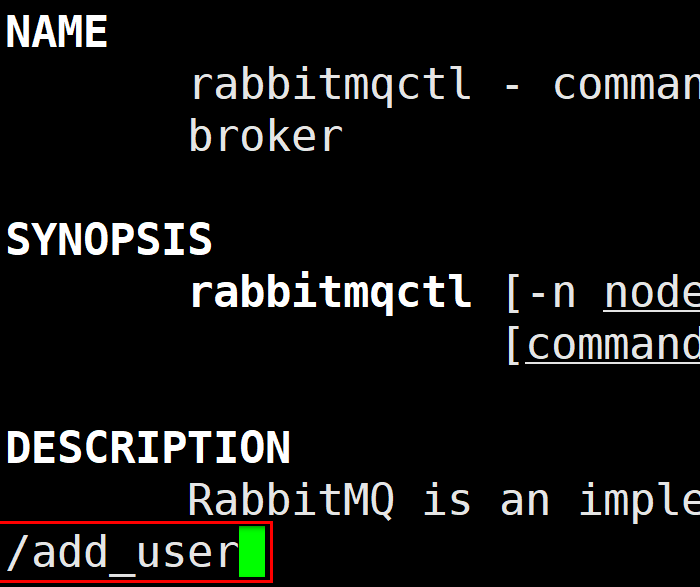


图- 68

会搜到下面的内容，如图-69 所示

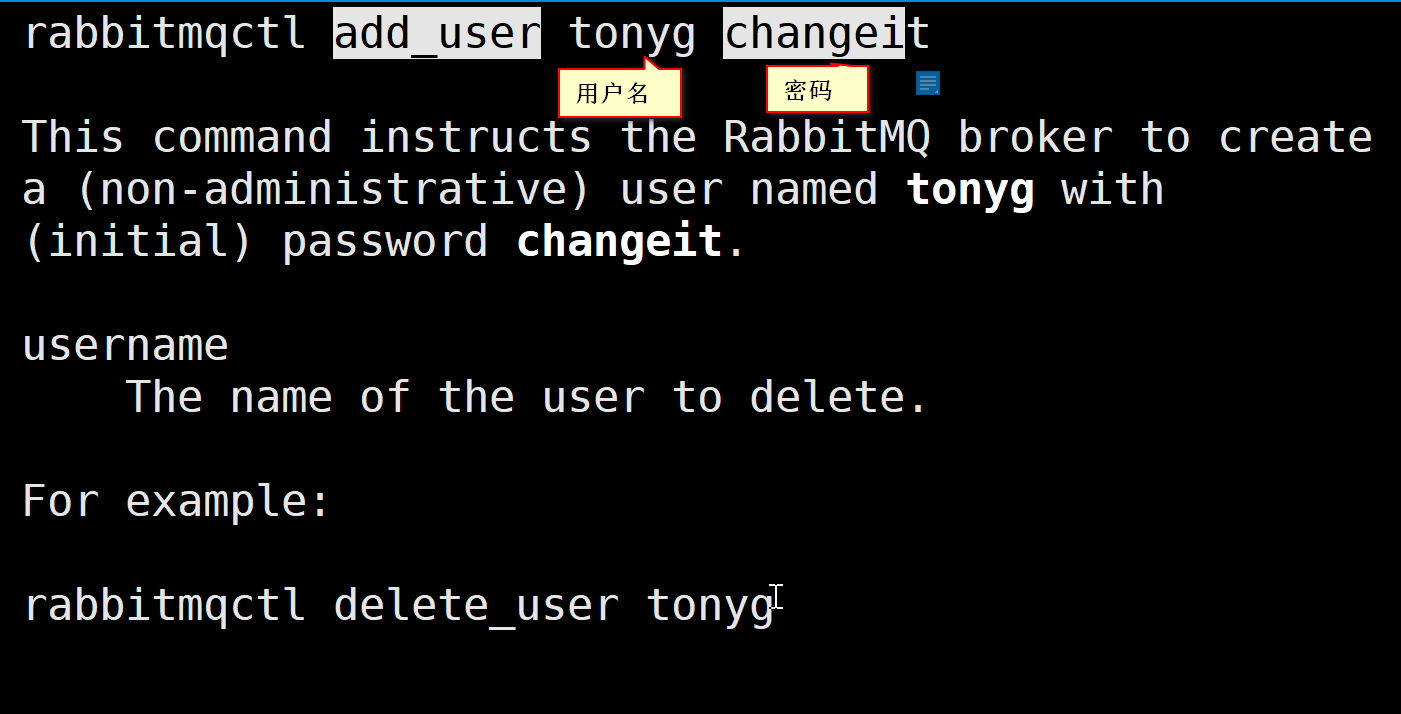


图- 69

进入命令模式输入q退出vi编辑器。

执行下面的命令创建一个用户 用户名是pdadmin,密码是pdadmin

rabbitmqctl add\_user pdadmin pdadmin

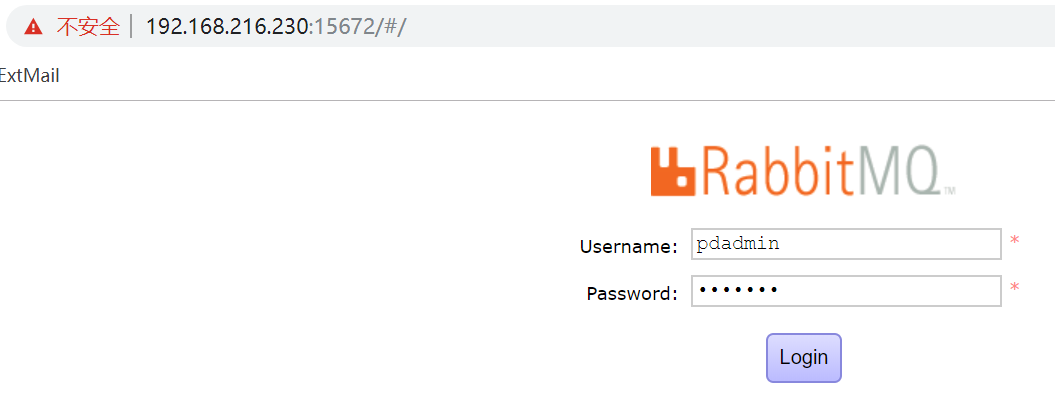
rabbitmqctl add\_user jtadmin jtadmin

执行下面的命令设置pdadmin是超级管理员。

rabbitmqctl set\_user\_tags pdadmin administrator

rabbitmqctl set\_user\_tags jtadmin administrator

在windows上用http://虚拟机ip:15672能看到登录界面，用pdadmin登录。



## 消息队列监控介绍

### 概念介绍

Virtualhost exchange queue channel之间的关系，如图-70 所示

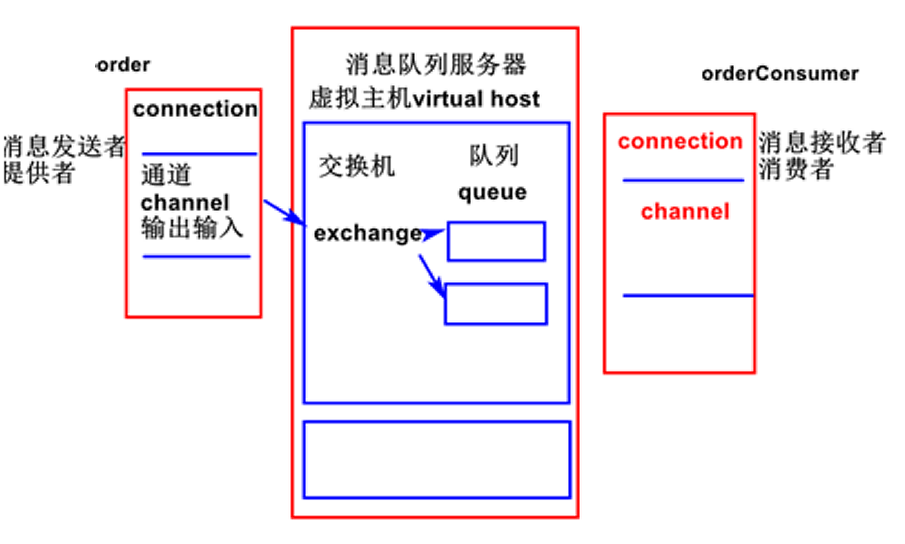


图- 70

virtualHost:设置权限，一个项目对应一个virtrualHost，相当于一个数据库

Exchanges: 交换机 作用:将消息发往不同的队列中

Queues:表示消息队列 存储数据的位置是内存.

### 角色介绍

|  |  |  |
| --- | --- | --- |
| **序号** | **角色名称** | **说明** |
| 1. | 超级管理员（administrator） | 可登陆管理控制台，可查看所有的信息，并且可以对用户，策略（policy）进行操作。 |
| 2. | 监控者（monitoring） | 可登陆管理控制台，同时可以查看rabbingmq节点的相关信息（进程数，内存使用情况，磁盘使用情况等） |
| 3. | 策略制定者（policymaker） | 可登陆管理控制台，同时可以对policy进行管理。但无法查看节点的相关信息 |
| 4. | 普通管理者（management） | 仅可登陆管理控制台，无法看到节点信息，也无法对策略进行管理。 |
| 5. | 其他 | 无法登录管理控制台，通常就是普通的生产者和消费者。 |

### 新增虚拟主机，如图-71 所示

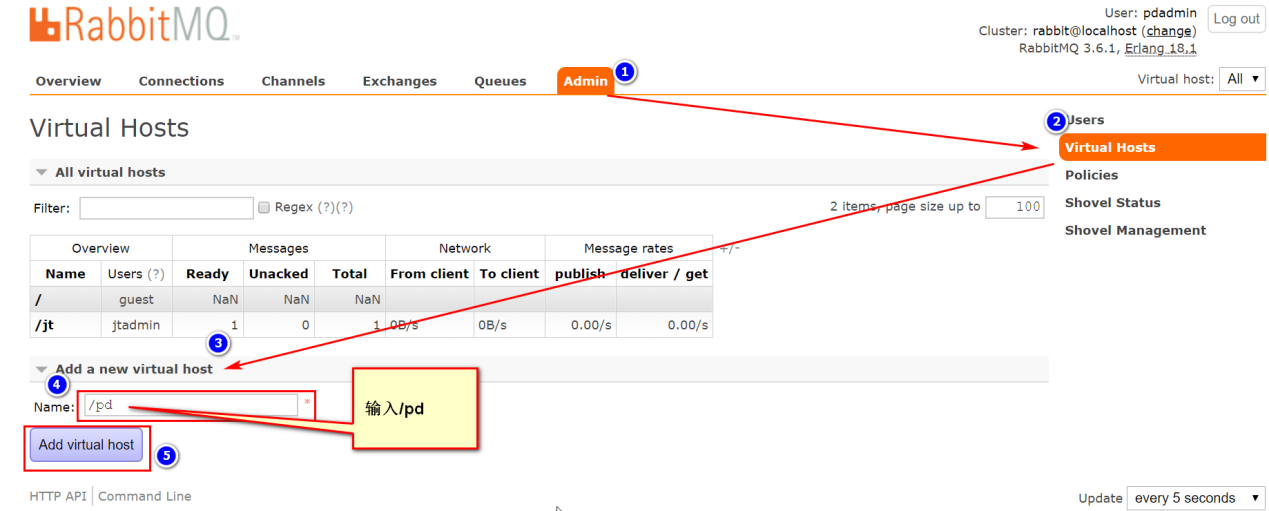


图- 71

### 设置用户pdadmin能管理虚拟主机pd

选中/pd 单击超连接，在权限管理界面中,设置pdadmin能管理pd，如图-72 所示



图- 72

设置成功后，如图-73 所示。

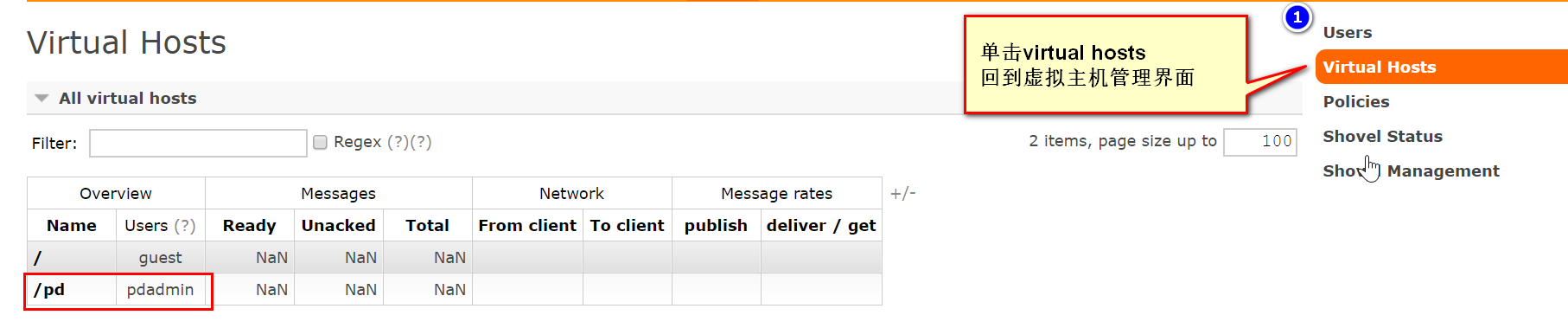


图- 73

### 创建jt虚拟主机,并设置权限

# 消息队列模式

## 简单模式

### 需求，如图-74 所示

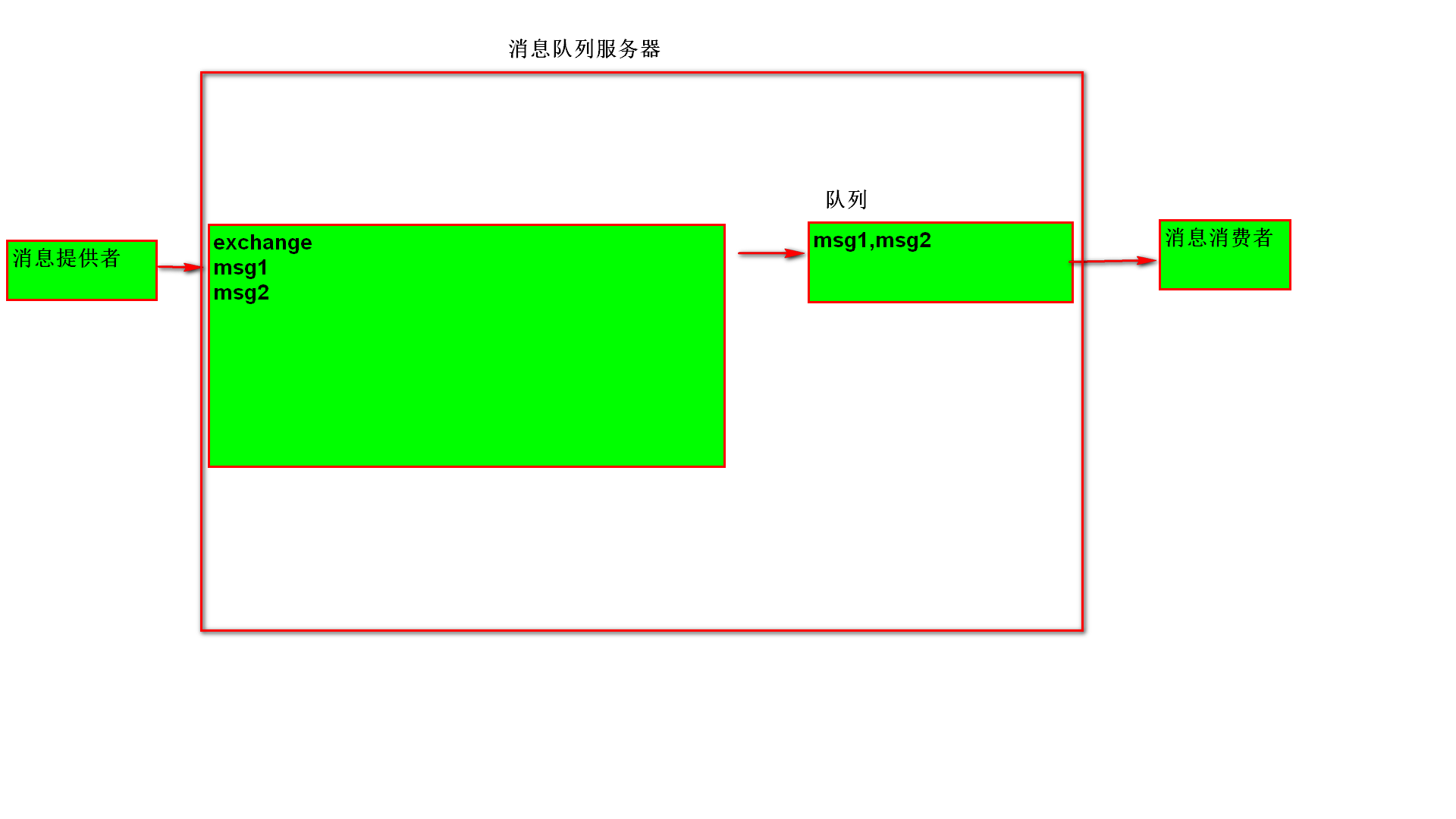


图- 74



图- 75

P:消息的生产者，如图-75 所示

红色部分:代表队列

C:代表消费者 从队列中获取消息后执行.

工作原理:

当客户端(生产者)将消息写入消息队列中时,消息队列中信息的数量加1.

消费者**实时监听**消息队列,当消息队列中有消息时,则获取消息,之后执行业务逻辑.同时消息队列的数量减一

### 实现生产者

#### 创建maven project，项目类型是quickstart

添加依赖

<!-- 消息队列 -->

<dependency>

<groupId>com.rabbitmq</groupId>

<artifactId>amqp-client</artifactId>

<version>3.5.1</version>

</dependency>

#### 代码实现

发送数据的过程与jdbc连接相似，共5步

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

@Test

**public** **void** provider()**throws** Exception{

//1,建立连接

//1.1得到factory

ConnectionFactory factory=**new** ConnectionFactory();

//1.2设置服务器ip

factory.setHost("192.168.216.230");

//1.3设置端口号

//rabbitmq服务器有两个端口号

//15672返回网页

//5672发消息，收消息

factory.setPort(5672);

//1.4设置虚拟主机

factory.setVirtualHost("/pd");

//1.5设置用户名和密码

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

//2,得到channel

//包名是com.rabbitmq.client.

Connection connection=factory.newConnection();

//包名是com.rabbitmq.client.

Channel channel=connection.createChannel();

//3,定义队列

String queueName="orderQueue1";

//false:队列不保存到硬盘

**boolean** durable=**false**;

//true:别的程序不能访问这个队列

//false:别的程序能访问这个队列

**boolean** exclusive=**false**;

//不要删除这个队列

**boolean** autoDelete=**false**;

HashMap<String, Object> arguments=**null**;

channel.queueDeclare(queueName,

durable,

exclusive,

autoDelete,

arguments);

//4,发消息

//交换机名称 ，值为"",使用的是默认交换机

String exchange="";

String routingKey=queueName;

//amqp包

BasicProperties props=**null**;

channel.basicPublish(exchange,

routingKey,

props,

"order1".getBytes());

//5,关闭连接

connection.close();

}

}

往order1Queue发送消息msg1

往order2Queue发送消息msg1,msg2

### 查看消息总数，如图-76 所示

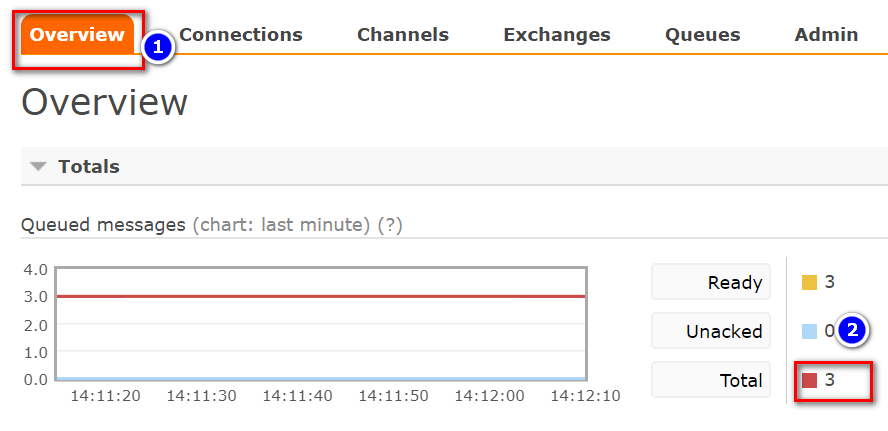


图- 76

### 查看队列，如图- 77所示

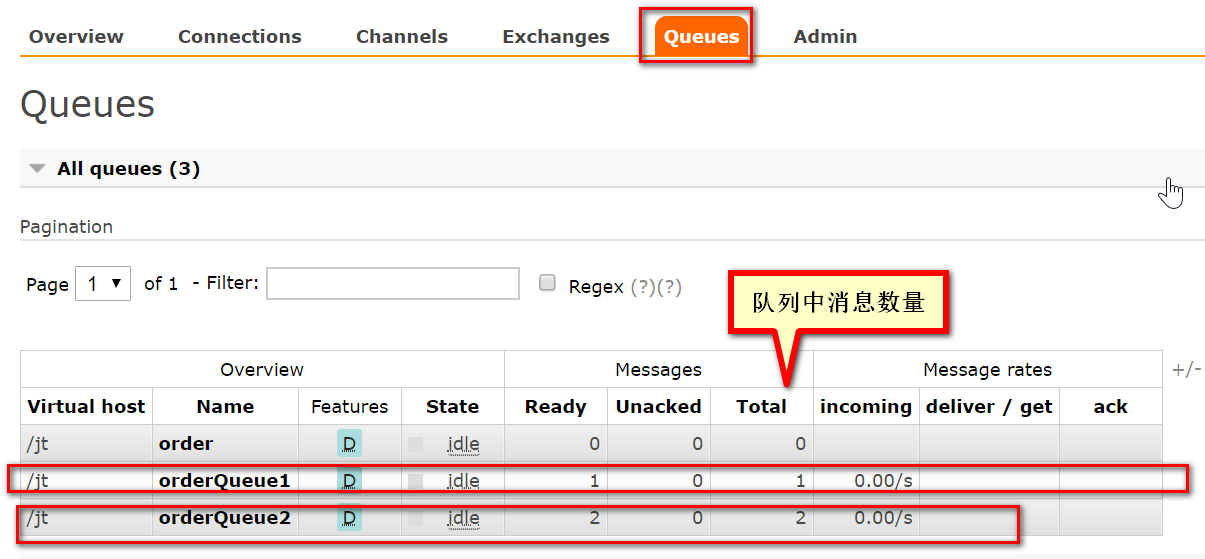


图- 77

### 查看每个队列的数据，如图- 78所示

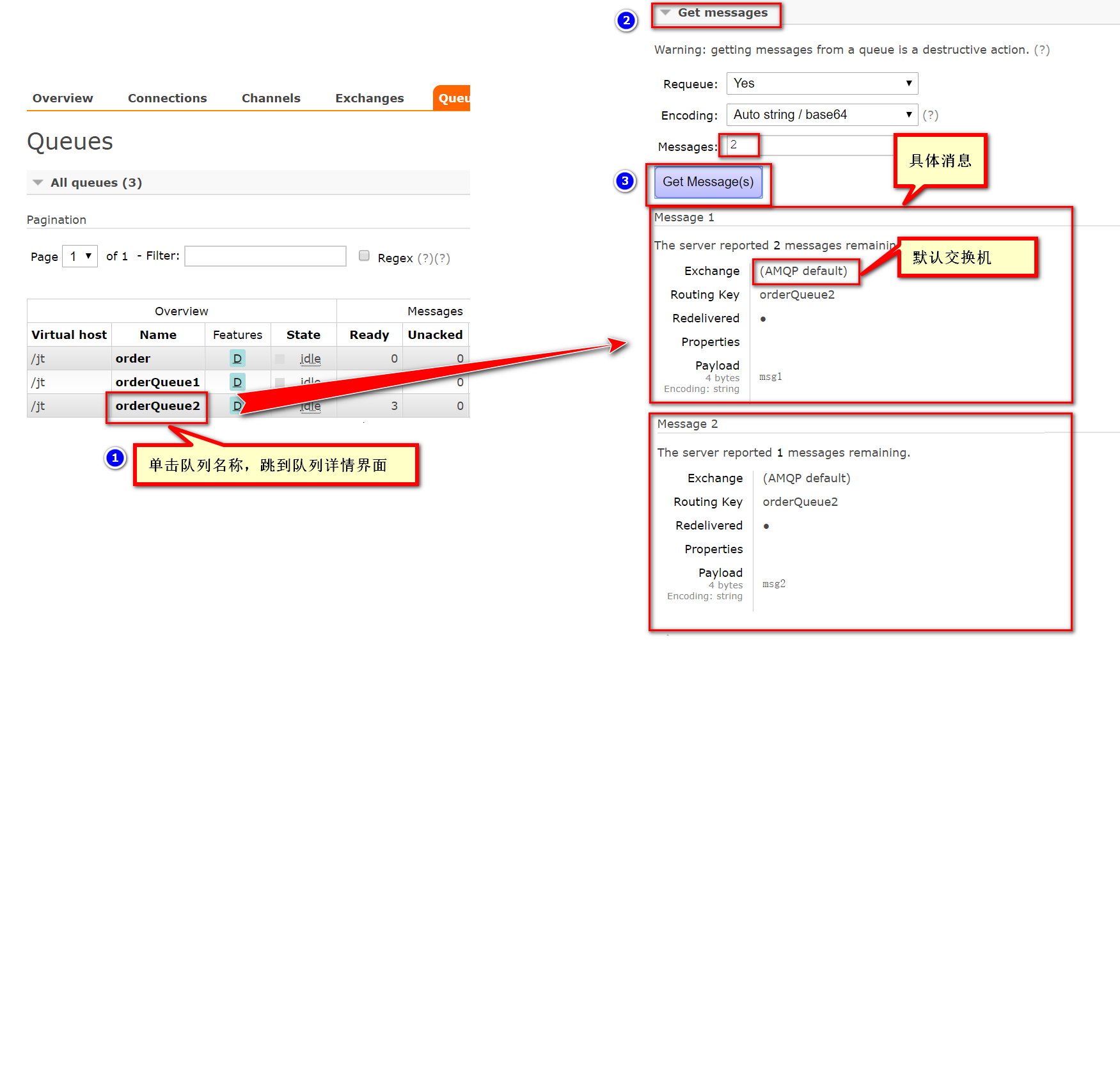


图- 78

### 查看连接

代码加上断点，发送消息后，连接还在

运行代码后，在监控中查看连接，如图- 79所示。

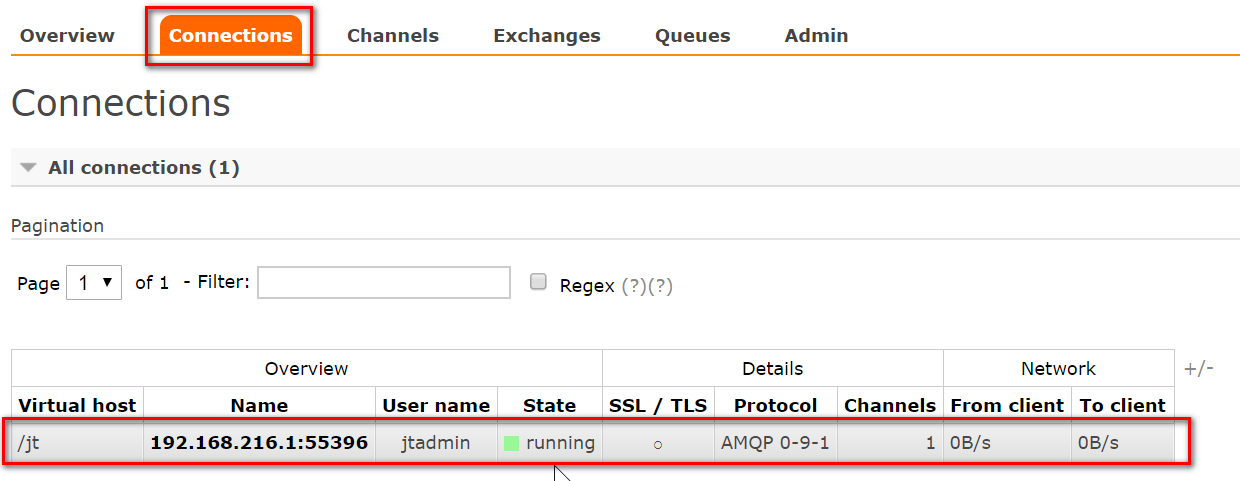


图- 79

去掉while，客户端运行完连接就断开了，在监控中查看不到连接。

//测试10秒钟能发送多少数据

**long** startTime=System.*currentTimeMillis*();

String exchange="";

String routingKey=queueName;

//amqp包

BasicProperties props=**null**;

**int** count=0;

**while**((System.*currentTimeMillis*()

-startTime)<10000)

{

count++;

channel.basicPublish(exchange,

routingKey,

props,

("order"+count).getBytes());

}

System.***out***.println("每秒发送"+count/10);

### 实现消费者

接收数据的过程与jdbc查询相似，共5步

//消息消费者（消息接收者）：从队列取消息，处理消息

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

@Test

**public** **void** consumer() **throws** Exception

{

//1,创建连接

ConnectionFactory factory=

**new** ConnectionFactory();

factory.setHost("192.168.216.200");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

Connection connection=

factory.newConnection();

//2,得到通道channel

Channel channel=connection.createChannel();

//申明队列

String queueName="orderQueue2";

**boolean** durable=**false**;

**boolean** exclusive=**false**;

**boolean** autoDelete=**false**;

HashMap<String, Object> arguments=**null**;

//如果先启提供者，队列存在，不创建队列

//不写也可以

//如果先启消费者，队列不在，就创建队列

channel.queueDeclare(queueName,

durable,

exclusive,

autoDelete,

arguments);

//3,创建consumer

//consumer是通过channel连上服务器的，

QueueingConsumer consumer=

**new** QueueingConsumer(channel);

//4,通过channel把consumer和队列连起来

//自动确认 true:

//消费者收到消息后,自动给服务器发个消息

//告诉服务器，我收到了，服务器就删除消息

**boolean** autoAck=**true**;

channel.basicConsume(queueName,

autoAck, consumer);

//5,遍历consumer,consumer相当于jdbc中的结果集

**int** msgCount=221501;

//5,遍历

**int** count=0;

//关闭的时候，把isRunning赋值成false

**boolean** isRunning=**true**;

**long** startTime=System.*currentTimeMillis*();

**while** (isRunning){

//delivery封装了消息和消息id

Delivery delivery=consumer.nextDelivery();

count++;

**if** (count==221501){

//取到最后一个消息

**byte**[] body=delivery.getBody();

String mString=**new** String(body);

System.***out***.println(mString);

**long** endTime=System.*currentTimeMillis*();

**long** useTime=endTime-startTime;

System.***out***.println(useTime/1000);

}

}

}

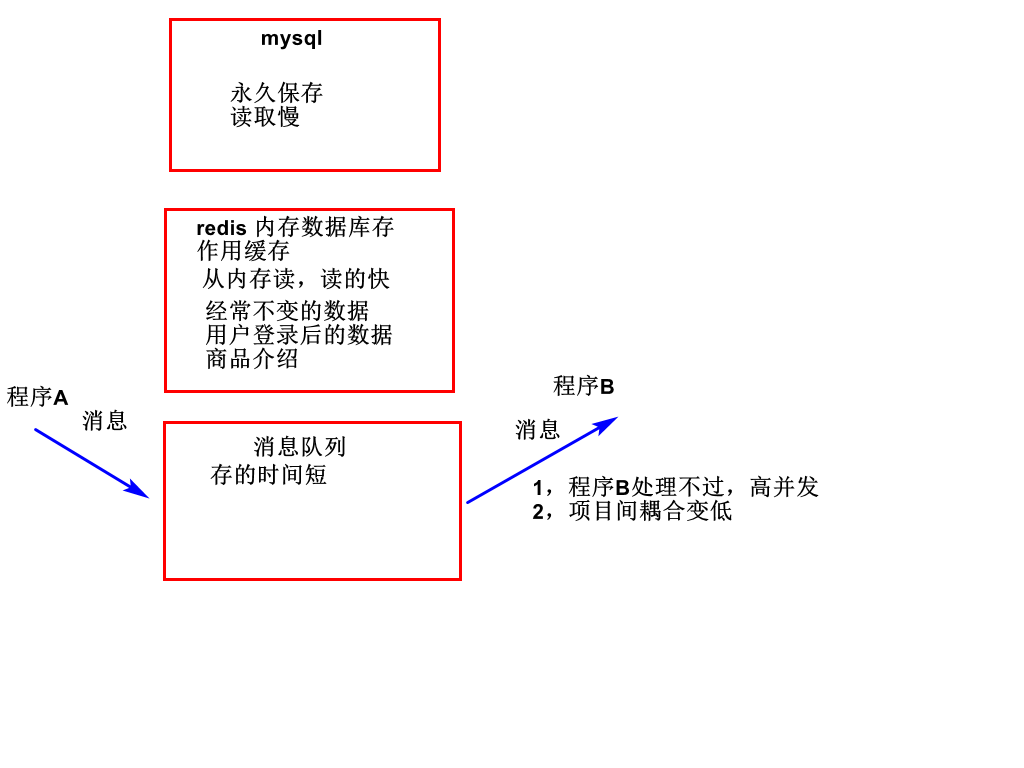
}

查看监控中消息被处理了

工作中，消息提供者和消息消费者运行在不同服务器上

Delivery的处理有点像resultSet和迭代器

### rabbitmq与 redis,mysql对比



消息队列保证数据一定被处理，提高可靠性。

## 数据持久化

当用户请求很多时，消费者处理不过来，信息先存在队列中。

当消费者没启动时，信息先存在队列中。

如果消息队列服务器关机重启，队列中的数据就没有了

### 关闭消费者，发送消息到队列中

队列中未处理消息，如图- 80所示

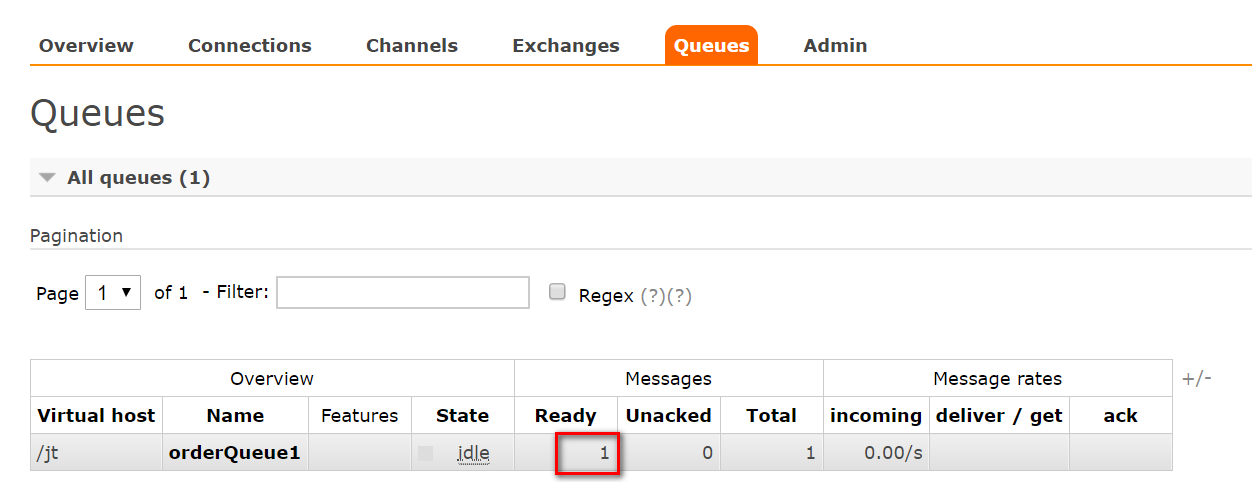


图- 80

### 重启服务器

cd /usr/sbin

service rabbitmq-server stop

service rabbitmq-server start

查看服务器队列没有了

### 队列持久化

//使用新队列

String queueName="orderQueue3";

//p2:durable true 持久化，保存到硬盘

**boolean** durable=**true**;

发送数据，查看后台有队列，有数据

重启服务器后，队列还存在，但消息没有了

### 消息持久化

在提供者代码中创建Properties设置deliveryMode为2，实现消息持久化。

//BasicProperties properties=null;

//实现消息的持久化，保存到硬盘。

//看源码发现Builder是BasicPropeties的内部类

BasicProperties.Builder builder=new BasicProperties().builder();

//消息持久化

builder.deliveryMode(2);

BasicProperties properties=builder.build();

**byte**[] body="msg5".getBytes();

//p4:body是消息内容

channel.basicPublish(exchange, routingKey,

properties, body);

1.向队列发送测试数据，查看数据

2.重启rabbitMQ服务器

3.通过后台查看数据还在

4.启动消费者，如果消费者的队列属性与服务器上的队列属性不一致会报错，要修改消费者的队列，支持持久化。消费者的队列名与提供者的队列名一致，修改后重新启动消费者就能取到数据。

## 手动确认

rabbitMQ确认有两种方式

* 1. 自动确认，如图-81 所示

消费者中autoAck=true就是自动确认

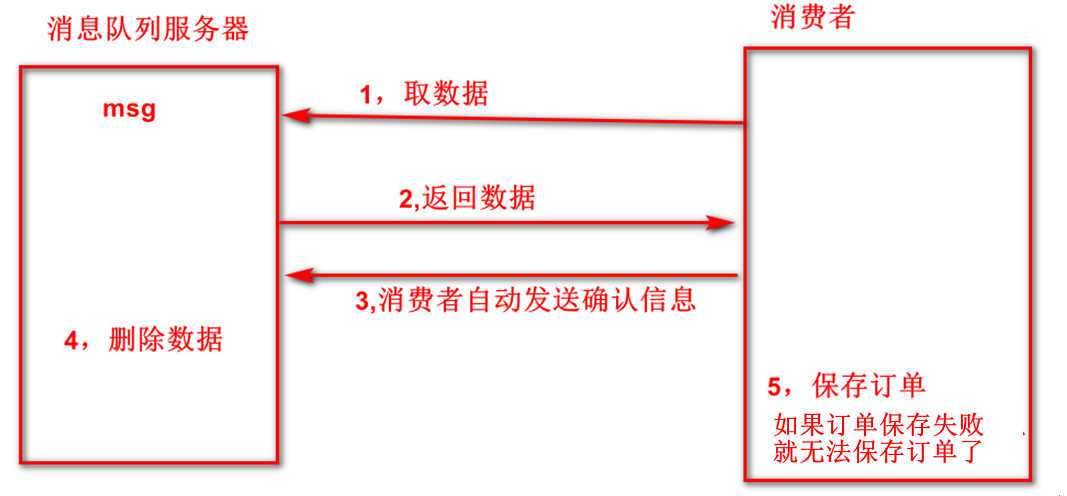


图- 81

* 1. 手动确认，如图-82 所示

autoAck=false;

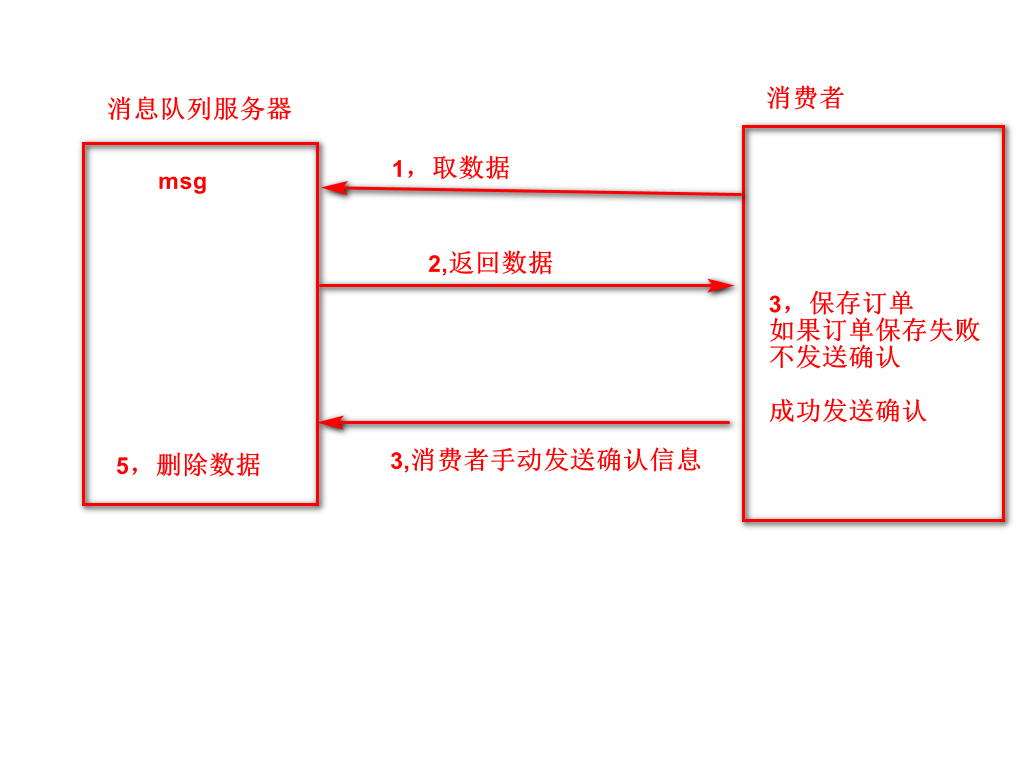


图- 82

拷贝Test\_1\_simple\_consumer，改名成Test\_1\_simple\_consumerACK

将autoAck置成false

关闭所有消费者

发送数据到队列

查看队列中的数据

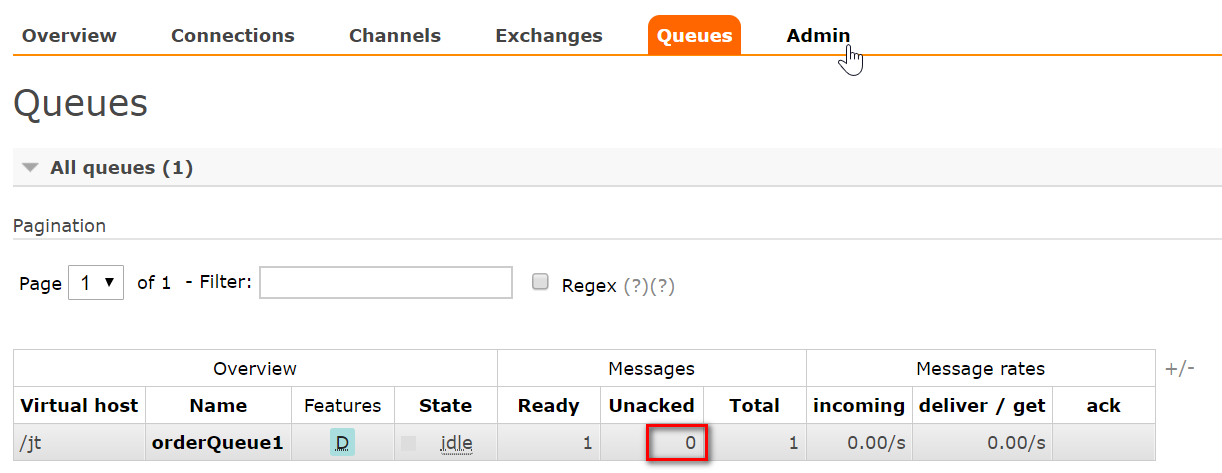


图- 83

模拟消费者收到数据，模拟处理失败。不发确认信息，查看队列有一个消息没有确认

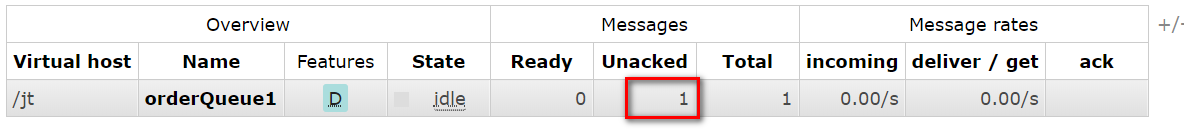


图- 84

添加发送确认信息代码

boolean autoAck=false;

channel.basicConsume(queueName, autoAck, consumer);

// 5,取消息

while (true) {

Delivery delivery = consumer.nextDelivery();

byte[] data = delivery.getBody();

String mString = new String(data);

System.*out*.println("消费者取到：" + mString);

try {

//数据处理正常，手动发确认

//如果前面数据处理的代码发生异常，不发确认

//数据还在服务器上

long deliveryTag=delivery.getEnvelope().getDeliveryTag();

System.*out*.println("deliveryTag="+deliveryTag);

channel.basicAck(deliveryTag, true);

System.out.println("发送确认信息");

} catch (Exception e) {

// TODO: handle exception

}

}

关闭消费者，再次启动消费者，处理成功，发送确认信息。

查看后台没有未确认消息，如图-85 所示

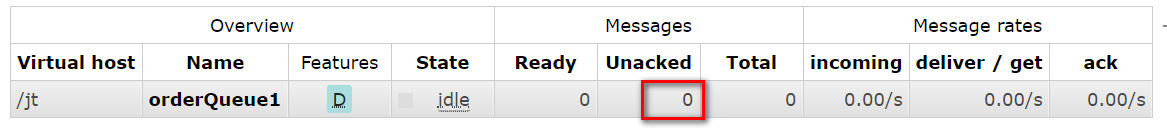


图- 85

## 工作模式

### 简单模式存在的问题

一个消费者能处理的消息态少，队列中消息如果很多，一个消费者处理不过来，需要多个消费者。

### 需求，如图- 86所示

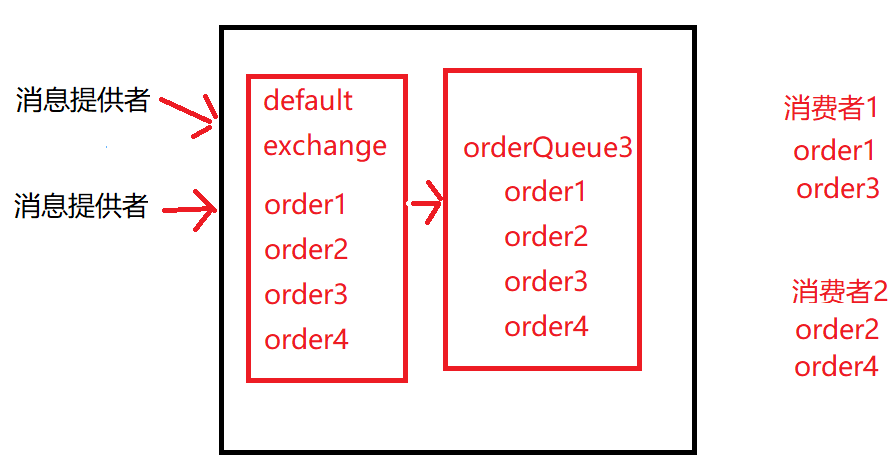


图- 86



图- 87

说明:

由一个生产者负责消息写入队列,但是如果有一个消费者负责消费,可能会造成消息的积压.所以准备多个消费者共同消费一个队列中的消息，如图- 87所示.

### 定义消费者1

拷贝简单模式的消费者代码，修改红色部分。

**public** **class** Test\_2\_work\_consumerAck1 {

// 从消息队列服务器取消息

@Test

**public** **void** consumer() **throws** Exception {

// 1.建立连接

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

// 3,创建队列

// 读消息时队列名称和写消息的队列名称必须一致

// 如果队列存在，不创建，服务器不会再创建新的了

// p1:队列名称

String queueName = "orderQueue5";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

// p5:arguments 队列的配置信息

HashMap<String, Object> argments = **null**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, argments);

// 4,创建消费者

QueueingConsumer consumer = **new** QueueingConsumer(channel);

// autoAck:自动回复消息

**boolean** autoAck=**false**;

channel.basicConsume(queueName, autoAck, consumer);

System.***out***.println("启动消费者1");

// 5,取消息

**while** (**true**) {

Delivery delivery = consumer.nextDelivery();

**byte**[] data = delivery.getBody();

String mString = **new** String(data);

System.***out***.println("消费者1取到：" + mString);

**try** {

**long** deliveryTag=delivery.getEnvelope().getDeliveryTag();

System.***out***.println("deliveryTag="+deliveryTag);

channel.basicAck(deliveryTag, **true**);

} **catch** (Exception e) {

// **TODO**: handle exception

}

}

// 6,连接关

// channel.close();

// connection.close();

// System.out.println("end");

}

}

### 拷贝消费者2

**public** **class** Test\_2\_work\_consumerAck2 {

// 从消息队列服务器取消息

@Test

**public** **void** consumer() **throws** Exception {

// 1.建立连接

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

// 3,创建队列

// 读消息时队列名称和写消息的队列名称必须一致

// 如果队列存在，不创建，服务器不会再创建新的了

// p1:队列名称

String queueName = "orderQueue5";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

// p5:arguments 队列的配置信息

HashMap<String, Object> argments = **null**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, argments);

// 4,创建消费者

QueueingConsumer consumer = **new** QueueingConsumer(channel);

// autoAck:自动回复消息

**boolean** autoAck=**false**;

channel.basicConsume(queueName, autoAck, consumer);

System.***out***.println("启动消费者2");

// 5,取消息

**while** (**true**) {

Delivery delivery = consumer.nextDelivery();

**byte**[] data = delivery.getBody();

String mString = **new** String(data);

System.***out***.println("消费者2取到：" + mString);

**try** {

**long** deliveryTag=delivery.getEnvelope().getDeliveryTag();

System.***out***.println("deliveryTag="+deliveryTag);

channel.basicAck(deliveryTag, **true**);

} **catch** (Exception e) {

// **TODO**: handle exception

}

}

// 6,连接关

// channel.close();

// connection.close();

// System.out.println("end");

}

}

### 定义生产者

拷贝简单模式提供者代码,加for循环

**public** **class** Test\_2\_work\_provider {

// 向消息队列写消息

@Test

**public** **void** provider() **throws** Exception {

// 1,连接服务器

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

// 浏览器访问rabbitmq后台管理用的是15672

// 发消息用的的5672

// 一个服务器可以有多个端口号，访问15672,服务器返回的是网页

// 访问5672,可以发消息，也可以取消息

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client.connection

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

// 3,定义队列

// p1:队列名称

String queueName = "orderQueue5";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

// p5:arguments 队列的配置信息

HashMap<String, Object> argments = **null**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, argments);

// 4,写消息

// p1:exchange 给"",使用的是default exchange

String exchange = "";

// p2:routing 路由 key关键字，决定消息放到那个队列

String routingKey = queueName;

// p3:props 是property

// com.rabbitmq.client.AMQP.BasicProperties

// BasicProperties properties=null;

AMQP.BasicProperties.Builder builder = **new** AMQP.BasicProperties.Builder();

// 消息持久化

builder.deliveryMode(2);

AMQP.BasicProperties properties = builder.build();

**for** (**int** i = 0; i < 10; i++) {

**byte**[] body = ("msg" + i).getBytes();

// p4:body是消息内容

channel.basicPublish(exchange, routingKey, properties, body);

}

// while(true){

//

// }

// 5,关闭连接

channel.close();

connection.close();

// System.out.println("发送了"+msg);

}

}

### 测试

修改队列名为新的名称“orderQueue5”

先启动消费者1,再启消费者2，消息全被第一个消费者处理了。

两个消费者都启动后，再启动提供者生产2个消息，分别被处理

## 发布订阅模式

### 需求，如图-88 所示

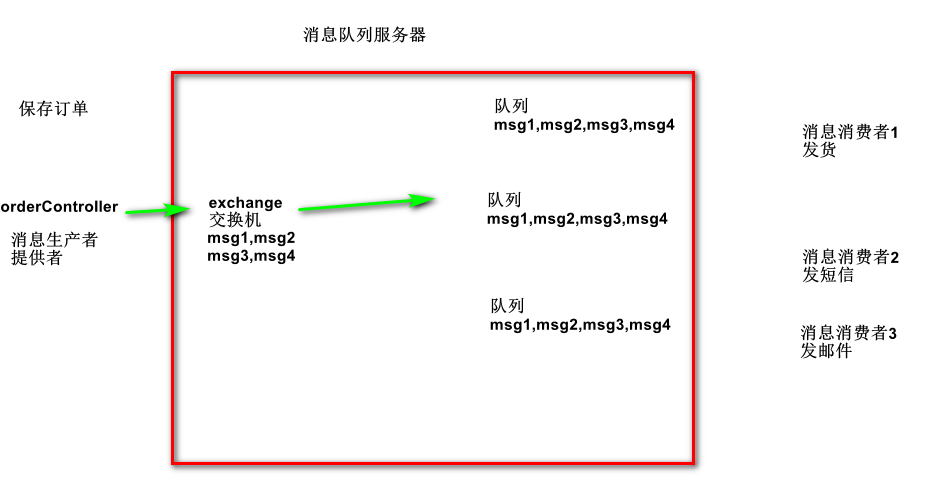


图- 88

每个消费者得到的消息是一样的。

### 生产者

代码与工作模式95%一样，导入“\亿发,移动端课前资料\rabbitMQ\测试代码”中的rabbitmqExchange

**public** **class** Test\_3\_publish\_provider {

// 向消息队列写消息

@Test

**public** **void** provider() **throws** Exception {

// 1,连接服务器

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

// 浏览器访问rabbitmq后台管理用的是15672

// 发消息用的的5672

// 一个服务器可以有多个端口号，访问15672,服务器返回的是网页

// 访问5672,可以发消息，也可以取消息

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client.connection

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

// 定义交换机

String exchangeName = "E1";

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "fanout");

// channel.queueDeclare("order", true, false,false, null);

// 4,写消息

**boolean** isRunning = **true**;

String msg = "msg 03";

String routingKey = "";

BasicProperties properties = **null**;

channel.basicPublish(exchangeName, routingKey, properties, msg.getBytes());

// 5,关闭连接

channel.close();

connection.close();

// System.out.println("发送了" + msg);

}

}

### 消费者1

**public** **class** Test\_3\_publish\_consumer1 {

// 从消息队列服务器取消息

@Test

**public** **void** consumer() **throws** Exception {

// 1.建立连接

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

String exchangeName = "E1";

// 创建交换机

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "fanout");

String queueName = "E1 queue1";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, **null**);

// 把队列和交换机绑定

channel.queueBind(queueName, exchangeName, "");

// 设置每次取几个数据

channel.basicQos(1);

// 4,得到消费者

// 创建的consumer通过channel来读数据

QueueingConsumer consumer = **new** QueueingConsumer(channel);

// 去取数据

// p2:autoack 开启手动确认

**boolean** autoAck = **false**;

channel.basicConsume(queueName, autoAck, consumer);

// 5,遍历消费者consumer

**boolean** isRunning = **true**;

System.***out***.println("消费者1启动了");

**while** (isRunning) {

// delivery代表的是消息队列中的一个数据

Delivery delivery = consumer.nextDelivery();

**byte**[] body = delivery.getBody();

String msg = **new** String(body);

System.***out***.println("消费者1收到：" + msg + "发货");

// 不发送确认信息，服务器上能看到队列中的消息

**long** deliveryTag = delivery.getEnvelope().getDeliveryTag();

channel.basicAck(deliveryTag, **true**);

}

// 6,连接关闭

channel.close();

connection.close();

}

}

### 消费者2

**public** **class** Test\_3\_publish\_consumer2 {

// 从消息队列服务器取消息

@Test

**public** **void** consumer() **throws** Exception {

// 1.建立连接

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

String exchangeName = "E1";

// 创建交换机

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "fanout");

String queueName = "E1 queue2";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, **null**);

// 把队列和交换机绑定

channel.queueBind(queueName, exchangeName, "");

// 设置每次取几个数据

channel.basicQos(1);

// 4,得到消费者

// 创建的consumer通过channel来读数据

QueueingConsumer consumer = **new** QueueingConsumer(channel);

// 去取数据

// p2:autoack 开启手动确认

**boolean** autoAck = **false**;

channel.basicConsume(queueName, autoAck, consumer);

// 5,遍历消费者consumer

**boolean** isRunning = **true**;

System.***out***.println("消费者2启动了");

**while** (isRunning) {

// delivery代表的是消息队列中的一个数据

Delivery delivery = consumer.nextDelivery();

**byte**[] body = delivery.getBody();

String msg = **new** String(body);

System.***out***.println("消费者2收到：" + msg + "发短信");

// 不发送确认信息，服务器上能看到队列中的消息

**long** deliveryTag = delivery.getEnvelope().getDeliveryTag();

channel.basicAck(deliveryTag, **true**);

}

// 6,连接关闭

channel.close();

connection.close();

}

}

### 查看每秒中有多少数据进来

先启动Test\_3\_publish\_provider不用启动消费者，用ctrl+F11频繁执行提供者，能看到每秒中有多少消息进到交换机，如图- 89所示.

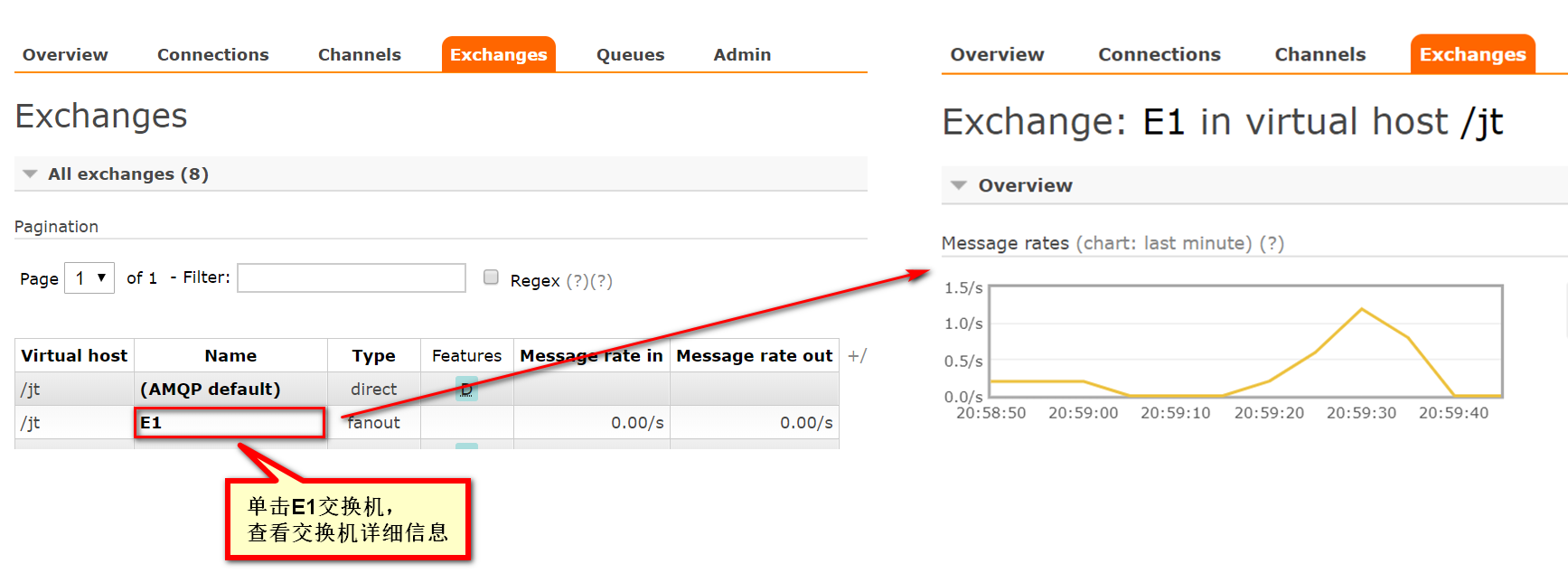


图- 89

在后台overview中看不到消息的数量，因为消息还没有进入队列。

### 查看队列

先启动两个消费者，

再启动提供者

查看队列，如图-90 所示

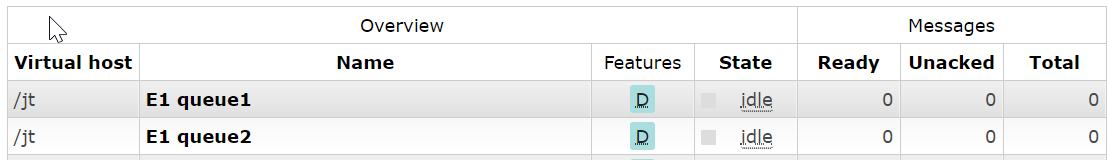


图- 90

测试结果每个消费者得到的消息是一样的

订阅来源于订阅报纸，每个用户拿到的报纸是一样的。

### 测试不发确认信息

注释掉消费者发送确认消息的代码

**while** (isRunning) {

// delivery代表的是消息队列中的一个数据

Delivery delivery = consumer.nextDelivery();

**byte**[] body = delivery.getBody();

String msg = **new** String(body);

System.***out***.println("消费者2收到：" + msg + "发货");

// 不发送确认信息，服务器上能看到队列中的消息

**long** deliveryTag = delivery.getEnvelope().getDeliveryTag();

//channel.basicAck(deliveryTag, true);

}

消费者收到消息后，如果不发确认信息，将收不到新的消息，直到返回确认信息给服务器为止。

rabbitMQ后台查不到未确认消息，能查到未发给消费者的消息。

如果队列中有很多消息，有可能是消费者处理失败，没有发确认信息。

### 案例

像广播。

案例：

1. 微信公众号，不同的人订阅同一个公众号，收到的消息是一样。
2. 不同的人加到同一个微信群后，收到的消息是一样的。

### 总结

订阅与工作模式区别是，订阅模式每个消费者得到的消息是一样的，工作模式每个消费者得到消息是不一样的。

解决高并发用工作模式

同一个消息，多个模块要处理用订阅

特点：

没有启动消费者，只运行提供者发消息 overview total中不显示消息的数量。Total只显示队列中消息的数量

每一次启动消费者，以前的消息的消息收不到。

再次启动消费者，以前的消息能收到。

测试项目时，要先启动消费者，再启动提供者

## 路由模式

### 需求，如图- 91所示

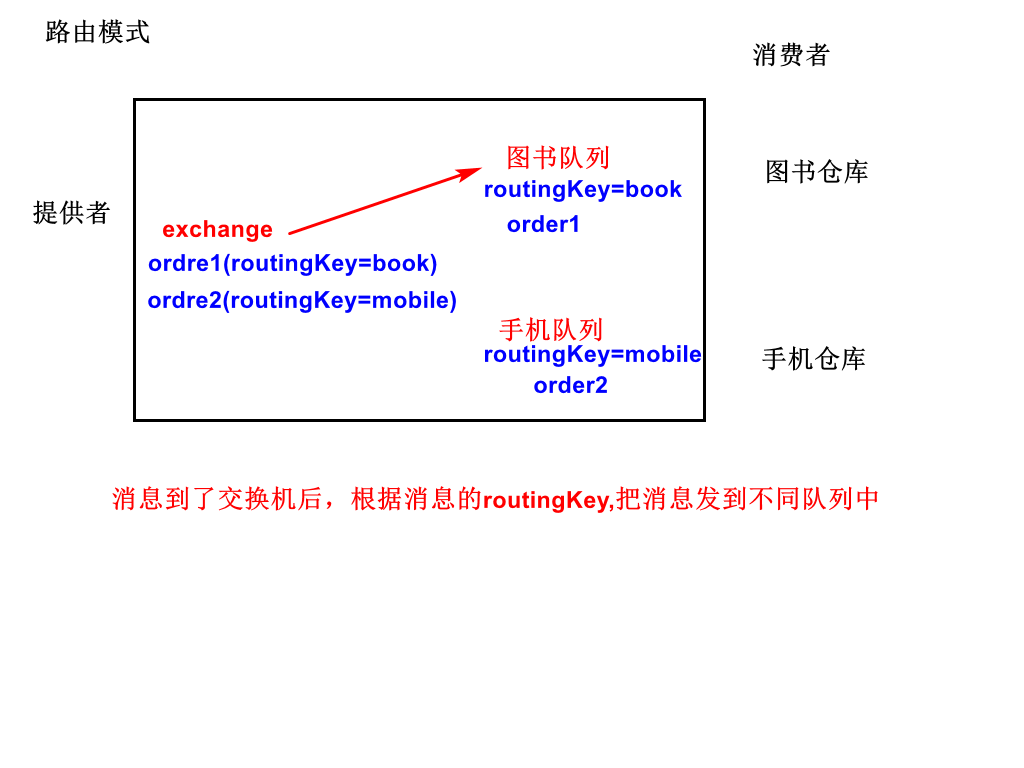


图- 91

说明:路由模式是发布订阅模式的升级,通过定义不同的路由key使得程序将消息发送到不同的队列中.

### 生产者

**public** **class** Test\_4\_direct\_provider {

// 向消息队列写消息

@Test

**public** **void** provider() **throws** Exception {

// 1,连接服务器

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

// 浏览器访问rabbitmq后台管理用的是15672

// 发消息用的的5672

// 一个服务器可以有多个端口号，访问15672,服务器返回的是网页

// 访问5672,可以发消息，也可以取消息

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("pd");

// 2,得到channel

// com.rabbitmq.client.connection

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

// 定义交换机

String exchangeName = "E2";

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "direct");

// channel.queueDeclare("order", true, false,false, null);

// 4,写消息

**boolean** isRunning = **true**;

String msg = "msg 01";

String routingKey = "mobile";

BasicProperties properties = **null**;

channel.basicPublish(exchangeName, routingKey, properties, msg.getBytes());

// 5,关闭连接

channel.close();

connection.close();

// System.out.println("发送了" + msg);

}

}

### 消费者1

**public** **class** Test\_4\_direct\_consumer1 {

// 从消息队列服务器取消息

@Test

**public** **void** consumer() **throws** Exception {

// 1.建立连接

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

String exchangeName = "E2";

// 创建交换机

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "direct");

String queueName = "E2 queue1";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, **null**);

// 把队列和交换机绑定

channel.queueBind(queueName, exchangeName, "mobile");

// 设置每次取几个数据

channel.basicQos(1);

// 4,得到消费者

// 创建的consumer通过channel来读数据

QueueingConsumer consumer = **new** QueueingConsumer(channel);

// 去取数据

// p2:autoack 开启手动确认

**boolean** autoAck = **false**;

channel.basicConsume(queueName, autoAck, consumer);

// 5,遍历消费者consumer

**boolean** isRunning = **true**;

System.***out***.println("消费者1启动了");

**while** (isRunning) {

// delivery代表的是消息队列中的一个数据

Delivery delivery = consumer.nextDelivery();

**byte**[] body = delivery.getBody();

String msg = **new** String(body);

System.***out***.println("消费者1收到：" + msg + "发手机");

// 不发送确认信息，服务器上能看到队列中的消息

**long** deliveryTag = delivery.getEnvelope().getDeliveryTag();

channel.basicAck(deliveryTag, **true**);

}

// 6,连接关闭

channel.close();

connection.close();

}

}

### 消费者2

**public** **class** Test\_4\_direct\_consumer2 {

// 从消息队列服务器取消息

@Test

**public** **void** consumer() **throws** Exception {

// 1.建立连接

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

String exchangeName = "E2";

// 创建交换机

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "direct");

String queueName = "E2 queue2";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, **null**);

// 把队列和交换机绑定

channel.queueBind(queueName, exchangeName, "book");

// 设置每次取几个数据

channel.basicQos(1);

// 4,得到消费者

// 创建的consumer通过channel来读数据

QueueingConsumer consumer = **new** QueueingConsumer(channel);

// 去取数据

// p2:autoack 开启手动确认

**boolean** autoAck = **false**;

channel.basicConsume(queueName, autoAck, consumer);

// 5,遍历消费者consumer

**boolean** isRunning = **true**;

System.***out***.println("消费者2启动了");

**while** (isRunning) {

// delivery代表的是消息队列中的一个数据

Delivery delivery = consumer.nextDelivery();

**byte**[] body = delivery.getBody();

String msg = **new** String(body);

System.***out***.println("消费者2收到：" + msg + "发书");

// 不发送确认信息，服务器上能看到队列中的消息

**long** deliveryTag = delivery.getEnvelope().getDeliveryTag();

channel.basicAck(deliveryTag, **true**);

}

// 6,连接关闭

channel.close();

connection.close();

}

}

启动消费者后，再用提供者发布消息，每个消费者收到消息不一样。

总结:消费者的队列只放有某个 routingkey的消息

相当于实现了select where routingkey=’mobile’功能.

订阅模式：每个消费者收到的消息都一样

路由模式：根据routingKey,每个消费者收到的消息不一样

工作模式：平均分。

## 主题模式

### 需求

不同消息由同一个队列处理，如图-92 所示。

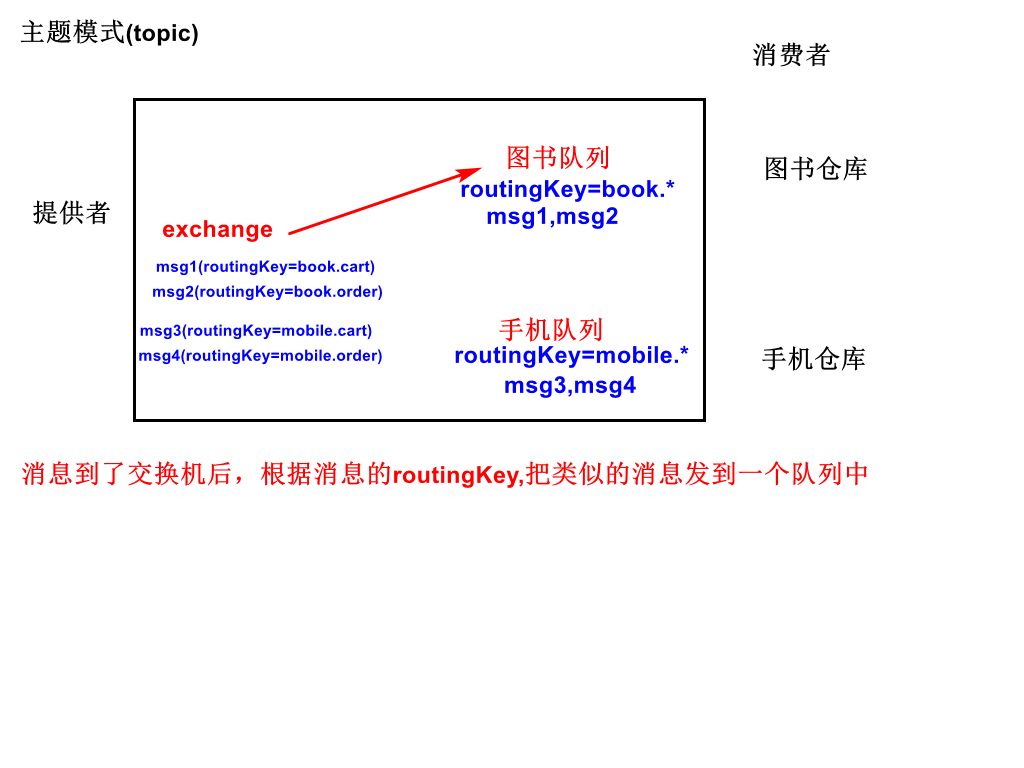


图- 92

说明:

可以通过路由key将消息发送到一类相同的key中 使用通配符实现

符号说明:

#号:表示任意字符(任意个.)

\*号:任意单个字符或者词组(单个.)

### 生产者

**public** **class** Test\_5\_topic\_provider {

// 向消息队列写消息

@Test

**public** **void** provider() **throws** Exception {

// 1,连接服务器

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

// 浏览器访问rabbitmq后台管理用的是15672

// 发消息用的的5672

// 一个服务器可以有多个端口号，访问15672,服务器返回的是网页

// 访问5672,可以发消息，也可以取消息

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client.connection

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

// 定义交换机

String exchangeName = "E3";

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "topic");

// channel.queueDeclare("order", true, false,false, null);

// 4,写消息

**boolean** isRunning = **true**;

String msg = "msg 01";

String routingKey = "mobile.cart";

BasicProperties properties = **null**;

channel.basicPublish(exchangeName, routingKey, properties, msg.getBytes());

// 5,关闭连接

channel.close();

connection.close();

// System.out.println("发送了" + msg);

}

}

### 消费者1

**public** **class** Test\_5\_topic\_consumer1 {

// 从消息队列服务器取消息

@Test

**public** **void** consumer() **throws** Exception {

// 1.建立连接

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

String exchangeName = "E3";

// 创建交换机

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "topic");

String queueName = "E3 queue1";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, **null**);

// 把队列和交换机绑定

channel.queueBind(queueName, exchangeName, "mobile.\*");

// 设置每次取几个数据

channel.basicQos(1);

// 4,得到消费者

// 创建的consumer通过channel来读数据

QueueingConsumer consumer = **new** QueueingConsumer(channel);

// 去取数据

// p2:autoack 开启手动确认

**boolean** autoAck = **false**;

channel.basicConsume(queueName, autoAck, consumer);

// 5,遍历消费者consumer

**boolean** isRunning = **true**;

System.***out***.println("消费者1启动了");

**while** (isRunning) {

// delivery代表的是消息队列中的一个数据

Delivery delivery = consumer.nextDelivery();

**byte**[] body = delivery.getBody();

String msg = **new** String(body);

System.***out***.println("消费者1 手机仓库收到：" + msg );

// 不发送确认信息，服务器上能看到队列中的消息

**long** deliveryTag = delivery.getEnvelope().getDeliveryTag();

channel.basicAck(deliveryTag, **true**);

}

// 6,连接关闭

channel.close();

connection.close();

}

}

### 消费者2

**public** **class** Test\_5\_topic\_consumer2 {

// 从消息队列服务器取消息

@Test

**public** **void** consumer() **throws** Exception {

// 1.建立连接

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("192.168.216.202");

factory.setPort(5672);

factory.setUsername("pdadmin");

factory.setPassword("pdadmin");

factory.setVirtualHost("/pd");

// 2,得到channel

// com.rabbitmq.client

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

String exchangeName = "E3";

// 创建交换机

// fanout 订阅模式

// direct 路由模式

// topic 主题模式

channel.exchangeDeclare(exchangeName, "topic");

String queueName = "E3 queue2";

// p2:durable true 持久化，保存到硬盘

**boolean** durable = **true**;

// p3:exclusive false 别的程序也能访问

**boolean** exclusive = **false**;

// p4:autoDelete false 队列中的消息处理完了，不删队列

**boolean** autoDelete = **false**;

channel.queueDeclare(queueName, durable, exclusive, autoDelete, **null**);

// 把队列和交换机绑定

channel.queueBind(queueName, exchangeName, "book.\*");

// 设置每次取几个数据

channel.basicQos(1);

// 4,得到消费者

// 创建的consumer通过channel来读数据

QueueingConsumer consumer = **new** QueueingConsumer(channel);

// 去取数据

// p2:autoack 开启手动确认

**boolean** autoAck = **false**;

channel.basicConsume(queueName, autoAck, consumer);

// 5,遍历消费者consumer

**boolean** isRunning = **true**;

System.***out***.println("消费者2启动了");

**while** (isRunning) {

// delivery代表的是消息队列中的一个数据

Delivery delivery = consumer.nextDelivery();

**byte**[] body = delivery.getBody();

String msg = **new** String(body);

System.***out***.println("消费者2 图书仓库收到：" + msg );

// 不发送确认信息，服务器上能看到队列中的消息

**long** deliveryTag = delivery.getEnvelope().getDeliveryTag();

channel.basicAck(deliveryTag, **true**);

}

// 6,连接关闭

channel.close();

connection.close();

}

}

### 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，平均分，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=mobile

主题topic:消息分类，消费者取消息时用mobile.\*能取到routingKey为mobile.cart和mobile.order的信息。

# 总结

## 为什么要用消息队列

1. 提高系统的可靠性
2. 流量削峰

消息进入队列后，马上被消息消费者处理

## 消息队列与mysql,redis的区别

Mysql中的数据永久保存，多张表，主外键关联，事务

用户量很大，要把经常查询的数据放在redis

消息队列是为了保存分布式中数据的可靠性。