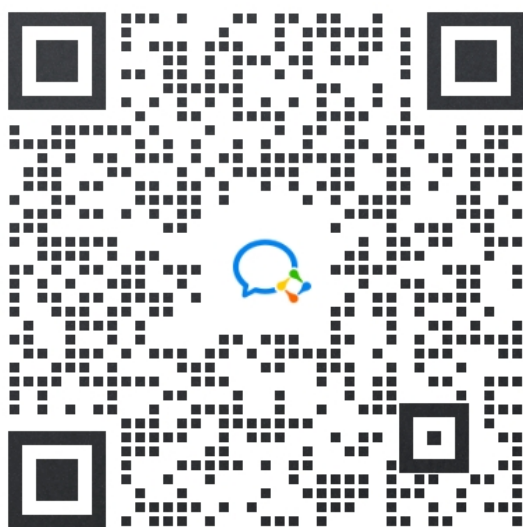


## 2. RabbitMQ快速上手

### 版权说明

本“比特就业课”课程（以下简称“本课程”）的所有内容，包括但不限于文字、图片、音频、视频、软件、程序、数据库、设计、布局、界面等，均由本课程的开发者或授权方拥有版权。我们鼓励个人学习者使用本课程进行学习和研究。在遵守相关法律法规的前提下，个人学习者可以下载、浏览、学习本课程的内容，并为了个人学习、研究或教学目的而使用其中的材料。但请注意，未经我们明确授权，个人学习者不得将本课程的内容用于任何商业目的，包括但不限于销售、转让、许可或以其他方式从中获利。此外，个人学习者也不得擅自修改、复制、传播、展示、表演或制作本课程内容的衍生作品。任何未经授权的使用均属侵权行为，我们将依法追究法律责任。如果您希望以其他方式使用本课程的内容，包括但不限于引用、转载、摘录、改编等，请事先与我们联系，获取书面授权。感谢您对“比特就业课”课程的关注与支持，我们将持续努力，为您提供更好的学习体验。特此说明。比特就业课版权所有方。

对比特课程感兴趣，可以联系这个微信。



### 1. RabbitMQ安装

我们对于RabbitMQ已经有了简单的了解,接下来我们来进行RabbitMQ的安装,并进行入门程序的演示,让大家对于RabbitMQ有一个更加直观的感受.

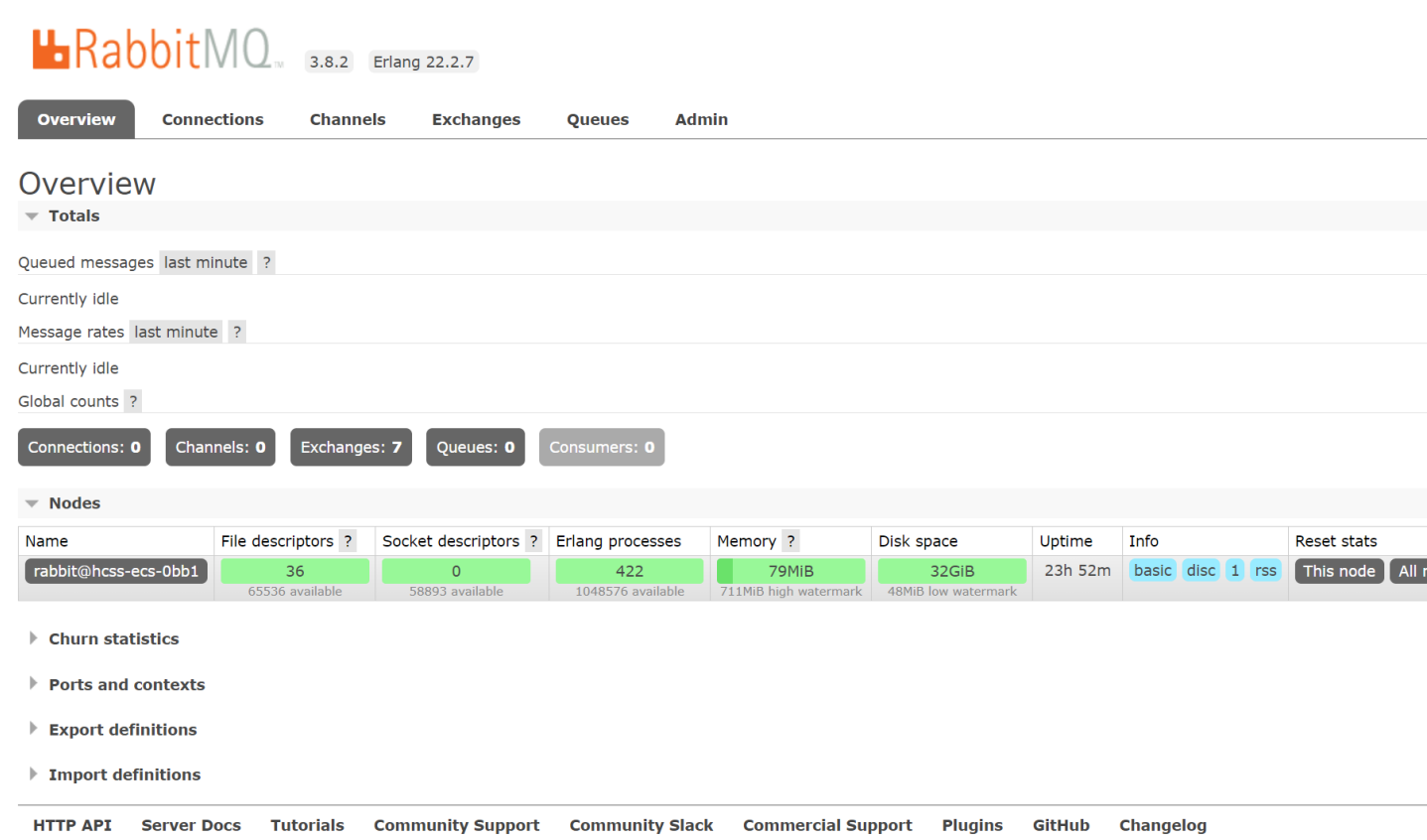
RabbitMQ 是一套开源的消息队列服务软件,基于 Erlang 语言编写,所以安装RabbitMQ之前,需要先安装部署 Erlang 环境,再安装 RabbitMQ 环境.

RabbitMQ大多部署在Linux操作系统,如未特别说明,本课程RabbitMQ都是在Linux环境下运行的.

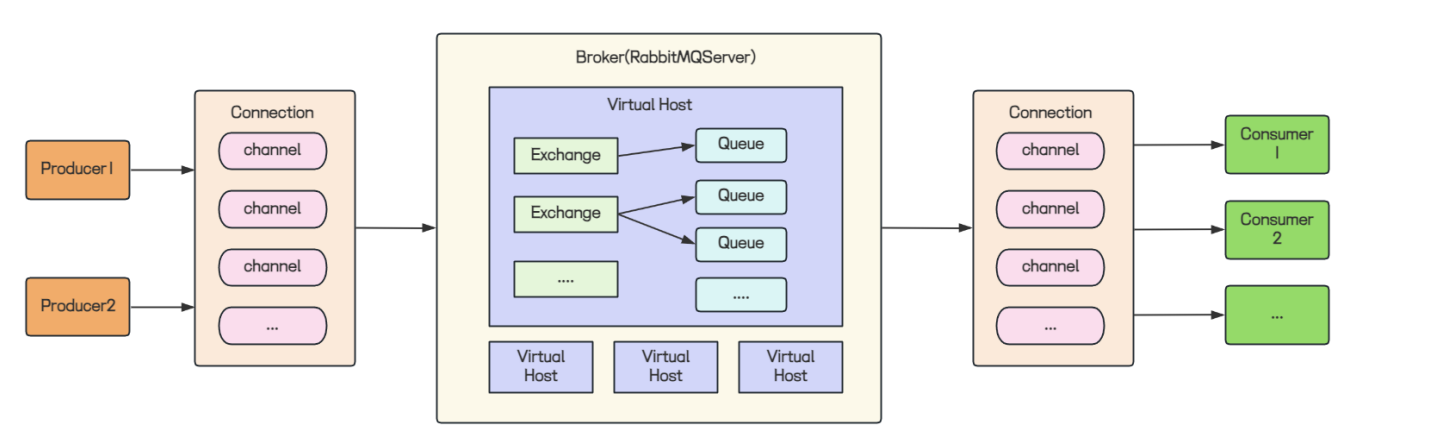
## 2. RabbitMQ 核心概念

在安装完RabbitMQ之后, 我们接下来学习如何去使用RabbitMQ

在上一个篇幅, 我们讲了RabbitMQ的安装, 并安装了管理界面



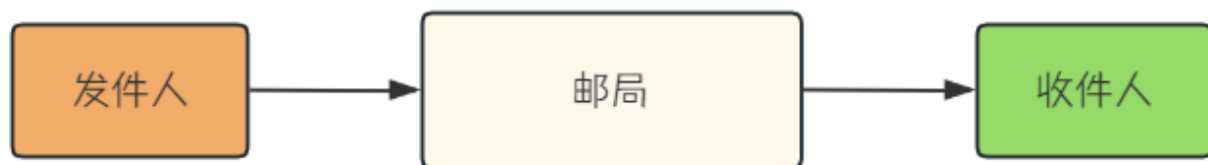
界面上的导航栏共分6部分, 这6部分分别是什么意思呢, 我们先看看RabbitMQ的工作流程



RabbitMQ是一个消息中间件, 也是一个生产者消费者模型. 它负责接收, 存储并转发消息.

消息传递的过程类似邮局。

当你要发送一个邮件时, 你把你的邮件放到邮局, 邮局接收邮件, 并通过邮递员送到收件人的手上。



按照这个逻辑, Producer 就类似邮件发件人. Consumer 就是收件人, RabbitMQ就类似于邮局

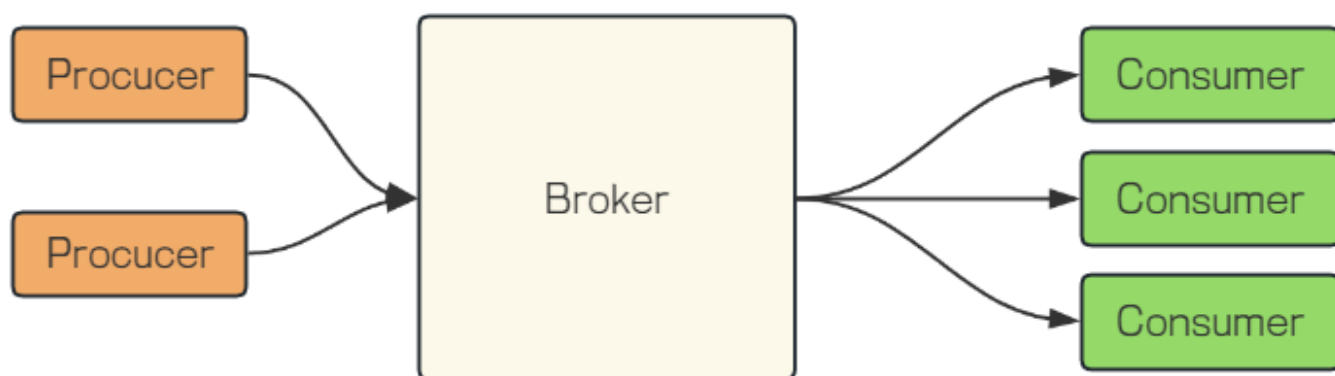
## 2.1 Producer和Consumer

**Producer:** 生产者, 是RabbitMQ Server的客户端, 向RabbitMQ发送消息

**Consumer:** 消费者, 也是RabbitMQ Server的客户端, 从RabbitMQ接收消息

**Broker:** 其实就是RabbitMQ Server, 主要是接收和收发消息

- 生产者(Producer)创建消息, 然后发布到RabbitMQ中. 在实际应用中, 消息通常是一个带有一定业务逻辑结构的数据, 比如JSON字符串. 消息可以带有一定的标签, RabbitMQ会根据标签进行路由, 把消息发送给感兴趣的消费者(Consumer).
- 消费者连接到RabbitMQ服务器, 就可以消费消息了, 消费的过程中, 标签会被丢掉. 消费者只会收到消息, 并不知道消息的生产者是谁, 当然消费者也不需要知道.
- 对于RabbitMQ来说, 一个RabbitMQ Broker可以简单地看作一个RabbitMQ服务节点, 或者RabbitMQ服务实例. 大多数情况下也可以将一个RabbitMQ Broker看作一台RabbitMQ服务器

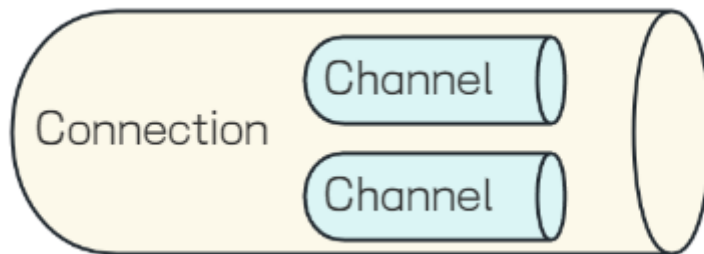


## 2.2 Connection和Channel

**Connection:** 连接. 是客户端和RabbitMQ服务器之间的一个TCP连接. 这个连接是建立消息传递的基础, 它负责传输客户端和服务端之间的所有数据和控制信息.

**Channel:** 通道, 信道. Channel是在Connection之上的一个抽象层. 在RabbitMQ中, 一个TCP连接可以有多个Channel, 每个Channel都是独立的虚拟连接. 消息的发送和接收都是基于Channel的.

通道的主要作用是将消息的读写操作复用到同一个TCP连接上, 这样可以减少建立和关闭连接的开销, 提高性能.



## 2.3 Virtual host

**Virtual host:** 虚拟主机. 这是一个虚拟概念. 它为消息队列提供了一种逻辑上的隔离机制. 对于 RabbitMQ 而言, 一个 BrokerServer 上可以存在多个 Virtual Host. 当多个不同的用户使用同一个 RabbitMQ Server 提供的服务时, 可以虚拟划分出多个 vhost, 每个用户在自己的 vhost 创建 exchange/queue 等

类似MySQL的"database", 是一个逻辑上的集合. 一个MySQL服务器可以有多个database.

## 2.4 Queue

Queue: 队列, 是RabbitMQ的内部对象, 用于存储消息.



多个消费者, 可以订阅同一个队列

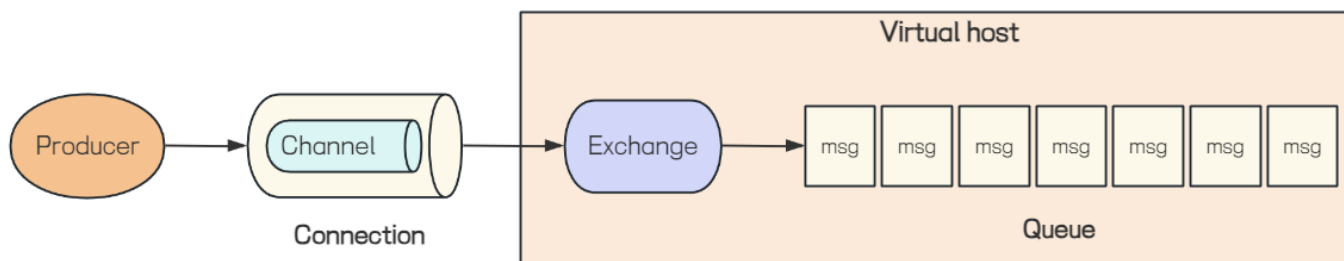


## 2.5 Exchange

**Exchange:** 交换机. message 到达 broker 的第一站, 它负责接收生产者发送的消息, 并根据特定的规则把这些消息路由到一个或多个Queue列中.

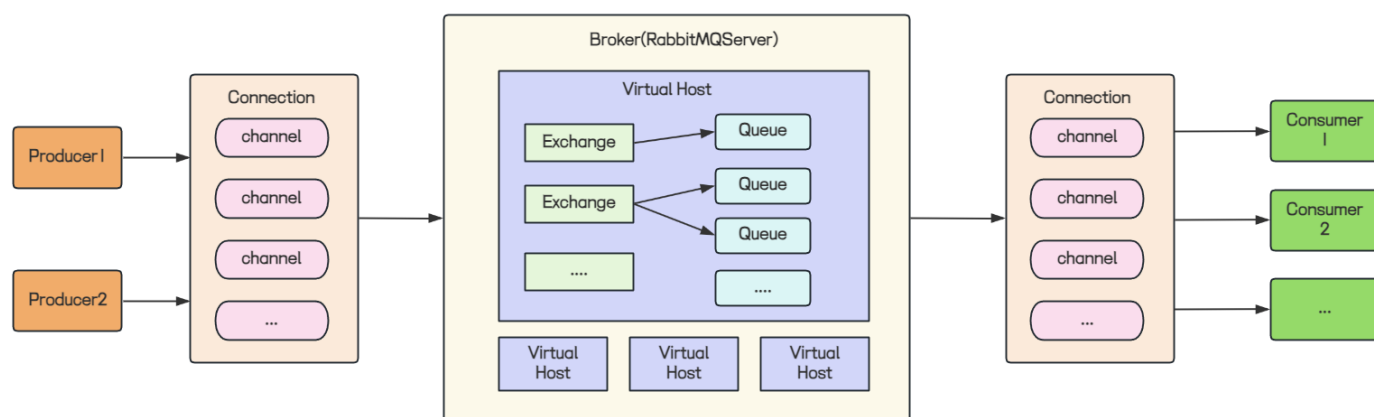
Exchange起到了消息路由的作用, 它根据类型和规则来确定如何转发接收到的消息.

类似于发快递之后, 物流公司怎么处理呢, 根据咱们的地址来分派这个快递到不同的站点, 然后再送到收件人手里. 这个分配的工作, 就是交换机来做的



## 2.6 RabbitMQ工作流程

理解了上面的概念之后, 再来回顾一下这个图, 来看RabbitMQ的工作流程



1. Producer 生产了一条消息
2. Producer 连接到RabbitMQBroker, 建立一个连接(Connection),开启一个信道(Channel)
3. Producer 声明一个交换机(Exchange), 路由消息
4. Producer 声明一个队列(Queue), 存放信息
5. Producer 发送消息至RabbitMQ Broker
6. RabbitMQ Broker 接收消息, 并存入相应的队列(Queue)中, 如果未找到相应的队列, 则根据生产者的配置, 选择丢弃或者退回给生产者.

如果我们把RabbitMQ比作一个物流公司, 那么它的一些核心概念可以这样理解:

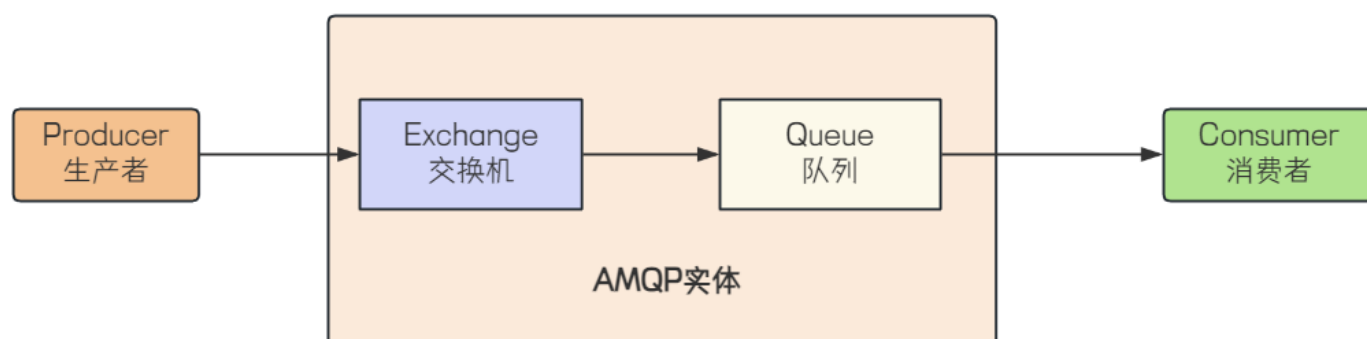
1. Broker就类似整个物流公司的总部, 它负责协调和管理所有的物流站点, 确保包裹安全、高效地送达.
2. Virtual Host可以看作是物流公司为不同的客户或业务部门划分的独立运营中心. 每个运营中心都有自己的仓库(Queue), 分拣规则(Exchange)和运输路线(Connection和Channel), 这样可以确保不同客户的包裹处理不会相互干扰, 同时提供定制化的服务
3. Exchange就像是站点里的分拣中心. 当包裹到达时, 分拣中心会根据包裹上的标签来决定这个包裹应该送往哪个目的地(队列). 快递站点可能有不同类型的分拣中心, 有的按照具体地址分拣, 有的将包裹复制给多个收件人等.

4. Queue就是快递站点里的一个个仓库, 用来临时存放等待派送的包裹. 每个仓库都有一个或多个快递员(消费者)负责从仓库中取出包裹并派送给最终的收件人.
5. Connection就像是快递员与快递站点之间的通信线路. 快递员需要通过这个线路来接收派送任务(消息).
6. Channel就像是快递员在执行任务时使用的多个并行的通信线路. 这样, 快递员可以同时处理多个包裹, 比如一边派送包裹, 一边接收新的包裹

### 3. AMQP

AMQP (Advanced Message Queuing Protocol) 是一种高级消息队列协议, AMQP定义了一套确定的消息交换功能, 包括交换器(Exchange), 队列(Queue) 等. 这些组件共同工作, 使得生产者能够将消息发送到交换器. 然后由队列接收并等待消费者接收. AMQP还定义了一个网络协议, 允许客户端应用通过该协议与消息代理和AMQP模型进行交互通信

RabbitMQ是遵从AMQP协议的,换句话说,RabbitMQ就是AMQP协议的Erlang的实现(当然RabbitMQ还支持STOMP2, MQTT2等协议). AMQP的模型结构和RabbitMQ的模型结构是一样的.



### 4. web界面操作

RabbitMQ管理界面上的Connections, Channels, Exchange, Queues 就是和上面流程图的概念是一样的, Overview就是视图的意思, Admin是用户管理.

我们在操作RabbitMQ前, 需要先创建Virtual host

接下来看具体操作:

#### 4.1 用户相关操作

##### 添加用户

a) 点击 Admin -> Add user

## Users

▼ All users

Filter:  ☐ Regex ?

Name	Tags	Can access virtual hosts	Has password
admin	administrator	No access	•
guest	administrator	/	•

?

▼ Add a user

Username:  \*

Password:  \*  \* (confirm)

Tags:  ?

Set **Admin** | **Monitoring** | **Polycmaker**  
**Management** | **Impersonator** | **None**

Add user

### b) 设置账号密码及权限

▼ Add a user

Username:  ① \*

Password:  ② \*  ③ \* (confirm)

Tags:  ?

Set **Admin** | **Monitoring** | **Polycmaker**  
**Management** | **Impersonator** | **None** ④

Add user

①: 设置账号

②: 设置密码

③: 确认密码

④: 设置权限

添加完成后, 点击[Add user]

### c) 观察用户是否添加成功

# Users

▼ All users

Filter:  ☐ Regex ?

Name	Tags	Can access virtual hosts	Has password
admin	administrator	No access	•
guest	administrator	/	•
study	administrator	No access	•

?

## 用户相关操作

a) 点击要删除的用户, 查看用户详情

OverviewConnectionsChannelsExchangesQueuesAdmin

# Users

▼ All users

Filter:  ☐ Regex ?

Name	Tags	Can access virtual hosts	Has password
admin	administrator	No access	•
guest	administrator	/	•
study	administrator	No access	•

?

b) 在用户详情页面, 进行更新或删除操作

- 设置对虚拟机的操作权限



## ▼ Permissions

Current permissions

... no permissions ...

Set permission

Virtual Host:

Configure regexp:

Write regexp:

Read regexp:

Set permission

- 更新/删除用户

## ▼ Update this user

Password:

\*

\*(confirm)

Tags:

administrator

?

[ Admin ] [ Monitoring ] [ Policymaker ] [ Management ] [ Impersonator ] [ None ]

Update user

## ▼ Delete this user

Delete

退出当前用户

Refreshed 2024-04-12 19:11:46

Refresh every 5 seconds



Virtual host

/



Cluster **rabbit@hcss-ecs-0bb1**

User **admin**

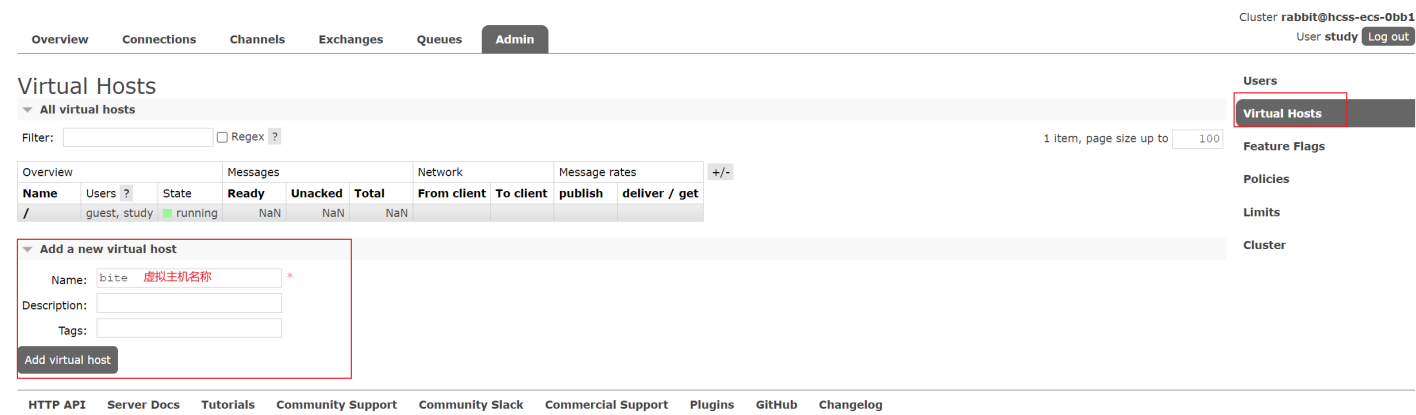
Log out

## 4.2 虚拟主机相关操作

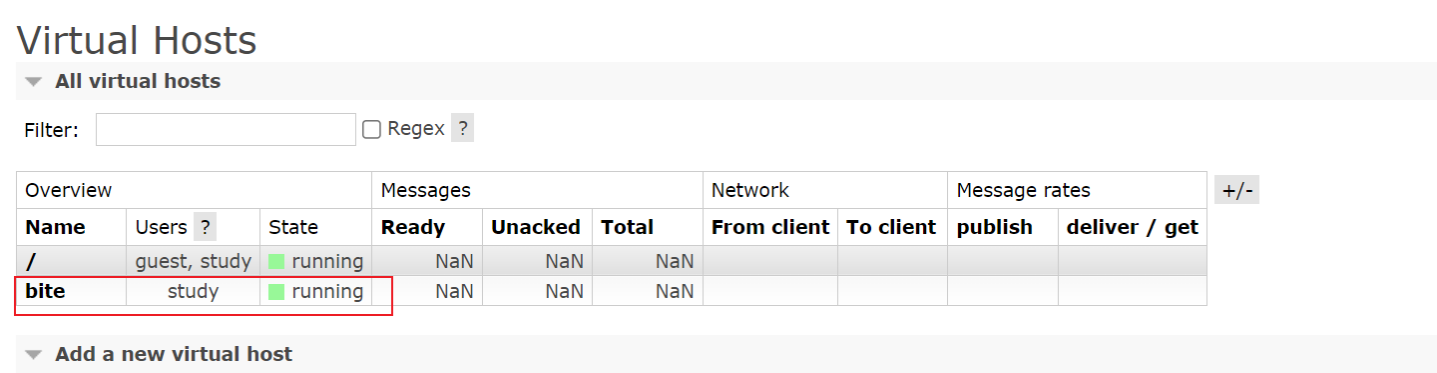
# 创建虚拟主机

在Admin标签页下, 点击右侧 Virtual Hosts -> Add a new virtual host

设置虚拟主机名称



观察设置结果



此操作会为当前登录用户设置虚拟主机

## 5. RabbitMQ快速入门

步骤: 1. 引入依赖

2. 编写生产者代码

3. 编写消费者代码

### 5.1 引入依赖

```
1 <dependency>
2   <groupId>com.rabbitmq</groupId>
3   <artifactId>amqp-client</artifactId>
4   <version>5.7.3</version>
5 </dependency>
```

## 5.2 编写生产者代码

### 5.2.1 创建连接



RabbitMQ 默认的用于客户端连接的TCP 端口号是5672, 需要提前进行开放

本课程修改了端口号为15673

```
1 // 1. 创建连接工厂
2 ConnectionFactory factory = new ConnectionFactory();
3 //2. 设置参数
4 factory.setHost("110.41.51.65");//ip 默认值localhost
5 factory.setPort(15673); //默认值5672
6 factory.setVirtualHost("bite");//虚拟机名称, 默认 /
7
8 factory.setUsername("study");//用户名, 默认guest
9 factory.setPassword("study");//密码, 默认guest
10 //3. 创建连接Connection
11 Connection connection = factory.newConnection();
```

### 5.2.2 创建Channel



生产者和消费者创建的channel并不是同一个

```
1 //4. 创建channel通道
2 Channel channel = connection.createChannel();
```

### 5.2.3 声明一个队列Queue

```
1 /*
2  queueDeclare(String queue, boolean durable, boolean exclusive, boolean
   autoDelete, Map<String, Object> arguments)
3   1.queue: 队列名称
4   2.durable: 是否持久化.true-设置队列为持久化, 持久化的队列会存盘,服务器重启之后, 消息不
   丢失。
5   3.exclusive:
6       * 是否独占, 只能有一个消费者监听队列
7       * 当Connection关闭时, 是否删除队列
```

```

8  4.autoDelete: 是否自动删除, 当没有Consumer时, 自动删除掉
9  5.arguments: 一些参数
10 */
11 //如果没有一个hello_world 这样的队列, 会自动创建, 如果有, 则不创建
12 channel.queueDeclare("hello",true,false,false,null);

```

## 5.2.4 发送消息

当一个新的 RabbitMQ 节点启动时, 它会预声明 (declare) 几个内置的交换机, 内置交换机名称是空字符串(""). 生产者发送的消息会根据队列名称直接路由到对应的队列.

Overview

Connections

Channels

Exchanges

Queues

Admin

Exchanges

▼ All exchanges (7)

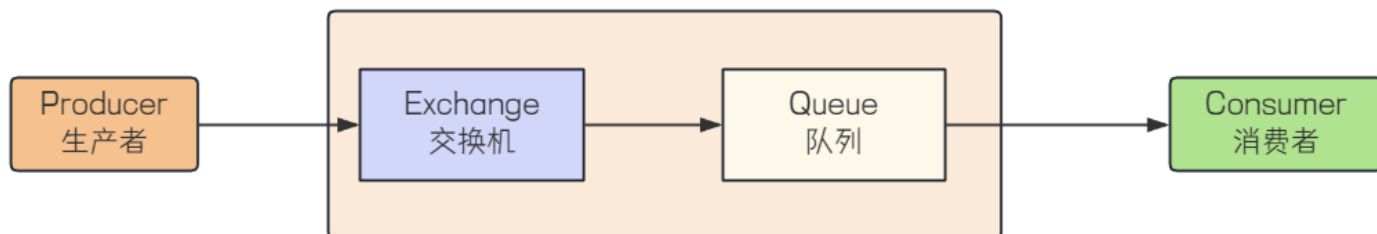
Pagination

Page 1 ▼ of 1 - Filter:

☐ Regex ?

Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
/	(AMQP default)	direct	D			
/	amq.direct	direct	D			
/	amq.fanout	fanout	D			
/	amq.headers	headers	D			
/	amq.match	headers	D			
/	amq.rabbitmq.trace	topic	D I			
/	amq.topic	topic	D			

例如: 如果有一个名为 "hello" 的队列, 生产者可以直接发送消息到 "hello" 队列, 而消费者可以从 "hello" 队列中接收消息, 而不需要关心交换机的存在. 这种模式非常适合简单的应用场景, 其中生产者和消费者之间的通信是一对一的.



```

1 //6. 通过channel发送消息到队列中
2 /*
3 basicPublish(String exchange, String routingKey, AMQP.BasicProperties props,
  byte[] body)

```

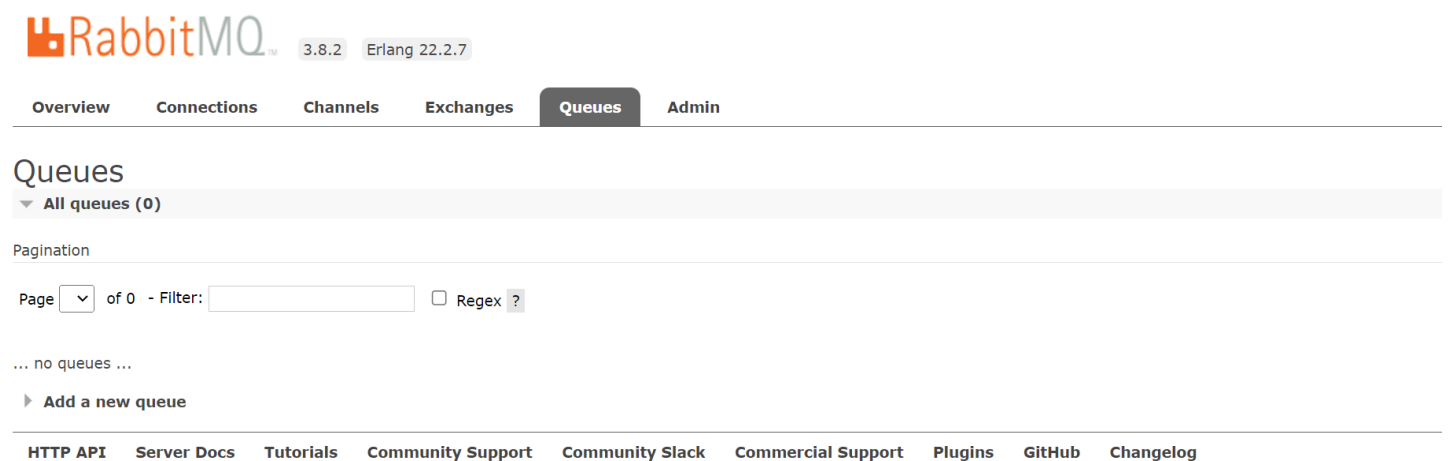
```
4 1.exchange: 交换机名称, 简单模式下, 交换机会使用默认的""
5 2.routingKey: 路由名称, routingKey = 队列名称
6 3.props: 配置信息
7 4.body: 发送消息的数据
8 */
9 String msg = "Hello World";
10 //使用的是内置交换机. 使用内置交换机时, routingKey要和队列名称一样, 才可以路由到对应的队
    列上去
11 channel.basicPublish("", "hello", null, msg.getBytes());
12 System.out.println(msg + "消息发送成功");
```

## 5.2.5 释放资源

```
1 //显式地关闭Channel是个好习惯, 但这不是必须的, Connection关闭的时候, Channel也会自动关
    闭.
2 channel.close();
3 connection.close();
```

## 5.2.6 运行代码, 观察结果


运行之前



运行之后, 队列中就已经有了hello这个队列的信息



右上角需要选择虚拟机

 3.8.2 Erlang 22.2.7

Overview Connections Channels Exchanges **Queues** Admin

Cluster **rabbit@hcss-ecs-0bb1**  
User **study** [Log out](#)

## Queues

▼ All queues (1)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Displaying 1 item, page size up to: 100

Overview					Messages			Message rates				+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack		
bite	hello	classic	<span>D</span>	idle	1	0	1	0.00/s				

► Add a new queue

[HTTP API](#) [Server Docs](#) [Tutorials](#) [Community Support](#) [Community Slack](#) [Commercial Support](#) [Plugins](#) [GitHub](#) [Changelog](#)

如果在代码中注掉资源释放的代码, 在Connections和Channels也可以看到相关信息

Overview Connections **Channels** Exchanges Queues Admin

## Channels

▼ All channels (1)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Overview					Details			Message rates					+/-
Channel	Virtual host	User name	Mode ?	State	Unconfirmed	Prefetch ?	Unacked	publish	confirm	unroutable (drop)	deliver / get	ack	
113.132.216.225:23498 (1)	bite	study		idle	0		0	0.00/s	0.00/s	0.00/s			

[HTTP API](#) [Server Docs](#) [Tutorials](#) [Community Support](#) [Community Slack](#) [Commercial Support](#) [Plugins](#) [GitHub](#) [Changelog](#)

Queue也可以配置显示Consumer相关信息

Overview Connections Channels Exchanges **Queues** Admin

## Queues

► All queues (1)

Overview				Consumers				Messages			Message rates			+/-
Virtual host	Name	Type	Features	Consumer count	Consumer utilisation	State	Ready	Unacked	Total	incoming	deliver / get	ack		
bite	hello	classic	<span>D</span>	0	0%	idle	4	0	4	0.00/s				

► Add a new queue

[HTTP API](#) [Server Docs](#) [Tutorials](#) [Community Support](#) [Community Slack](#) [Commercial Support](#) [Plugins](#) [GitHub](#) [Changelog](#)

Overview:

☒ Type

☒ Features (with policy)

☐ Features (no policy)

☐ Policy

☒ Consumer count

☒ Consumer utilisation

☒ State

Messages:

☒ Ready

## 5.3 编写消费者代码

消费者代码和生产者前3步都是一样的, 第4步改为消费当前队列

1. 创建连接
2. 创建Channel
3. 声明一个队列Queue
4. 消费消息
5. 释放资源

### 5.3.1 消费当前队列

#### basicConsume

```
1 /*
2 basicConsume(String queue, boolean autoAck, Consumer callback)
```

```
3  参数:
4  1. queue: 队列名称
5  2. autoAck: 是否自动确认, 消费者收到消息之后, 自动和MQ确认
6  3. callback: 回调对象
7  */
8  String basicConsume(String queue, boolean autoAck, Consumer callback) throws
    IOException;
```

## Consumer

`Consumer` 用于定义消息消费者的行为. 当我们需要从RabbitMQ接收消息时, 需要提供一个实现了 `Consumer` 接口的对象.

`DefaultConsumer` 是 RabbitMQ提供的一个默认消费者, 实现了 `Consumer` 接口.

核心方法:

1. `handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body)`: 从队列接收到消息时, 会自动调用该方法.

在这个方法中, 我们可以定义如何处理接收到的消息, 例如打印消息内容, 处理业务逻辑或者将消息存储到数据库等.

参数说明如下:

- `consumerTag`: 消费者标签, 通常是消费者在订阅队列时指定的.
- `envelope`: 包含消息的封包信息, 如队列名称, 交换机等.
- `properties`: 一些配置信息
- `body`: 消息的具体内容

```
1  //6. 接收消息, 并消费
2  /*
3  basicConsume(String queue, boolean autoAck, Consumer callback)
4  参数:
5  1. queue: 队列名称
6  2. autoAck: 是否自动确认, 消费者收到消息之后, 自动和MQ确认
7  3. callback: 回调对象
8  */
9  DefaultConsumer consumer = new DefaultConsumer(channel) {
10     /*
11     回调方法, 当收到消息后, 会自动执行该方法
12     1. consumerTag: 标识
13     2. envelope: 获取一些信息, 交换机, 路由key
```

```

14      3. properties:配置信息
15      4. body:数据
16
17      */
18      @Override
19      public void handleDelivery(String consumerTag, Envelope envelope,
    AMQP.BasicProperties properties, byte[] body) throws IOException {
20          System.out.println("接收到消息: " + new String(body));
21      }
22  };
23  channel.basicConsume("hello", true, consumer);

```

### 5.3.2 释放资源

```

1  //等待回调函数执行完毕之后, 关闭资源
2  TimeUnit.SECONDS.sleep(5);
3  //7. 释放资源    消费者相当于是一个监听程序, 不需要关闭资源
4  channel.close();
5  connection.close();

```

实际上消费者相当于是一个监听程序, 不需要关闭资源

### 5.3.3 运行代码, 观察结果

运行程序, 我们刚才发送的消息, 就收到了

```

1  接收到消息: Hello World

```

如果我们不释放资源, 可以看到响应的Connection, channel

[Overview](#)
[Connections](#)
[Channels](#)
[Exchanges](#)
[Queues](#)
[Admin](#)

## Connections

▼ All connections (1)

Pagination

Page  of 1 - Filter:  ☐ Regex ?

Overview				Details			Network		+/-
Virtual host	Name	User name	State	SSL / TLS	Protocol	Channels	From client	To client	
bite	113.132.216.225:23356?	study	running	o	AMQP 0-9-1	1	0iB/s	0iB/s	

[HTTP API](#)
[Server Docs](#)
[Tutorials](#)
[Community Support](#)
[Community Slack](#)
[Commercial Support](#)
[Plugins](#)
[GitHub](#)
[Changelog](#)



## Channels

▼ All channels (1)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Overview					Details			Message rates					+/-
Channel	Virtual host	User name	Mode ?	State	Unconfirmed	Prefetch ?	Unacked	publish	confirm	unroutable (drop)	deliver / get	ack	
113.132.216.225:23356 (1)	bite	study		Idle	0		0				0.00/s	0.00/s	

[HTTP API](#)
[Server Docs](#)
[Tutorials](#)
[Community Support](#)
[Community Slack](#)
[Commercial Support](#)
[Plugins](#)
[GitHub](#)
[Changelog](#)

## 5.4 附源码

## 生产者代码

```

1 import com.rabbitmq.client.Channel;
2 import com.rabbitmq.client.Connection;
3 import com.rabbitmq.client.ConnectionFactory;
4
5 public class RabbitProducer {
6
7     public static void main(String[] args) throws Exception {
8         // 1. 创建连接工厂
9         ConnectionFactory factory = new ConnectionFactory();
10        //2. 设置参数
11        factory.setHost("110.41.51.65");//ip 默认值localhost
12        factory.setPort(15673); //默认值5672
13        factory.setVirtualHost("bite");//虚拟机名称, 默认 /
14
15        factory.setUsername("study");//用户名,默认guest
16        factory.setPassword("study");//密码, 默认guest
17        //3. 创建连接Connection
18        Connection connection = factory.newConnection();
19        //4. 创建channel通道
20        Channel channel = connection.createChannel();
21        //5. 声明队列
22        /*
23        queueDeclare(String queue, boolean durable, boolean exclusive,
24        boolean autoDelete, Map<String, Object> arguments)
25        1.queue: 队列名称
26        2.durable: 是否持久化, 当mq重启之后, 消息还在
27        3.exclusive:
28            * 是否独占, 只能有一个消费者监听队列
29            * 当Connection关闭时, 是否删除队列
30        4.autoDelete: 是否自动删除, 当没有Consumer时, 自动删除掉
31        5.arguments: 一些参数

```

```

31      */
32      //如果没有一个hello 这样的队列，会自动创建，如果有，则不创建
33      channel.queueDeclare("hello", true, false, false, null);
34      //6. 通过channel发送消息到队列中
35      /*
36      basicPublish(String exchange, String routingKey, AMQP.BasicProperties
37      props, byte[] body)
38      1. exchange: 交换机名称，简单模式下，交换机会使用默认的""
39      2.routingKey: 路由名称，routingKey = 队列名称
40      3.props: 配置信息
41      4.body: 发送消息的数据
42      */
43      String msg = "Hello World";
44      //使用的是内置交换机。使用内置交换机时，routingKey要和队列名称一样，才可以路由
45      到对应的队列上去
46      channel.basicPublish("", "hello", null, msg.getBytes());
47
48      //7. 释放资源
49      System.out.println(msg + "消息发送成功");
50      channel.close();
51      connection.close();
52  }
53 }

```

## 消费者代码

```

1  import com.rabbitmq.client.*;
2
3  import java.io.IOException;
4  import java.util.concurrent.TimeUnit;
5
6  public class RabbitmqConsumer {
7      public static void main(String[] args) throws Exception {
8          // 1. 创建连接工厂
9          ConnectionFactory factory = new ConnectionFactory();
10         //2. 设置参数
11         factory.setHost("110.41.51.65");//ip 默认值localhost
12         factory.setPort(15673); //默认值5672
13         factory.setVirtualHost("bte");//虚拟机名称，默认 /
14
15         factory.setUsername("study");//用户名,默认guest
16         factory.setPassword("study");//密码，默认guest
17         //3. 创建连接Connection
18         Connection connection = factory.newConnection();
19         //4. 创建channel通道

```

```

20     Channel channel = connection.createChannel();
21     //5. 声明队列
22     /*
23         queueDeclare(String queue, boolean durable, boolean exclusive,
boolean autoDelete, Map<String, Object> arguments)
24         1.queue: 队列名称
25         2.durable: 是否持久化, 当mq重启之后, 消息还在
26         3.exclusive:
27             * 是否独占, 只能有一个消费者监听队列
28             * 当Connection关闭时, 是否删除队列
29         4.autoDelete: 是否自动删除, 当没有Consumer时, 自动删除掉
30         5.arguments: 一些参数
31     */
32     //如果没有一个hello 这样的队列, 会自动创建, 如果有, 则不创建
33     channel.queueDeclare("hello", true, false, false, null);
34     //6. 接收消息, 并消费
35     /*
36         basicConsume(String queue, boolean autoAck, Consumer callback)
37     参数:
38         1. queue: 队列名称
39         2. autoAck: 是否自动确认, 消费者收到消息之后, 自动和MQ确认
40         3. callback: 回调对象
41     */
42     DefaultConsumer consumer = new DefaultConsumer(channel) {
43         /*
44             回调方法, 当收到消息后, 会自动执行该方法
45             1. consumerTag: 标识
46             2. envelope: 获取一些信息, 交换机, 路由key
47             3. properties: 配置信息
48             4. body: 数据
49
50         */
51         @Override
52         public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
53             System.out.println("接收到消息: " + new String(body));
54         }
55     };
56     channel.basicConsume("hello", true, consumer);
57     //等待回调函数执行完毕之后, 关闭资源
58     TimeUnit.SECONDS.sleep(5);
59     //7. 释放资源    消费者相当于是一个监听程序, 不需要关闭资源
60     //顺序不可改变
61     // channel.close();
62     // connection.close();
63 }
64 }

```