

MQ多用于分布式系统之间进行通信。

系统之间的调用通常有两种方式：

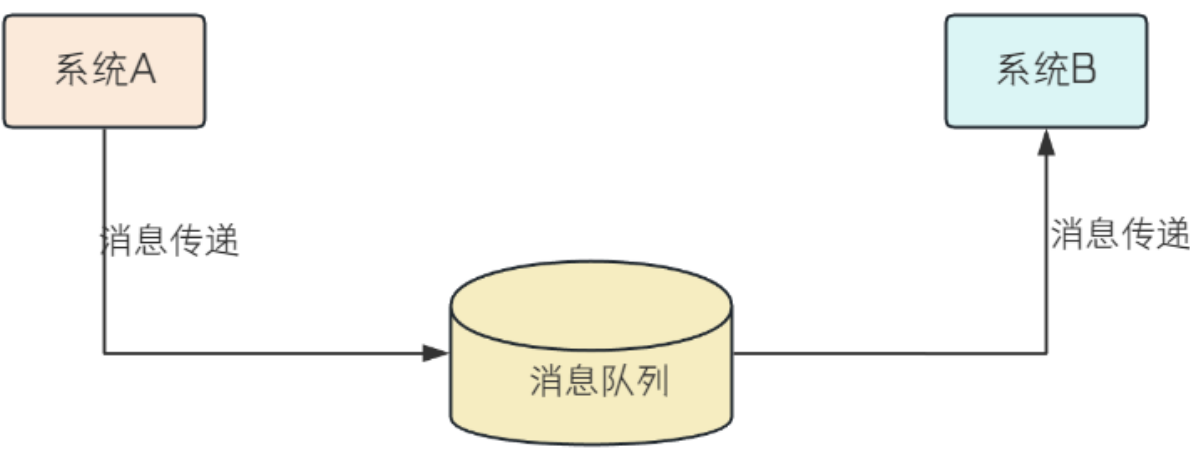
1. 同步通信

直接调用对方的服务, 数据从一端发出后立即就可以达到另一端。



2. 异步通信

数据从一端发出后，先进入一个容器进行临时存储，当达到某种条件后，再由这个容器发送给另一端。容器的一个具体实现就是MQ( `message queue` )



`RabbitMQ` 就是MQ的一种实现

2. MQ的作用

MQ主要工作是接收并转发消息, 在不同的应用场景下可以展现不同的作用

可以把MQ想象成一个仓库. 采购部门进货之后, 把零件放进仓库里, 生产部门从仓库中取出零件, 并加工成产品. MQ和仓库的区别是, 仓库里放的是物品, MQ里放的是消息, 仓库负责存储物品,并转发物品, MQ负责存储和转发消息



1. **异步解耦:** 在业务流程中, 一些操作可能非常耗时, 但并不需要即时返回结果. 可以借助MQ把这些操作异步化, 比如 用户注册后发送注册短信或邮件通知, 可以作为异步任务处理, 而不必等待这些操作完成后才告知用户注册成功.
2. **流量削峰:** 在访问量剧增的情况下, 应用仍然需要继续发挥作用, 但是是这样的突发流量并不常见. 如果以能处理这类峰值为标准而投入资源,无疑是巨大的浪费. 使用MQ能够使关键组件支撑突发访问压力, 不会因为突发流量而崩溃. 比如秒杀或者促销活动, 可以使用MQ来控制流量, 将请求排队, 然后系统根据自己的处理能力逐步处理这些请求.
3. **异步通信:** 在很多时候应用不需要立即处理消息, MQ提供了异步处理机制, 允许应用把一些消息放入MQ中, 但并不立即处理它,在需要的时候再慢慢处理.
4. **消息分发:** 当多个系统需要对同一数据做出响应时, 可以使用MQ进行消息分发. 比如支付成功后, 支付系统可以向MQ发送消息, 其他系统订阅该消息, 而无需轮询数据库.
5. **延迟通知:** 在需要在特定时间后发送通知的场景中, 可以使用MQ的延迟消息功能, 比如在电子商务平台中, 如果用户下单后一定时间内未支付, 可以使用延迟队列在超时后自动取消订单
6. ....

## 3. 为什么选择 RabbitMQ

目前业界有很多的MQ产品, 例如RabbitMQ, RocketMQ, ActiveMQ, Kafka, ZeroMQ等, 也有直接使用Redis充当消息队列的案例, 这些消息队列, 各有侧重, 也没有好坏, 只有适合不适合, 在实际选型时, 需要结合自身需求以及MQ产品特征, 综合考虑

下面我们介绍一下当前最主流的3种MQ产品

### 3.1 Kafka

Kafka一开始的目的就是用于日志收集和传输, 追求高吞吐量, 性能卓越, 单机吞吐达到十万级, 在日志领域比较成熟, 功能较为简单, 主要支持简单的 MQ 功能, 如果有日志采集需求, 肯定是首选kafka了。

### 3.2 RocketMQ

RocketMQ采用Java语言开发, 由阿里巴巴开源, 后捐赠给了Apache.

它在设计时借鉴了Kafka, 并做出了一些自己的改进, 青出于蓝而胜于蓝, 经过多年双十一的洗礼, 在可用性、可靠性以及稳定性等方面都有出色的表现. 适合对于可靠性比较高,且并发比较大的场景, 比如互联网金融. 但支持的客户端语言不多, 且社区活跃度一般

### 3.3 RabbitMQ

采用Erlang语言开发, MQ 功能比较完备, 且几乎支持所有主流语言, 开源提供的界面也非常友好, 性能较好, 吞吐量能达到万级, 社区活跃度也比较高, 比较适合中小型公司, 数据量没那么大, 且并发没

那么高的场景.

综合: 由于 RabbitMQ 的综合能力较强, 咱们这边的项目没有那么大的高并发, 且RabbitMQ社区比较成熟, 管理界面友好, 所以咱们接下来主要学习RabbitMQ的使用

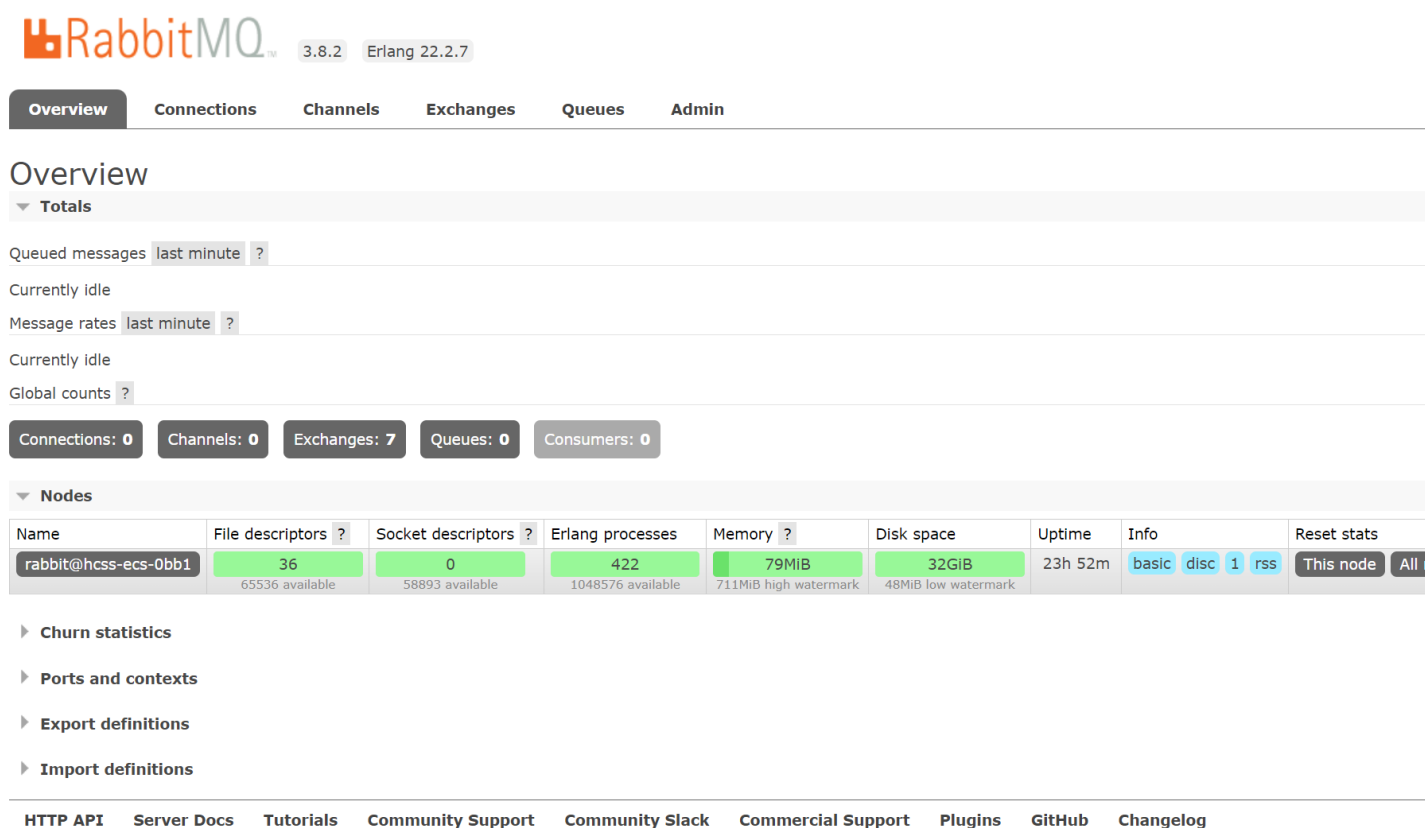
## 4. RabbitMQ 的安装

📖 RabbitMQ 安装

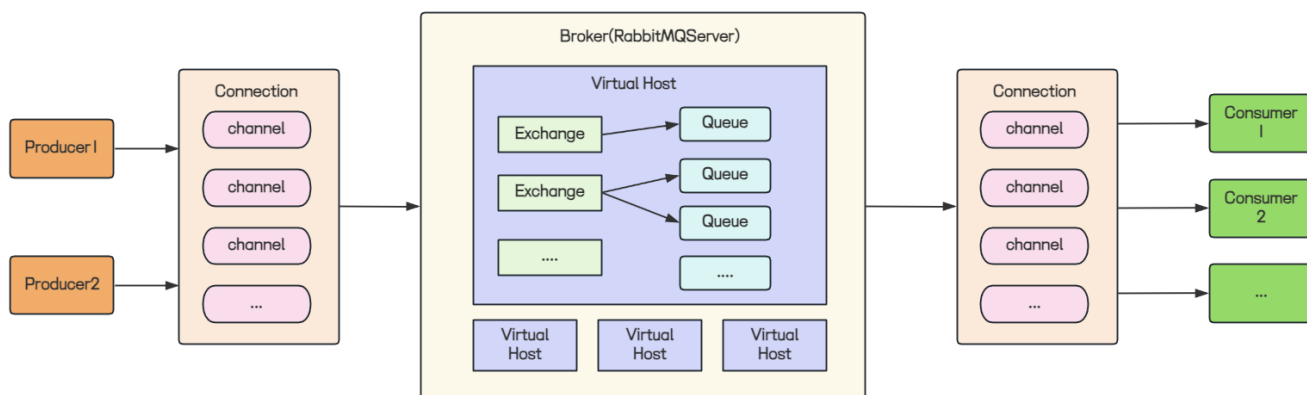
## 5. RabbitMQ 核心概念

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



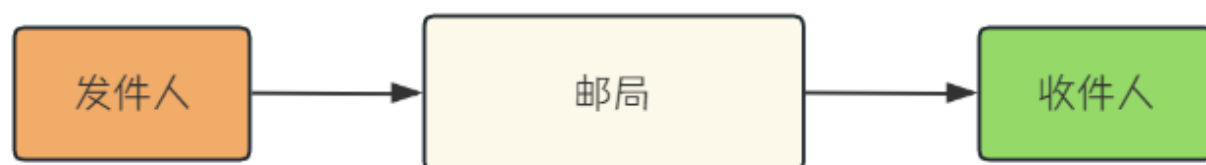
界面上的导航栏共分6部分, 这6部分分别是什么意思呢, 我们先看看RabbitMQ的工作流程



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

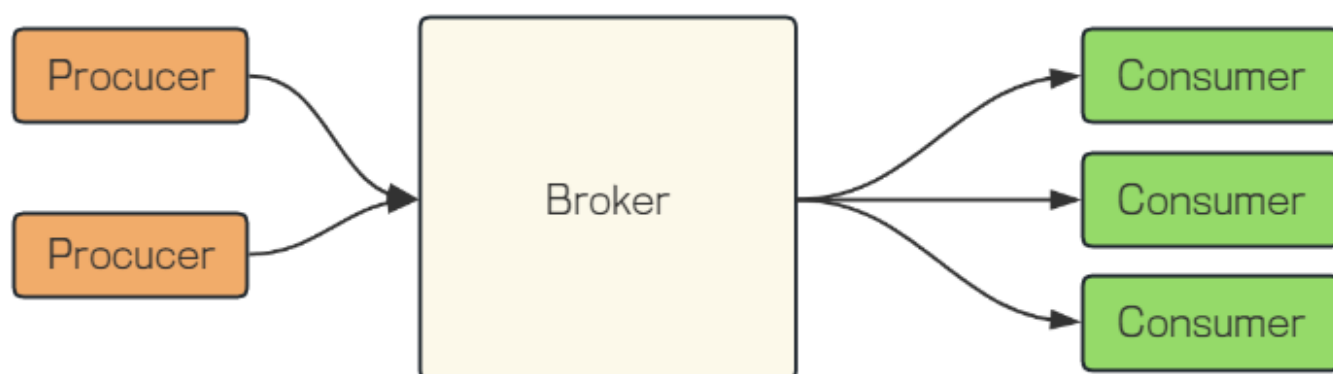
## 5.1 Producer和Consumer

**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

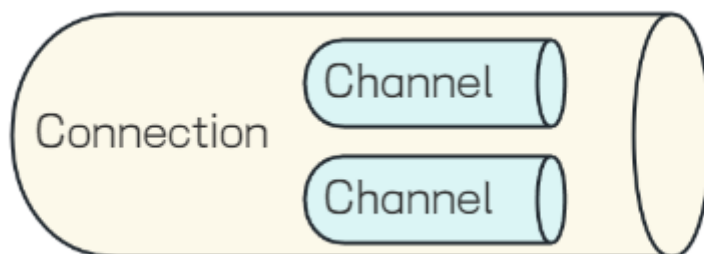


## 5.2 Connection和Channel

**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.



## 5.3 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

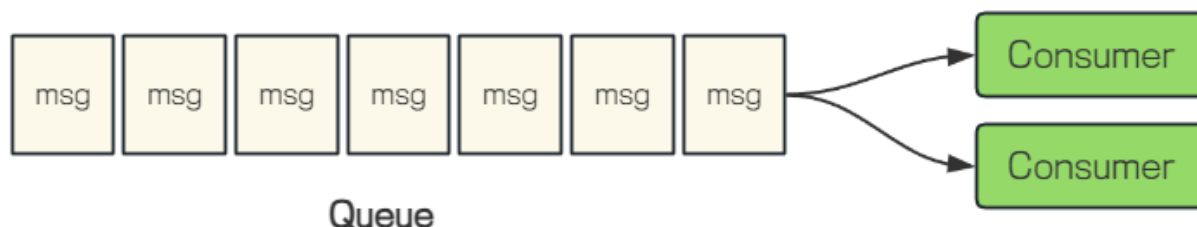
类似MySQL的"database", 是一个逻辑上的集合. 一个MySQL服务器可以有多个database.

## 5.4 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

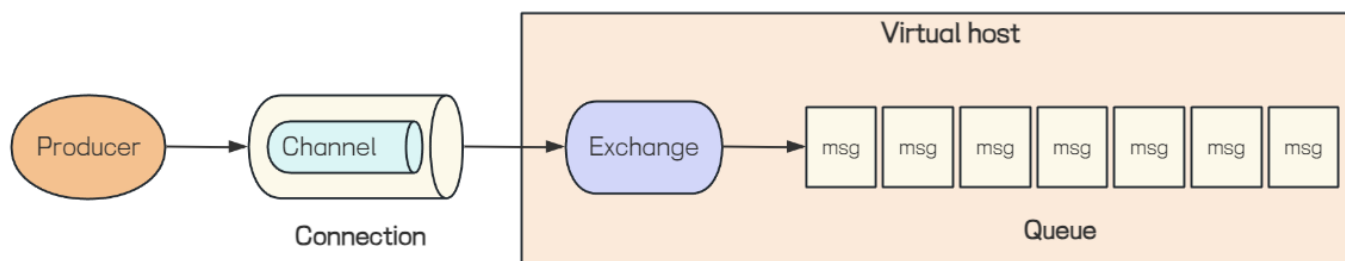


## 5.5 Exchange

**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.

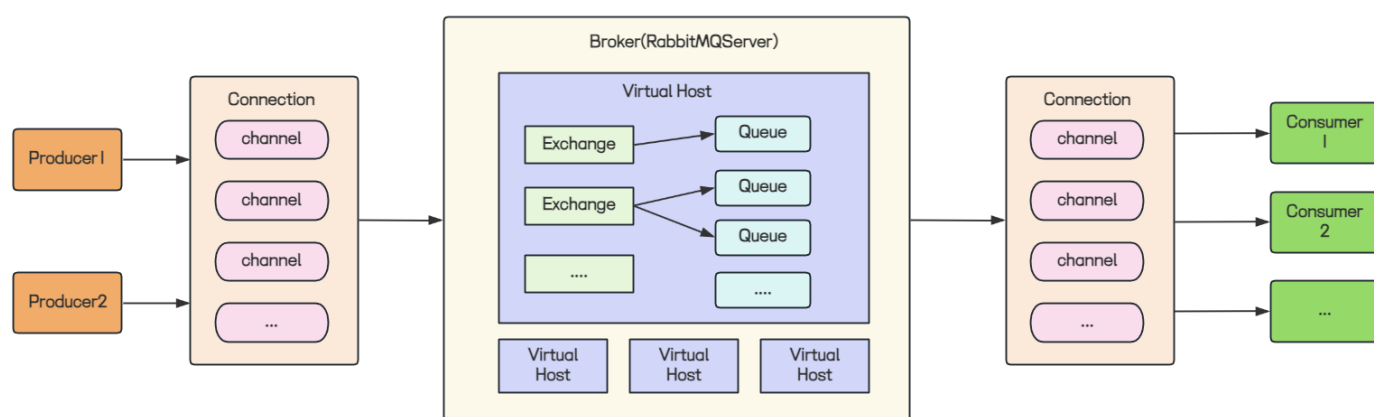
Exchange起到了消息路由的作用, 它根据类型和规则来确定如何转发接收到的消息.

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



## 5.6 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:

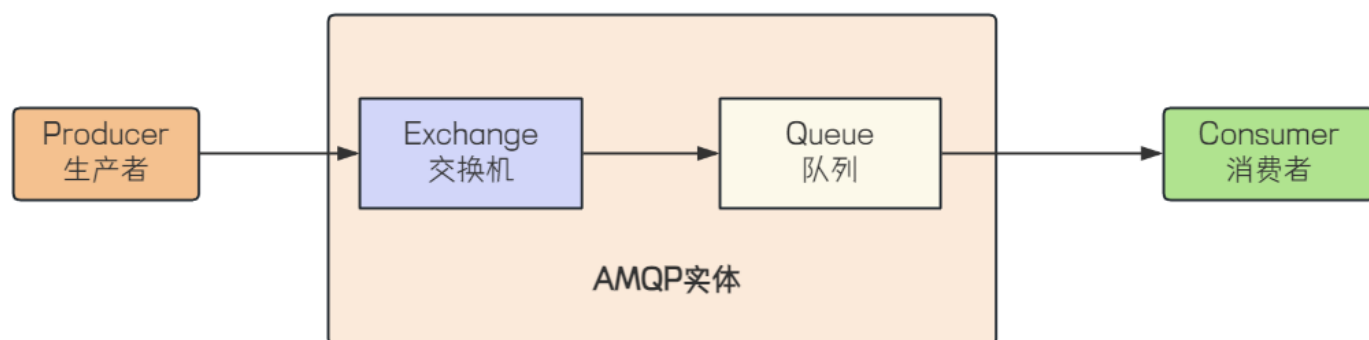
1. Broker就类似整个物流公司的总部, 它负责协调和管理所有的物流站点, 确保包裹安全、高效地送达.

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

## 6. 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.



## 7. 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:

### 7.1 用户相关操作

# 添加用户

a) 点击 Admin -> Add user

3.8.2Erlang 22.2.7

Overview

Connections

Channels

Exchanges

Queues

Admin

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 | Policymaker Management | Impersonator | None

Add user

HTTP API

Server Docs

Tutorials

Community Support

Community Slack

Commercial Support

Plugins

GitHub

Changelog

b) 设置账号密码及权限

Add a user

Username: 

study

①

Password: 

▼

.....

.....

②

③

(confirm)

Tags: 

administrator

?

Set Admin | Monitoring | Policymaker 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

Overview

Connections

Channels

Exchanges

Queues

Admin

# 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**

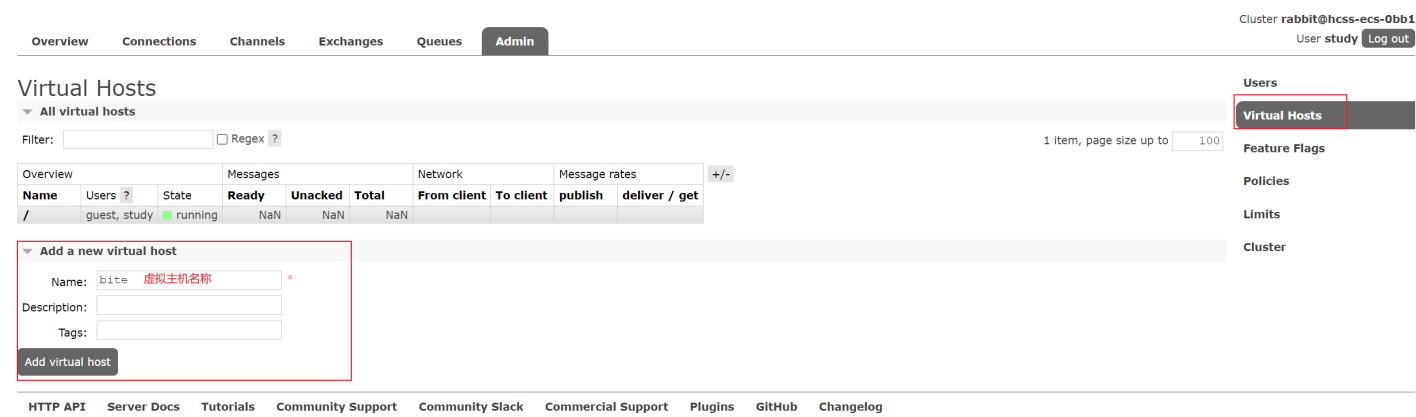
Log out

## 7.2 虚拟主机相关操作

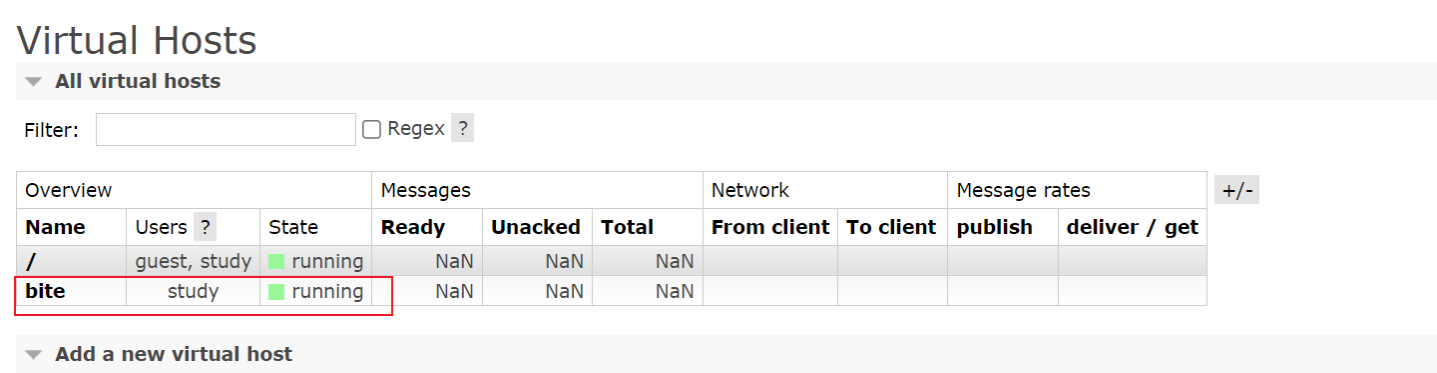
# 创建虚拟主机

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

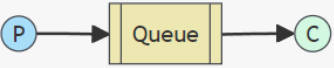
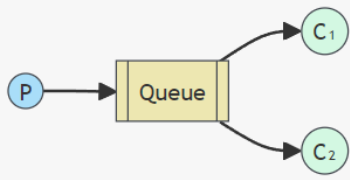
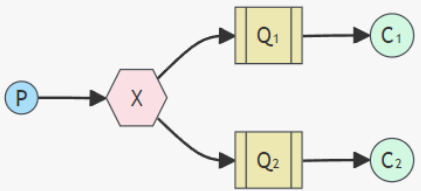
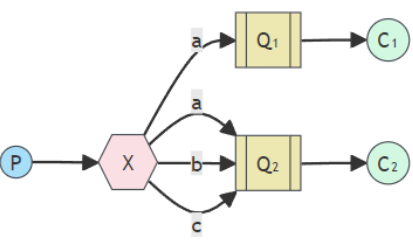
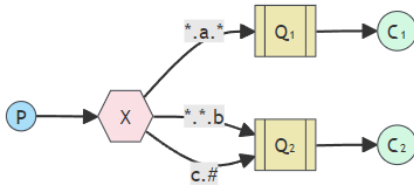
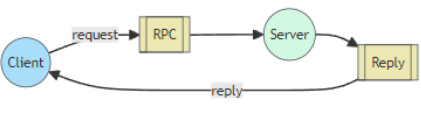
## 8. SpringBoot 继承RabbitMQ

对于RabbitMQ开发, Spring 也提供了一些便利. Spring 和RabbitMQ的官方文档对此均有介绍

Spring官方: [Spring AMQP](#)

RabbitMQ 官方: [RabbitMQ tutorial - "Hello World!"](#) | [RabbitMQ](#)

RabbitMQ 共提供了7种工作模式, 进行消息传递, 我们简单介绍下:

<p><b>1. "Hello World!"</b></p> <p>The simplest thing that does <i>something</i></p> 	<p><b>2. Work Queues</b></p> <p>Distributing tasks among workers (the <i>competing consumers pattern</i>)</p> 	<p><b>3. Publish/Subscribe</b></p> <p>Sending messages to many consumers at once</p> 
<p><b>4. Routing</b></p> <p>Receiving messages selectively</p> 	<p><b>5. Topics</b></p> <p>Receiving messages based on a pattern (topics)</p> 	<p><b>6. RPC</b></p> <p><i>Request/reply pattern</i> example</p> 
<p><b>7. Publisher Confirms</b></p> <p>Reliable publishing with publisher confirms</p>		

下面来看如何基于SpringBoot 进行RabbitMQ的开发.

步骤如下:

1. 引入依赖
2. 编写yml配置, 基本信息配置
3. 编写生产者代码
4. 编写消费者代码
  - a. 定义监听类, 使用@RabbitListener注解完成队列监听
5. 运行观察结果

引入依赖

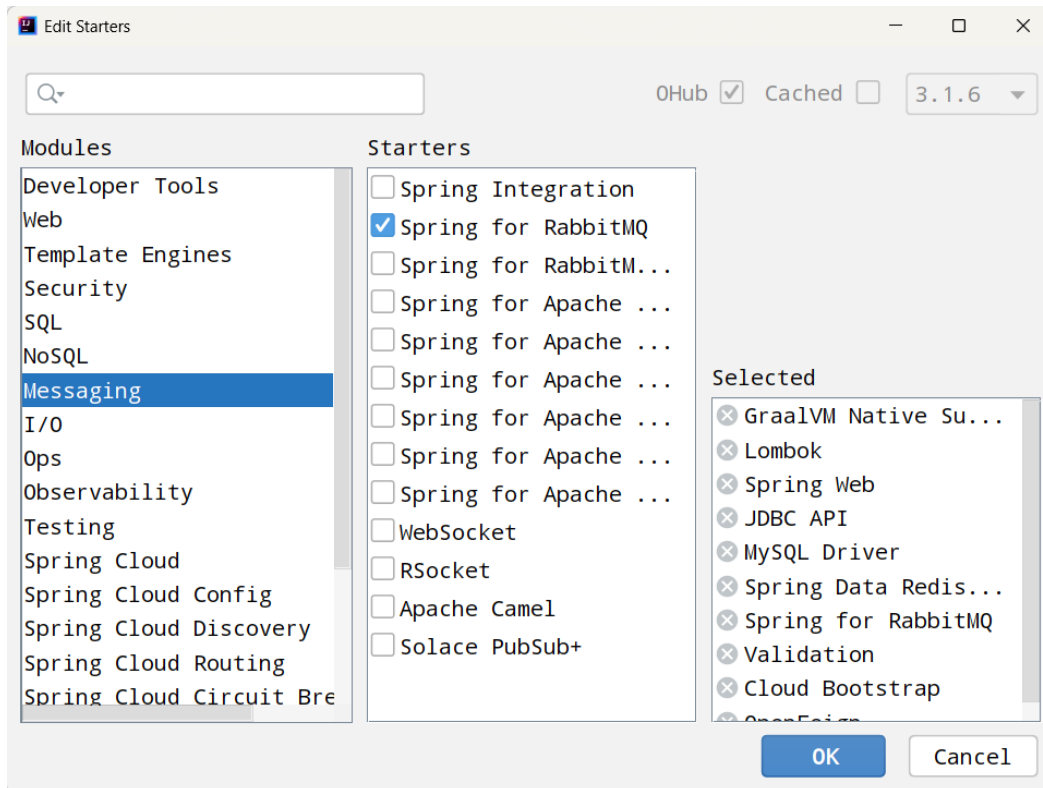
```
1 <!--RabbitMQ相关依赖-->
```

```

2 <dependency>
3   <groupId>org.springframework.boot</groupId>
4   <artifactId>spring-boot-starter-amqp</artifactId>
5 </dependency>

```

也可以通过创建项目时, 加入依赖



## 添加配置

```

1 #配置RabbitMQ的基本信息
2 spring:
3   rabbitmq:
4     host: 110.41.51.65
5     port: 15673 #默认为5672
6     username: study
7     password: study
8     virtual-host: bite #默认值为 /

```

或以下配置

```

1 #amqp://username:password@Ip:port/virtual-host
2 spring:
3   rabbitmq:
4     addresses: amqp://study:study@47.108.157.13:5672/bite

```

# 编写生产者代码

声明队列

```
1 @Configuration
2 public class RabbitMQConfig {
3     @Bean("helloQueue")
4     public Queue workQueue() {
5         return QueueBuilder.durable("hello").build();
6     }
7 }
```

测试发送消息

```
1 @SpringBootTest
2 public class RabbitMQTest {
3     @Autowired
4     private RabbitTemplate rabbitTemplate;
5
6     @Test
7     void send(){
8         rabbitTemplate.convertAndSend("", "hello", "hello, rabbitMQ...");
9     }
10 }
```

运行程序, 观察消息发送成功

Overview						Messages			Message rates			+/-
Virtual host	Name	Node	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
bite	hello	rabbit@iZ2vc7a1n9gvhfp589oav7Z	classic	D	idle	1	0	1	0.00/s			

# 编写消费者代码

定义监听类

```
1 @Component
2 public class QueueListener {
3     @RabbitListener(queues = "hello")
4     public void listenerQueue(Message message){
5         System.out.println("收到消息:" + message);
6     }
7 }
```

`@RabbitListener` 是Spring框架中用于监听RabbitMQ队列的注解, 通过使用这个注解, 可以定义一个方法, 以便从RabbitMQ队列中接收消息. 该注解支持多种参数类型, 这些参数类型代表了从RabbitMQ接收到的消息和相关信息.

以下是一些常用的参数类型:

1. `String`: 返回消息的内容
2. `Message` (`org.springframework.amqp.core.Message`): Spring AMQP的 `Message` 类, 返回原始的消息体以及消息的属性, 如消息ID, 内容, 队列信息等.
3. `Channel` (`com.rabbitmq.client.Channel`): RabbitMQ的通道对象, 可以用于进行更高级的操作, 如手动确认消息.

## 运行程序, 观察结果

消费者打印消息内容

- ```
1 收到消息:GenericMessage [payload=hello, rabbitMQ..., headers=
  {amqp_receivedDeliveryMode=PERSISTENT, amqp_receivedRoutingKey=hello,
  amqp_contentEncoding=UTF-8, amqp_deliveryTag=1, amqp_consumerQueue=hello,
  amqp_redelivered=false, id=82d4b2e1-8232-54ad-1a9d-5b4b8c62e399,
  amqp_consumerTag=amq.ctag-Ri81oWHNp1Ey1cuS6LkliQ, amqp_lastInBatch=false,
  contentType=text/plain, timestamp=1727235645949}]
2 收到消息:GenericMessage [payload=hello, rabbitMQ..., headers=
  {amqp_receivedDeliveryMode=PERSISTENT, amqp_receivedRoutingKey=hello,
  amqp_contentEncoding=UTF-8, amqp_deliveryTag=2, amqp_consumerQueue=hello,
  amqp_redelivered=false, id=353140e5-4da6-9664-fbe3-589fed9d86aa,
  amqp_consumerTag=amq.ctag-Ri81oWHNp1Ey1cuS6LkliQ, amqp_lastInBatch=false,
  contentType=text/plain, timestamp=1727235646384}]
```