# 官网

https://www.rabbitmq.com/

# 下载地址

https://www.rabbitmq.com/download.html

## windows版本安装包下载地址

https://www.rabbitmq.com/install-windows.html#installer

# windows安装rabbitmq教程

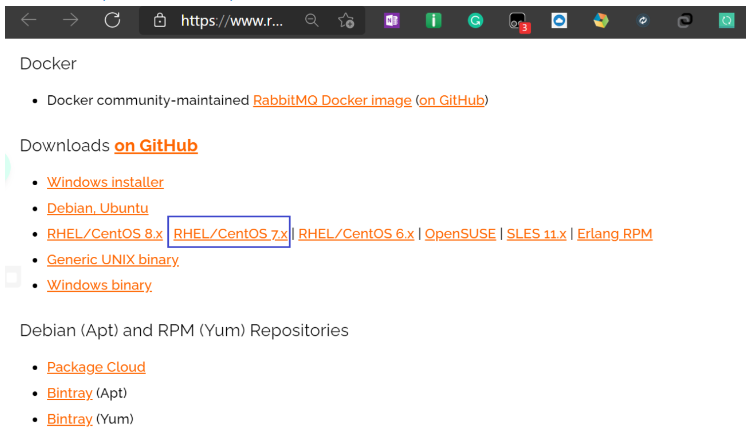
https://blog.csdn.net/qq\_25919879/article/details/113055350

# linux安装rabbitmq教程

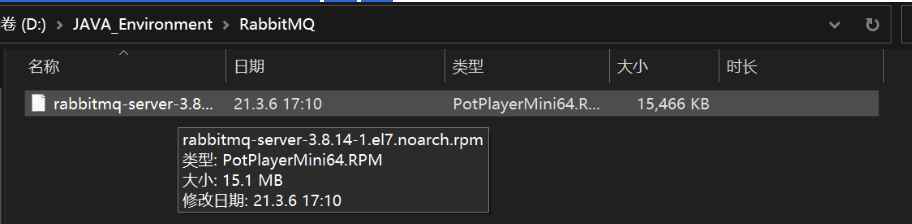
https://blog.csdn.net/qq\_45173404/article/details/116429302

## 下载RabbitMQ

**下载地址**：<https://www.rabbitmq.com/download.html>

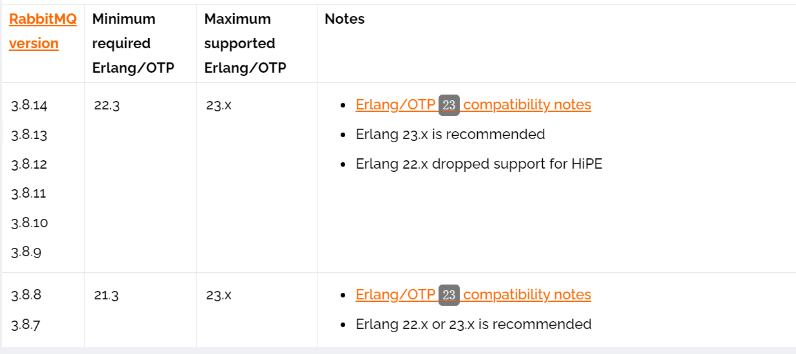


选择对应的系统版本点击下载，下载后会得到.rpm文件

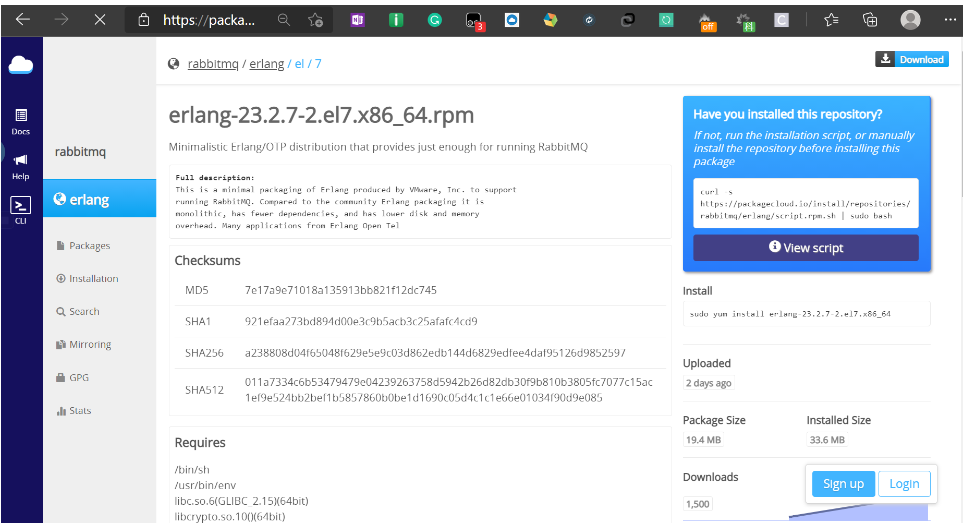


## 下载Erlang

Erlang和RabbitMQ版本对照：<https://www.rabbitmq.com/which-erlang.html>

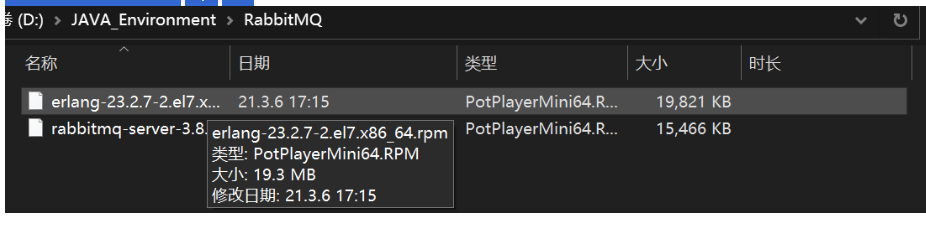


下载地址：<https://packagecloud.io/rabbitmq/erlang/packages/el/7/erlang-23.2.7-2.el7.x86_64.rpm>



其中的el7表示Red Hat 7.x，即CentOS 7.x

点击右上角下载即可得到.rpm文件

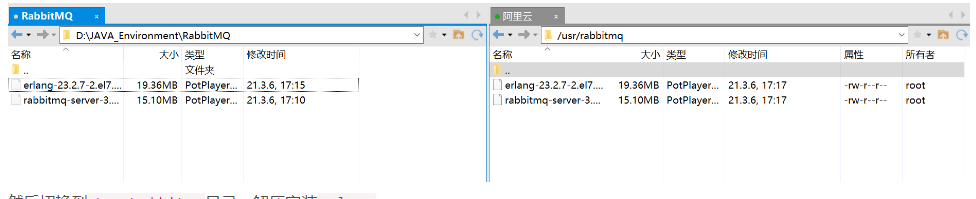


## 安装Erlang

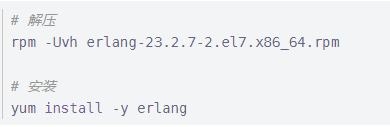
首先将下载好的文件上传到服务器，创建一个文件夹用来存放文件。

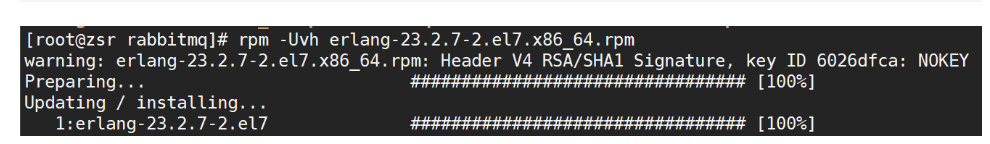


再利用xftp工具将上述下载的两个.rpm文件上传到服务器的刚创建的文件夹中。



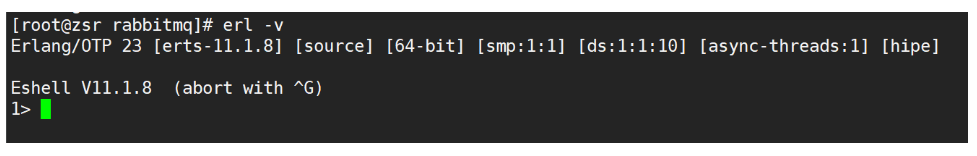
后切换到/usr/rabbitmq目录，解压安装erlang





安装完成后输入如下指令查看版本号

erl -v

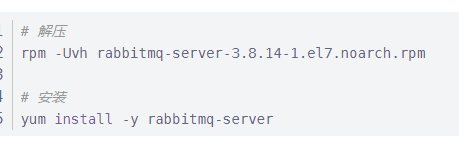


## 安装RabbitMQ

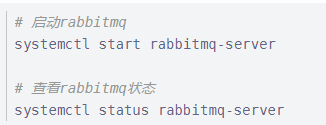
在RabiitMQ安装过程中需要依赖socat插件，首先安装该插件



然后解压安装RabbitMQ的安装包



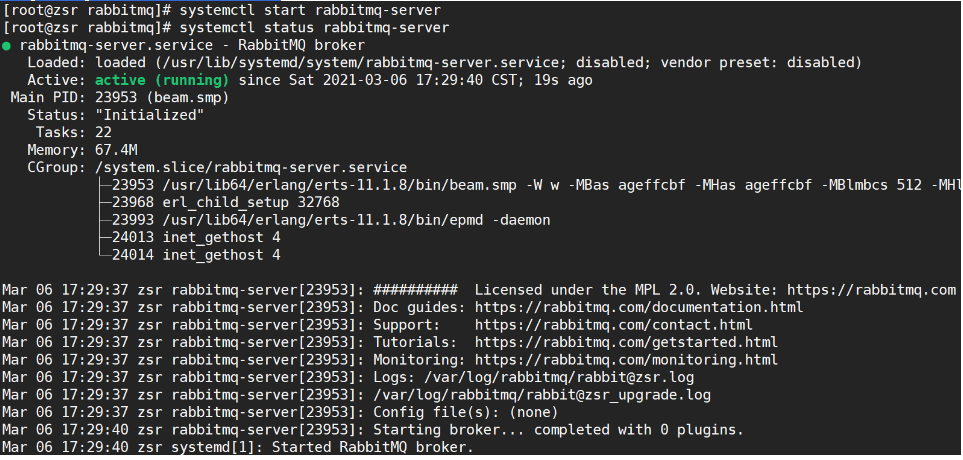
## 启动RabbitMQ服务



systemctl start rabbitmq-server

systemctl status rabbitmq-server

显示active则表示服务安装并启动成功



## 命令

### 启动rabbitmq

systemctl start rabbitmq-server

### 查看rabbitmq状态

systemctl status rabbitmq-server

### 设置rabbitmq服务开机自启动

systemctl enable rabbitmq-server

### 关闭rabbitmq服务

systemctl stop rabbitmq-server

### 重启rabbitmq服务

systemctl restart rabbitmq-server

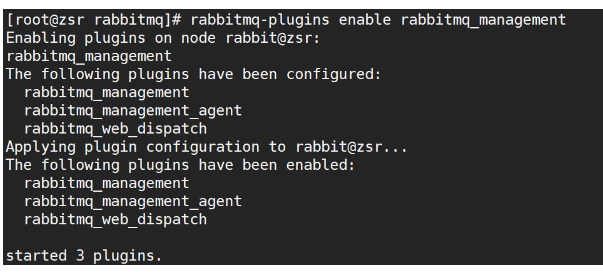
## RabbitMQWeb管理界面及授权操作

### 安装启动RabbitMQWeb管理界面

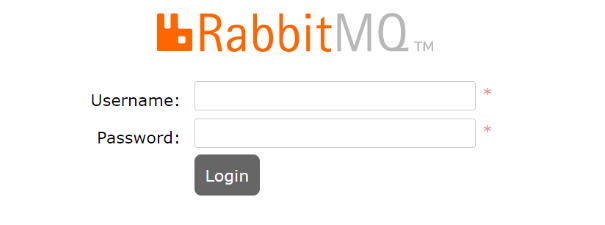
默认情况下，rabbitmq没有安装web端的客户端软件，需要安装才可以生效

# 打开RabbitMQWeb管理界面插件

rabbitmq-plugins enable rabbitmq\_management



然后我们打开浏览器，访问服务器公网ip:15672（注意打开阿里云安全组以及防火墙的15672端口），就可以看到管理界面。



rabbitmq有一个默认的账号密码guest，但该情况仅限于本机localhost进行访问，所以需要添加一个远程登录的用户。

### 添加远程用户



rabbitmqctl add\_user 用户名 密码

rabbitmqctl set\_user\_tags 用户名 administrator

rabbitmqctl set\_permissions -p / 用户名 “.\*” “.\*” “.\*”

#### 角色有四种

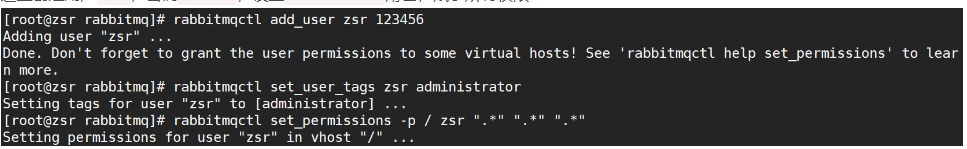
administrator：可以登录控制台、查看所有信息、并对rabbitmq进行管理

monToring：监控者；登录控制台，查看所有信息

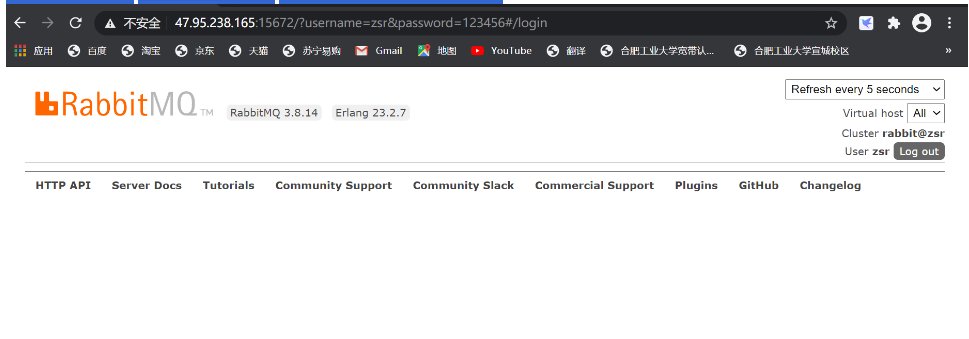
policymaker：策略制定者；登录控制台指定策略

managment：普通管理员；登录控制

这里创建用户zsr，密码123456，设置adminstator角色，赋予所有权限



创建完成后，访问服务器公网ip:15672进行登录，然后便可进入到后台。



### 其他指令

#### 修改密码

rabbitmqctl change\_ password 用户名 新密码

#### 删除用户

rabbitmqctl delete\_user 用户名

#### 查看用户清单

rabbitmqctl list\_users

# linux怎么卸载rabbitmq

https://blog.csdn.net/m0\_37679452/article/details/104778978

# linux进入erl怎么退出

https://blog.csdn.net/cnxieyang/article/details/52710967

# rabbitmq-server.service

# 控制台地址

http://localhost:15672/#/

# 网站端口号

15672

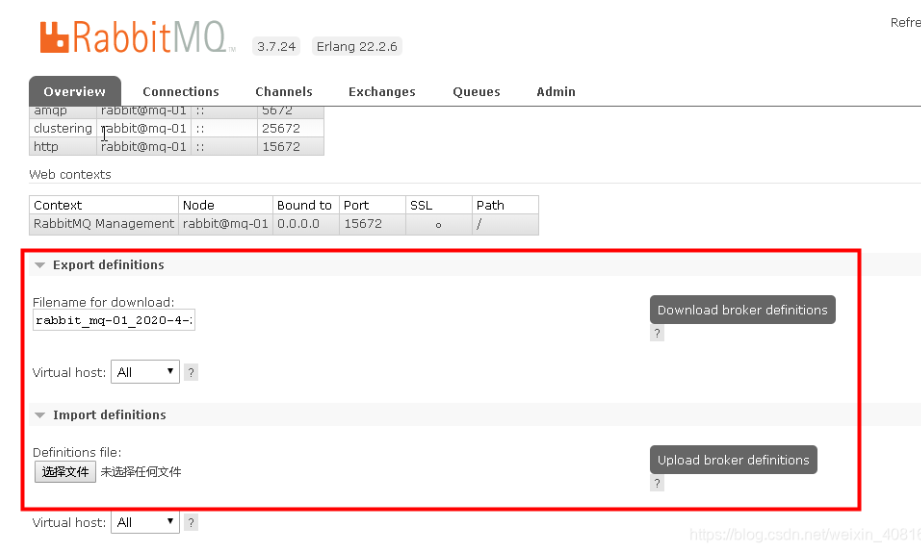
# rabbitmq控制台使用教程

https://blog.csdn.net/qq\_36551991/article/details/107166065

# RabbitMQ 控制台导入导出队列

1、登录rabbitmq控制台

2、执行下面的操作：



# 默认端口

5672

# RabbitMQ优缺点

# Kafka、ActiveMQ、RabbitMQ RocketMQ 有什么优缺点?



1、一般的业务系统要引入 MQ，最早大家都用 ActiveMQ，但是现在确实大家用的不多了，没经过大规模吞吐量场景的验证，社区也不是很活跃，所以大家还是算了吧，我个人不推荐用这个了。

2、后来大家开始用 RabbitMQ，但是确实 erlang 语言阻止了大量的 Java 程师去深入研究和掌控它，对公司而言，几乎处于不可控的状态，但是确实人家是开源的，比较稳定的支持，活跃度也高。

3、不过现在确实越来越多的公司会去用 RocketMQ，确实很不错，毕竟是阿里出品，但社区可能有突然黄掉的风险 (目前 RocketMQ 已捐给 ApacheGitHub 上的活跃度其实不算高) 对自己公司技术实力有绝对自信的，推荐用RocketMQ，否则回去老老实实用 RabbitMQ 吧，人家有活跃的开源社区，绝对不会黄。

4、所以中小型公司，技术实力较为一般，技术挑战不是特别高，用 RabbitMQ是不错的选择;大型公司，基础架构研发实力较强，用 RocketMQ 是很好的选择。

5、如果是大数据领域的实时计算、日志采集等场景，用 Kafka 是业内标准的绝对没问题，社区活跃度很高，绝对不会黄，何况几乎是全世界这个领域的事实性规范。

# 消息队列优点

## 解耦

传统模式的缺点：系统间耦合性太强，例如系统A在代码中直接调用系统B和系统C的代码，如果将来D系统接入，系统A还需要修改代码，过于麻烦。中间件模式的的优点：将消息写入消息队列，需要消息的系统自己从消息队列中订阅，从而系统A不需要做任何修改。

## 异步

传统模式的缺点：一些非必要的业务逻辑以同步的方式运行，太耗费时间。中间件模式的的优点：将消息写入消息队列，非必要的业务逻辑以异步的方式运行，加快响应速度。

## 肖峰

并发量大的时候，所有的请求直接怼到数据库，造成数据库连接异常。

# 核心概念

## Connection（连接）

每个producer（生产者）或者consumer（消费者）要通过RabbitMQ发送与消费消息，首先就要与RabbitMQ建立连接，这个连接就是Connection。Connection是一个TCP长连接。

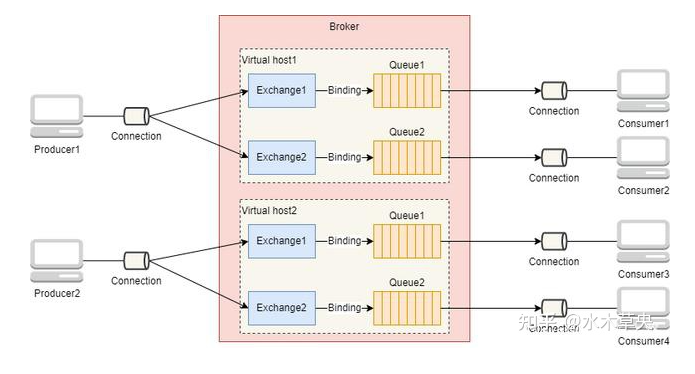
## Channel（信道）

Channel是在Connection的基础上建立的虚拟连接，RabbitMQ中大部分的操作都是使用Channel完成的，比如：声明Queue、声明Exchange、发布消息、消费消息等。

既然已经有了Connection，我们完全可以使用Connection完成Channel的工作，为什么还要引入Channel这样一个虚拟连接的概念呢？因为现在的程序都是支持多线程的，如果没有Channel，那么每个线程在访问RabbitMQ时都要建立一个Connection这样的TCP连接，对于操作系统来说，建立和销毁TCP连接是非常大的开销，在系统访问流量高峰时，会严重影响系统性能。Channel就是为了解决这种问题，通常情况下，每个线程创建单独的Channel进行通讯，每个Channel都有自己的channel id帮助Broker和客户端识别Channel，所以Channel之间是完全隔离的。Connection与Channel之间的关系可以比作光纤电缆，如果把Connection比作一条光纤电缆，那么Channel就相当于是电缆中的一束光纤。

## Virtual host（虚拟主机）

Virtual host是一个虚拟主机的概念，一个Broker中可以有多个Virtual host，每个Virtual host都有一套自己的Exchange和Queue，同一个Virtual host中的Exchange和Queue不能重名，不同的Virtual host中的Exchange和Queue名字可以一样。这样不同的用户在访问同一个RabbitMQ Broker时，可以创建自己单独的Virtual host，然后在自己的Virtual host中创建Exchange和Queue，很好地做到了不同用户之间相互隔离的效果。



## Queue（队列）

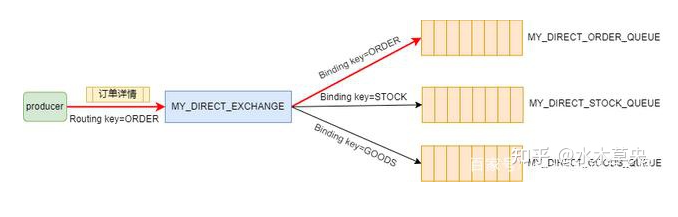
Queue是一个用来存放消息的队列，生产者发送的消息会被放到Queue中，消费者消费消息时也是从Queue中取走消息。

## Exchange（交换机）

Exchange是一个比较重要的概念，它是消息到达RabbitMQ的第一站，主要负责根据不同的分发规则将消息分发到不同的Queue，供订阅了相关Queue的消费者消费指定的消息。那Exchange有哪些分发消息的规则呢？这就要说到Exchange的4种类型了：direct、fanout、topic、headers。

### direct（精确匹配）

direct的意思是直接的，direct类型的Exchange会将消息转发到指定Routing key的Queue上，Routing key的解析规则为精确匹配，也就是只有当producer发送的消息的Routing key与某个Binding key相等时，消息才会被分发到对应的Queue上。



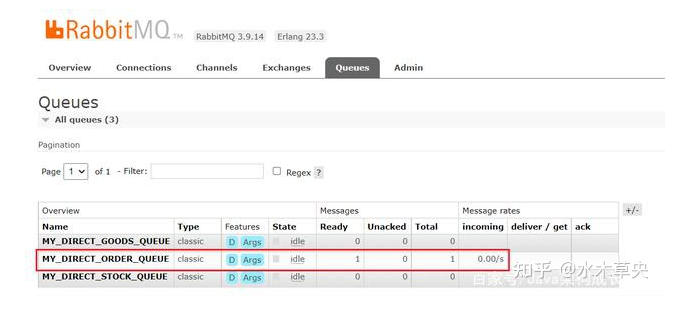
比如我们现在有一个direct类型的Exchange，它下面绑定了三个Queue，Binding key分别是ORDER/GOODS/STOCK：



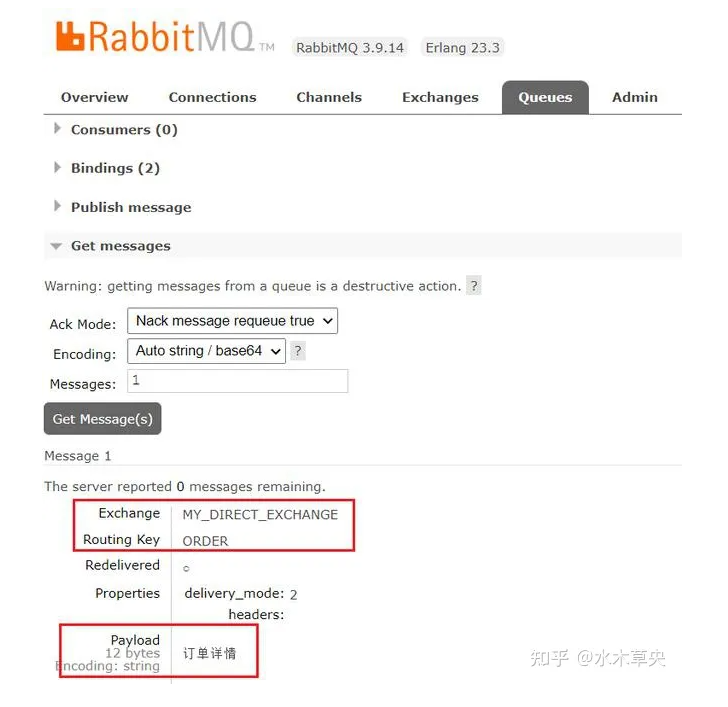
然后我们向该Exchange中发送一条消息，消息的Routing key是ORDER：



按照规则分析，这条消息应该被路由到MY\_EXCHANGE\_ORDER\_QUEUE这个Queue。消息发送成功之后，我们去Queues中查看，发现确实只有MY\_EXCHANGE\_ORDER\_QUEUE这个QUEUE接收到了一条消息。



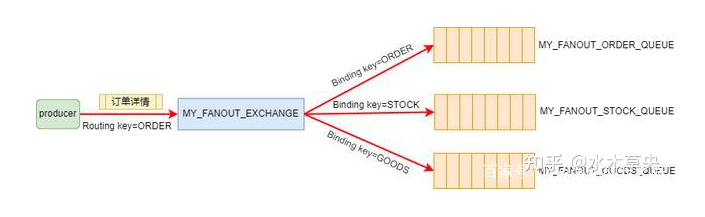
进入这个队列，通过getMessage取出消息查看，确实是我们刚才手动发送的那条消息。



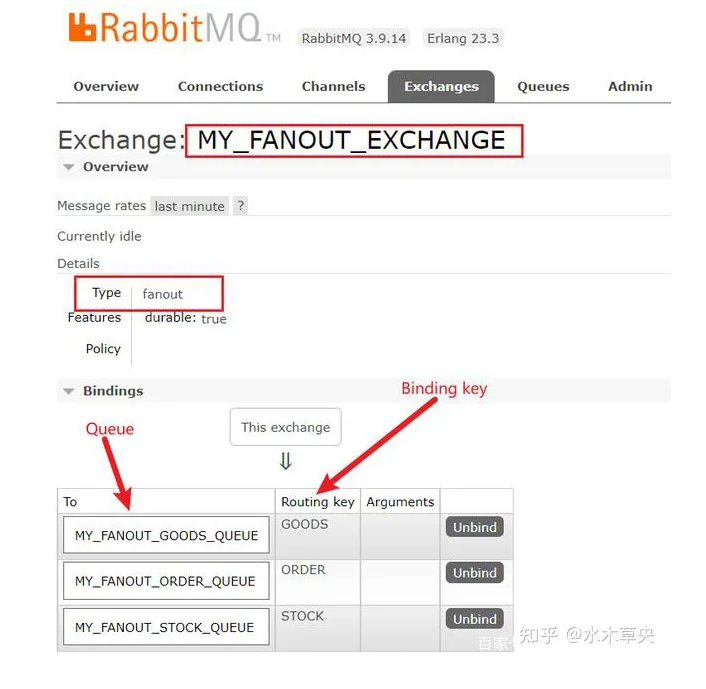
所以direct类型的Exchange在分发消息时，必须保证producer发送消息的Routing key与Exchange和Queue绑定的Binding key相等才可以。

### fanout（广播类型）

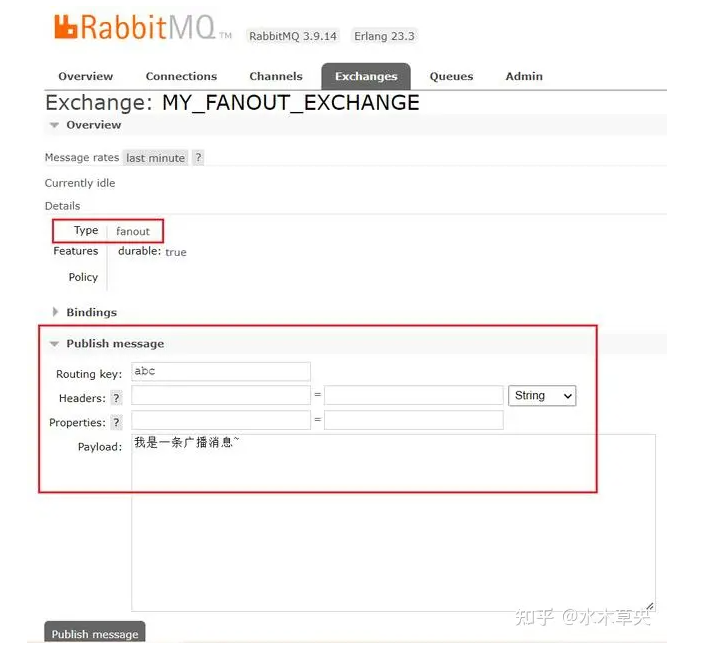
fanout是扇形的意思，该类型通常叫作广播类型。fanout类型的Exchange不处理Routing key，而是会将发送给它的消息路由到所有与它绑定的Queue上。



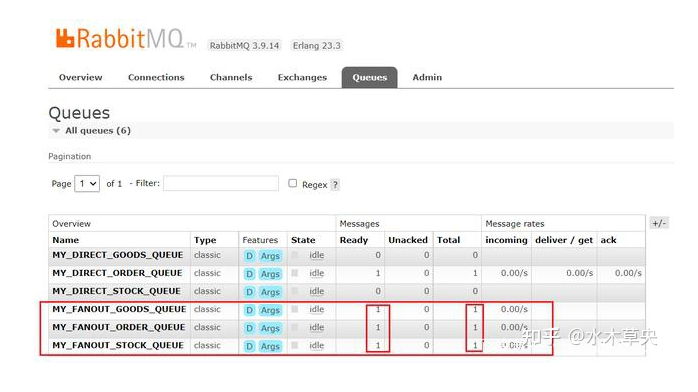
比如我们现在有一个fanout类型的Exchange，它下面绑定了三个Queue，Binding key分别是ORDER/GOODS/STOCK：



然后我们向该Exchange中发送一条消息，消息的Routing key随便填一个值abc：



按照规则分析，这条消息应该被路由到所有与该Exchange绑定的Queue，即三个Queue都应该会受到消息。消息发送成功之后，我们去Queues中查看，发现确实每个QUEUE都接收到了一条消息。



进入这三个QUEUE，通过getMessage取出消息查看，确实是我们刚才手动发送的那条消息。



所以，fanout类型的Exchange不管Routing key是什么，它都会将接收到的消息分发给所有与自己绑定了的Queue上。

### topic（主题）

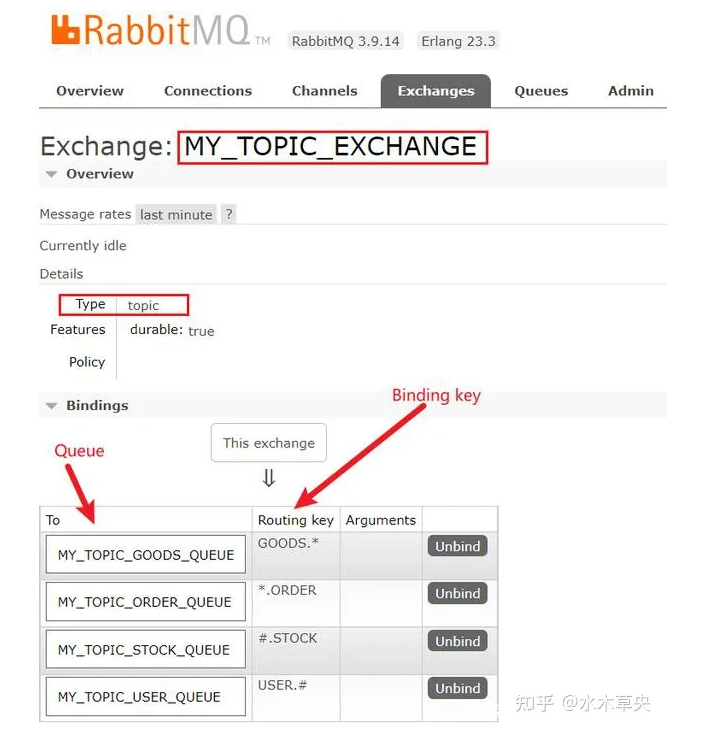
topic的意思是主题，topic类型的Exchange会根据通配符对Routing key进行匹配，只要Routing key满足某个通配符的条件，就会被路由到对应的Queue上。通配符的匹配规则如下：

Routing key必须是一串字符串，每个单词用“.”分隔。

符号“#”表示匹配一个或多个单词；

符号“\*”表示匹配一个单词。

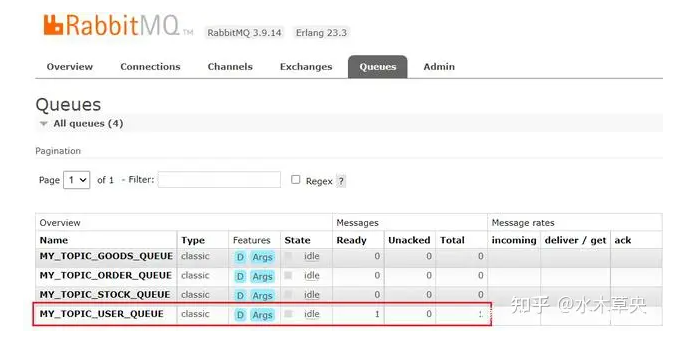
例如：“\*.123” 能够匹配到 “abc.123”，但匹配不到 “abc.def.123”；“#.123” 既能够匹配到 “abc.123”，也能匹配到 “abc.def.123”。  
比如我们现在有一个topic类型的Exchange，它下面绑定了4个Queue，Binding key分别是 \*.ORDER、GOODS.\*、#.STOCK、USER.#。



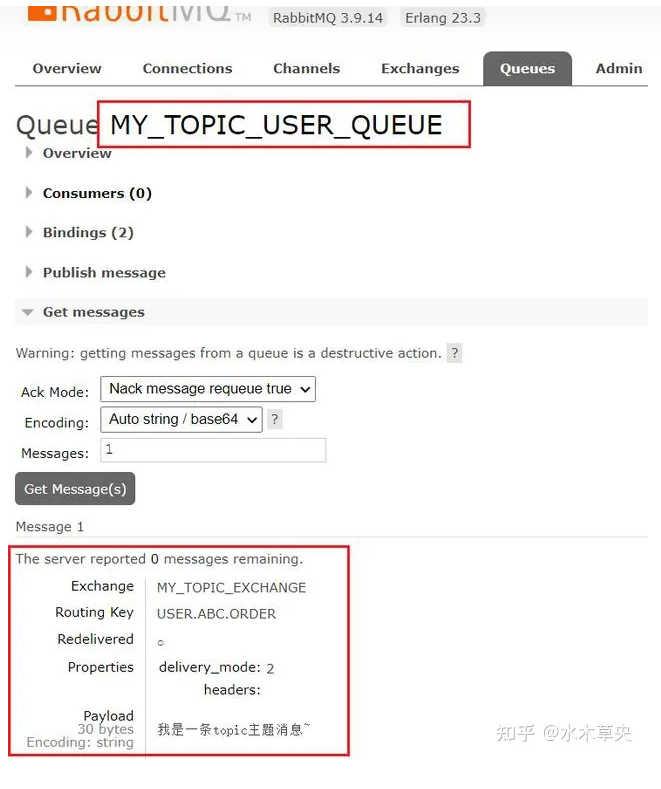
然后我们向该Exchange中发送一条消息，消息的Routing key为：USER.ABC.ORDER。



按照规则分析，USER.ABC.ORDER这个Routing key只可以匹配到 “USER.#” ，所以，这条消息应该被路由到MY\_TOPIC\_USER\_QUEUE这个Queue中。消息发送成功之后，我们去Queues中查看，发现结果符合我们的预期。



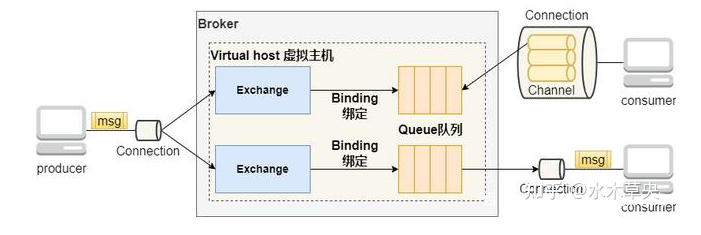
进入这个QUEUE，通过getMessage取出消息查看，确实是我们刚才手动发送的那条消息。



### headers（这个基本不用）

headers Exchange中，Exchange与Queue之间的绑定不再通过Binding key绑定，而是通过Arguments绑定。

# 工作原理



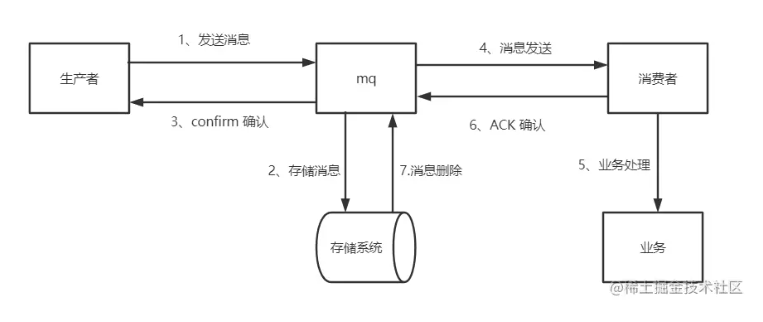
# 消息确认机制

## 基本概念

在mq 中，消费者和生产者并不直接进行通信，生产者只负责把消息发送到队列，消费者只负责从队列获取消息（不管是push还是pull）。

消费者从队列中获取到消息之后，这条消息就不存在队列中了，但是如果此时消费者所在的信道因为网络中断没有消费到，那这条消息就被永远的丢失了，所以我们希望等待消费者成功消费掉这个消息之后再删除这条消息。而在发送消息的时候也是这样的，生产者发消息给交换机，也不能保证消息准确发送过去了，消息就像石沉大海一样，所以这样需要一个消息确认。

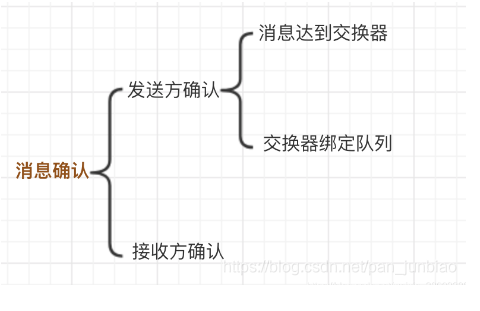
## 消息确认流程



在流程图中，我们可与看到消息确认是分为生产者确认和消费者确认的。

说明：在RabbitMQ 中有两种事务机制来确保消息的安全送达，分别是事务机制和确认机制。事务机制需要每个消息或每组消息发布提交的通道设置为事务性的，非常耗费性能，降低了消息吞吐量。因此实际中通常采用确认机制即可。

其中发送方确认又分为：生产者到交换器到确认、交换器到队列的确认。如下图：



## 消息发送确认

# 优先级队列

# 枚举

## AcknowledgeMode

# 接口

## Channel

### 基本概念

Channel 接口是 RabbitMQ 提供的一个通信接口，用于在应用程序和 RabbitMQ 服务器之间进行消息传递。它提供了一个操作消息队列的抽象层，允许应用程序与 RabbitMQ 进行交互。

### 作用

#### 1、建立和维护连接

通过 Channel 接口，应用程序可以与 RabbitMQ 建立连接，并保持连接状态。一个应用程序可以同时创建多个 Channel，每个 Channel 都可以独立地发送和接收消息，这样可以实现更高效的消息处理。

#### 2、创建、声明和删除队列

通过 Channel 接口，应用程序可以创建、声明和删除队列。队列是存储消息的容器，应用程序可以通过队列进行消息的发布和接收。

#### 3、发布和消费消息

通过 Channel 接口，应用程序可以向队列中发布消息，或者从队列中消费消息。应用程序可以指定消息的属性（例如路由键、优先级等），并根据特定的规则进行消息的投递和消费。

#### 4、设置和绑定交换器

通过 Channel 接口，应用程序可以设置和绑定交换器。交换器用于将发布的消息路由到一个或多个队列。应用程序可以选择不同的交换器类型（例如直连交换器、主题交换器等），以满足不同的消息路由需求。

#### 5、确认消息的投递

#### 6、实现事务机制

## ChannelAwareMessageListener

# 类

## ExchangeBuilder

## RabbitTemplate

### 基本概念

RabbitTemplate 类是 RabbitMQ 客户端库中的一个核心类，用于在 Java 应用程序与 RabbitMQ 消息代理之间进行消息发送和接收操作。

RabbitTemplate 类的主要作用是简化 RabbitMQ 的使用，提供了一组方法来发送和接收消息，以及处理消息的确认、回退和重试等机制。它封装了与 RabbitMQ 交互的底层细节，使得开发者可以更方便地与 RabbitMQ 进行通信。

### 方法

#### public void convertAndSend(String exchange, String routingKey, final Object object)

将消息转换为指定格式，并发送到指定的交换器和路由键。可以用于发送任何类型的消息对象。

## Envelope

## DirectExchange

## SimpleRabbitListenerContainerFactory

# 注解

## @EnableRabbit

### 基本概念

@EnableRabbit 是 Spring AMQP 提供的一个注解，用于启用 RabbitMQ 的功能支持。通过在 Spring Boot 项目中添加 @EnableRabbit 注解，可以自动配置 RabbitMQ 的相关组件和功能，使得我们可以方便地使用 RabbitMQ 进行消息的发送、接收和处理。

### 作用

#### 1、****自动配置 RabbitMQ 连接工厂（ConnectionFactory）****

#### **2、**自动配置 RabbitMQ 模板（RabbitTemplate）****

#### **3、**自动配置 RabbitMQ 监听容器（SimpleMessageListenerContainer）****

#### **4、**自动注册 RabbitMQ 注解驱动监听器（RabbitListener）****

## @RabbitHandler

### 基本概念

用于标识消费者方法。在 RabbitMQ 中，消息的消费者通常需要定义一个或多个处理消息的方法。这些方法需要用特定的注解进行标识，以告诉 RabbitMQ 框架如何将接收到的消息分发给对应的方法进行处理。@RabbitHandler 就是其中的一种注解。

### 作用

## @RabbitListener

### 基本概念

用于将一个方法标记为 RabbitMQ 的消息监听器。

### 作用

#### 1、定义消息监听器

通过在方法上添加 @RabbitListener 注解，将该方法声明为 RabbitMQ 的消息监听器。当监听到指定队列中有消息到达时，框架会自动调用被注解的方法进行消息处理。

#### 2、指定监听的队列

通过配置 @RabbitListener 注解的参数，可以指定需要监听的队列。可以使用队列名称或者队列的 Bean 对象作为参数值。例如，@RabbitListener(queues = "myQueue")

指定监听名为 "myQueue" 的队列。

#### 3、处理消息

被 @RabbitListener 注解的方法需要定义一个消息参数，用于接收监听到的消息。方法可以根据实际需求自定义参数类型，如 String、byte[]、自定义对象等。当消息到达时，框架会自动将消息内容转换为相应的参数类型，并传递给被注解的方法进行处理。

#### 4、处理消费确认

默认情况下，Spring AMQP 框架会在方法成功处理完消息后自动发送消费确认（ACK）给 RabbitMQ 服务器，告知消息已被消费。这样可以确保消息不会被重复消费。如果方法抛出异常，框架会发送消费拒绝（NACK），并根据具体配置进行重试或将消息投递到死信队列。

### 属性

#### bindings（重点）

用于绑定消息队列和交换机的配置项。

作用：bindings 属性用于将消息队列和交换机进行绑定，以便消息能够从指定的队列中接收到，这样 @RabbitListener 所标识的方法或类就能够监听并处理这些消息。

配置方式：

bindings 属性使用 @QueueBinding 注解进行配置，@QueueBinding 注解包含多个属性，用于指定队列、交换机和绑定键等信息。

#### queues

#### concurrency

指定并发消费者的数量，可以平行处理多个消息。

#### errorHandler

自定义异常处理器，用于处理消费过程中的异常情况。

#### ackMode

指定消费确认模式，如自动确认、手动确认等。

#### containerFactory

指定使用的容器工厂，可以用于配置监听器的线程池、拦截器等。

## @QueueBinding

### 基本概念

用于配置队列绑定关系的注解。它通常与 @RabbitListener 注解一起使用，用于指示消息队列和交换机的绑定方式。

### 作用

用于配置消息队列和交换机之间的绑定关系，以确保消息能够正确地路由到指定的队列。通过 @QueueBinding 注解，可以将队列、交换机和绑定键等相关属性进行配置。

### 属性

#### exchange

用于指定要绑定的交换机。

#### key

用于指定绑定键，用于确保消息能够路由到正确的队列。

#### value

用于指定要绑定的队列。

### 总结

@QueueBinding 注解用于配置消息队列和交换机之间的绑定关系。通过配置 @QueueBinding 注解，可以指定要绑定的队列、交换机和绑定键等信息，确保消息能够正确地路由到指定的队列。

## @Queue

## @Exchange

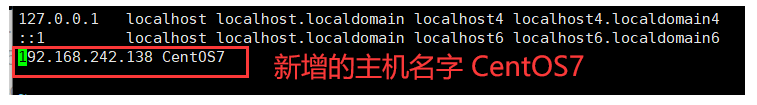
# 问题

## 1、Linux启动RabbitMQ出现错误：ERROR: epmd error for host 192: badarg (unknown POSIX error)

解决方法：

这是因为我们的主机名首字母是数字，需要修改主机名

然后使用vim /etc/hosts修改hosts文件



重启rabbitmq服务，查看一下rabbitmq的状态



