# RabbitMQ

## 1.什么是MQ

* 消息队列（Message Queue，简称MQ），从字面意思上看，本质是个队列，FIFO先入先出，只不过队列中存放的内容是message而已。
* 其主要用途：不同进程Process/线程Thread之间通信。

为什么会产生消息队列？有几个原因：

* 不同进程（process）之间传递消息时，两个进程之间耦合程度过高，改动一个进程，引发必须修改另一个进程，为了隔离这两个进程，在两进程间抽离出一层（一个模块），所有两进程之间传递的消息，都必须通过消息队列来传递，单独修改某一个进程，不会影响另一个；
* 不同进程（process）之间传递消息时，为了实现标准化，将消息的格式规范化了，并且，某一个进程接受的消息太多，一下子无法处理完，并且也有先后顺序，必须对收到的消息进行排队，因此诞生了事实上的消息队列；
* 关于消息队列的详细介绍请参阅：

《Java帝国之消息队列》

《一个故事告诉你什么是消息队列》

《到底什么时候该使用MQ》

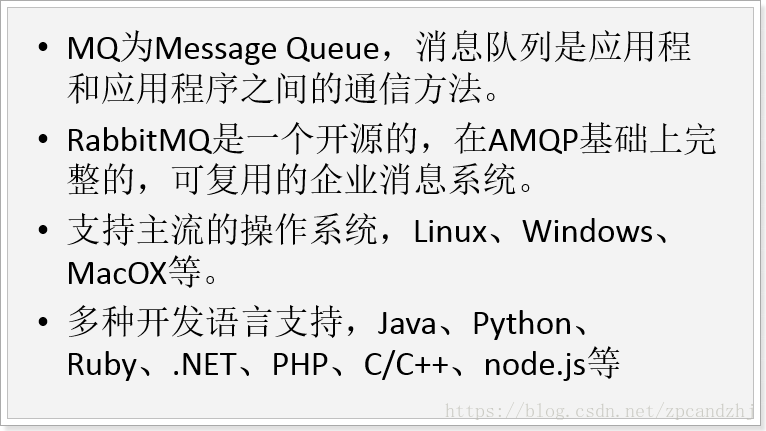
* MQ框架非常之多，比较流行的有RabbitMq、ActiveMq、ZeroMq、kafka，以及阿里开源的RocketMQ。本文主要介绍RabbitMq。
* 本教程pdf及代码下载地址：

代码：https://download.csdn.net/download/zpcandzhj/10585077

教程：https://download.csdn.net/download/zpcandzhj/10585092

## 2.RabbitMQ

### 2.1.RabbitMQ的简介



开发语言：Erlang – 面向并发的编程语言。

#### 2.1.1.AMQP

AMQP是消息队列的一个协议。

### 2.2.官网

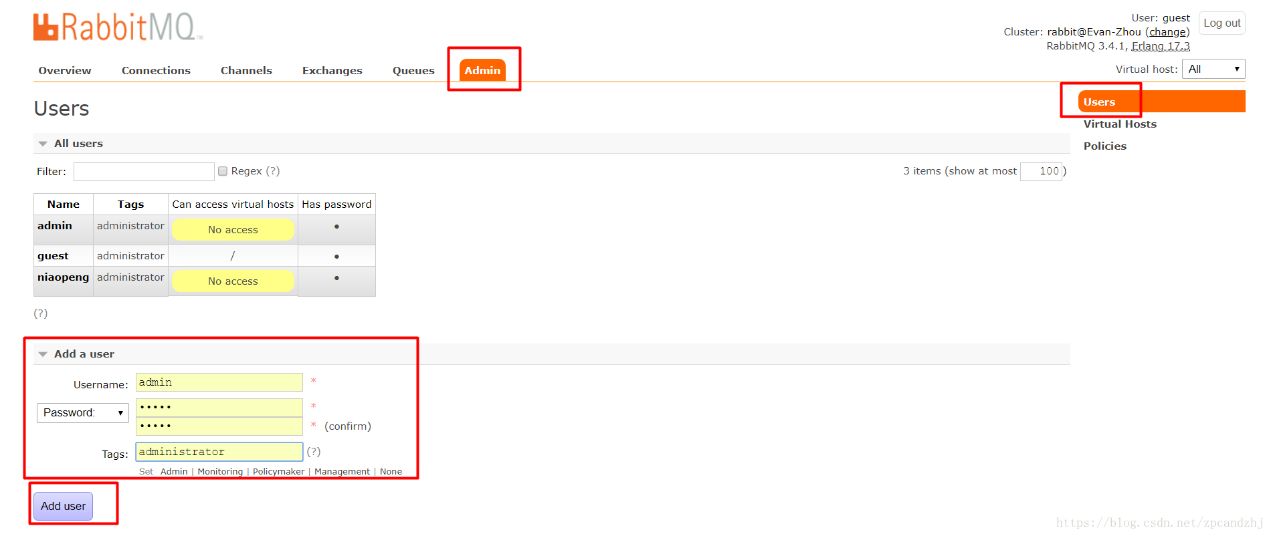
<https://www.rabbitmq.com/#features>

## 3.搭建RabbitMQ环境

<https://blog.csdn.net/hellozpc/article/details/81436980>

## 4.添加用户

### 4.1.添加admin用户



### 4.2.用户角色

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rabbitmq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(上图红框标识的部分)。

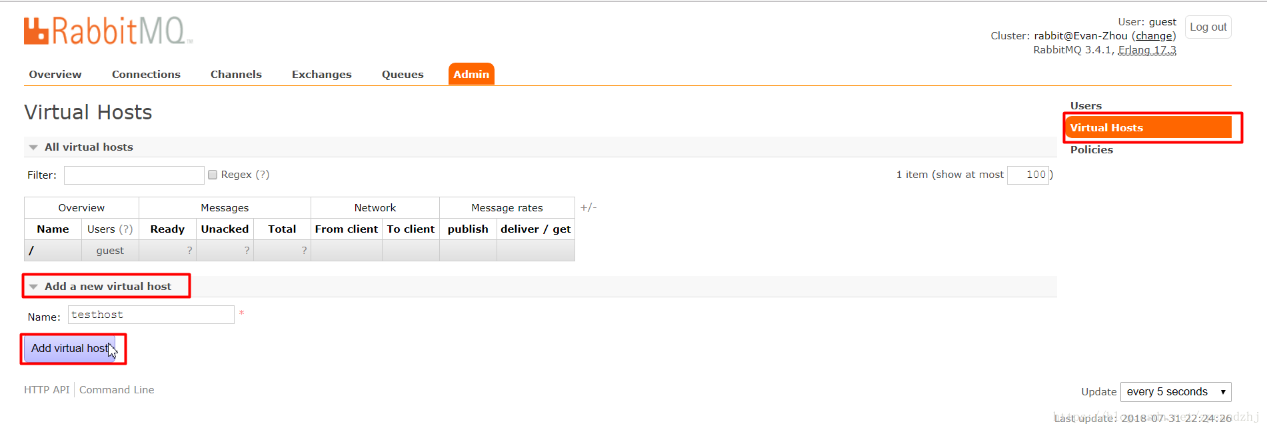
4、普通管理者(management)

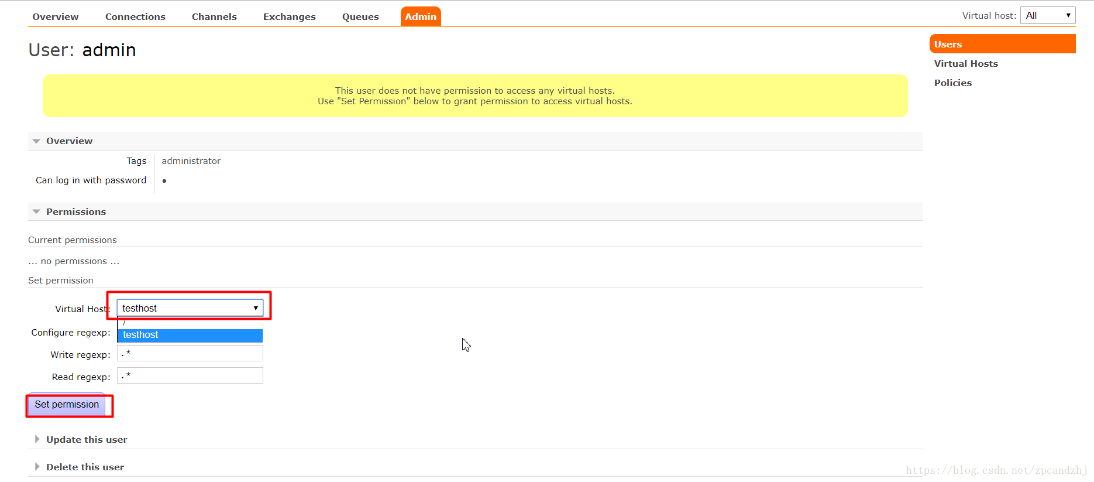
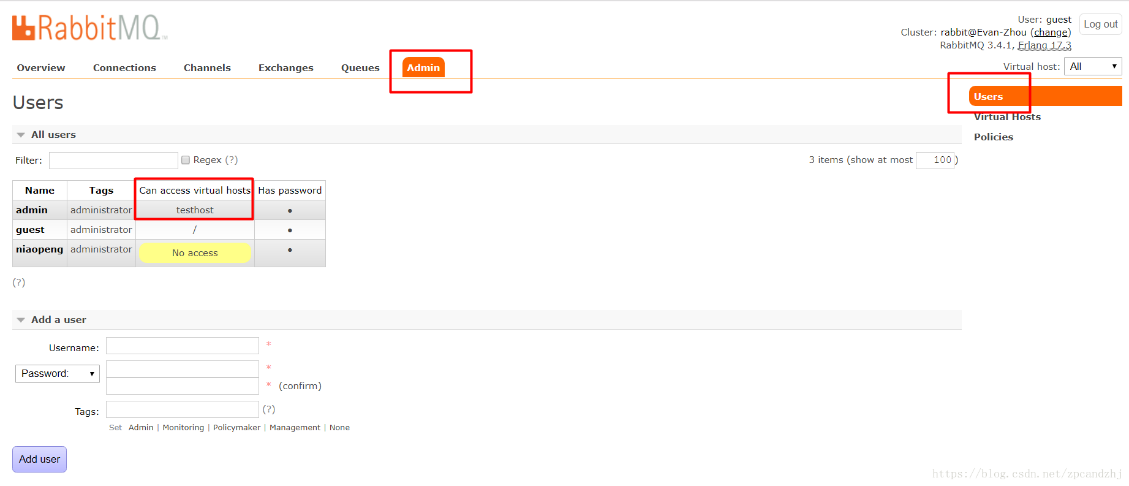
仅可登陆管理控制台，无法看到节点信息，也无法对策略进行管理。

5、其他

无法登陆管理控制台，通常就是普通的生产者和消费者。

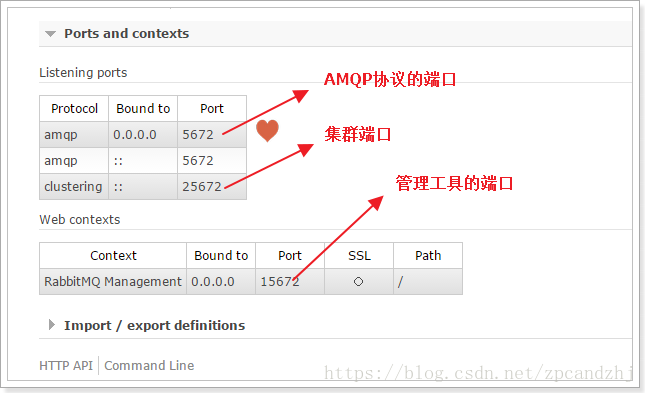
### 4.3.创建Virtual Hosts



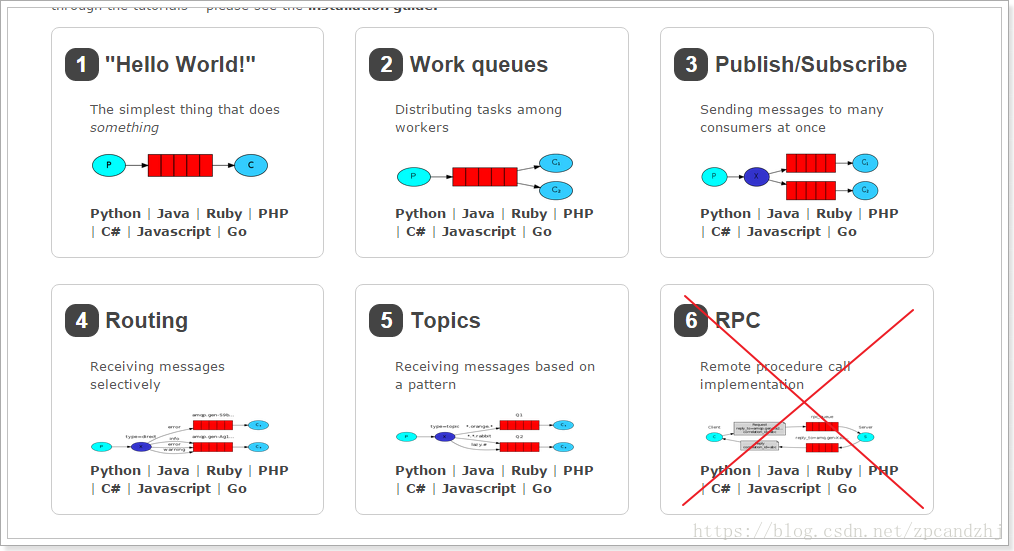
选中Admin用户，设置权限：  
  
看到权限已加：  


### 4.4.管理界面中的功能





## 5.五种队列

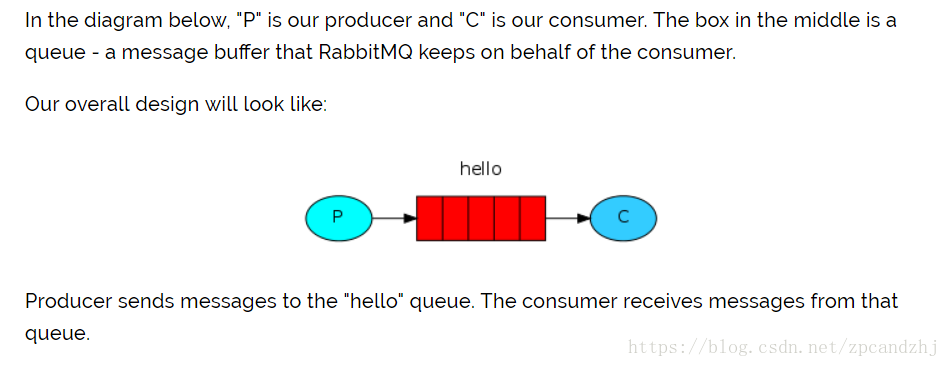


### 5.1.导入my-rabbitmq项目

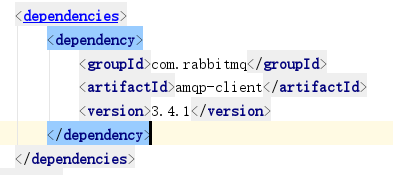
### 5.2.简单队列

#### 5.2.1.图示

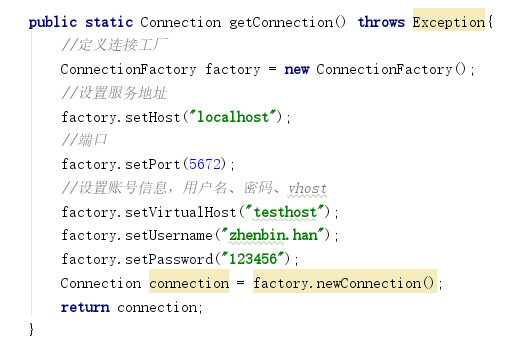
在下面的图表中，“P”是我们的生产者，“C”是我们的消费者。中间的框是一个队列—RabbitMQ代表消费者保存的消息缓冲区。我们的整体设计如下:



生产者向“hello”队列发送消息。使用者从该队列接收消息。



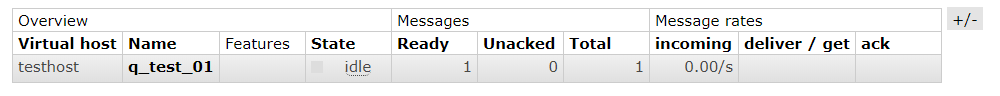
获取MQ的连接工具类

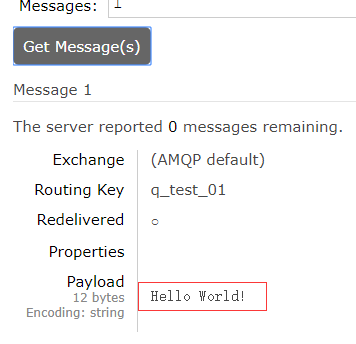


生产者发送消息到队列

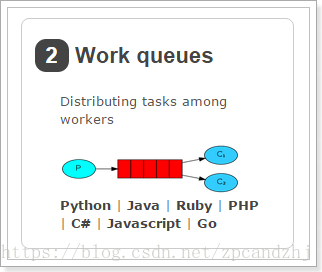
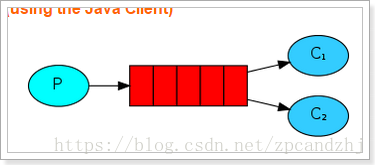


管理工具中查看消息



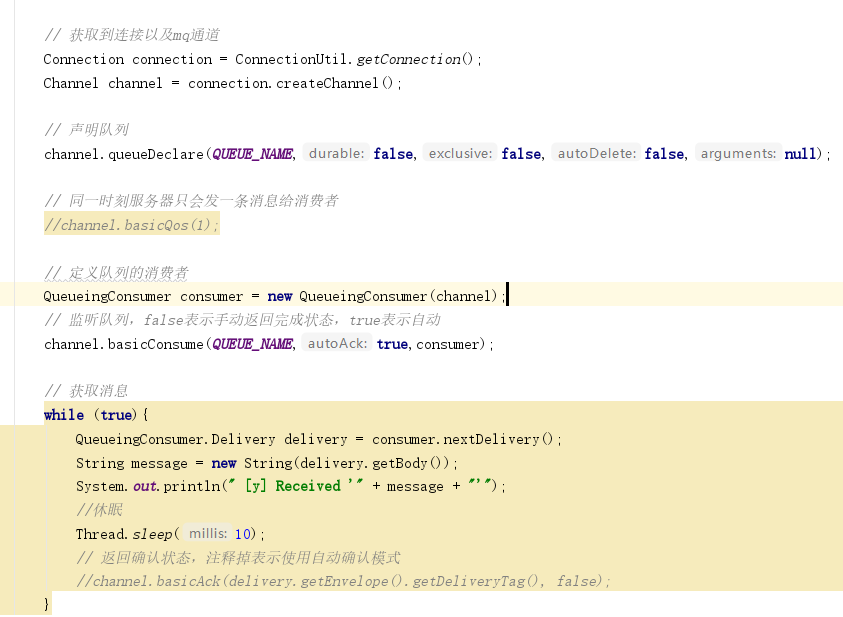


### 5.3.Work模式

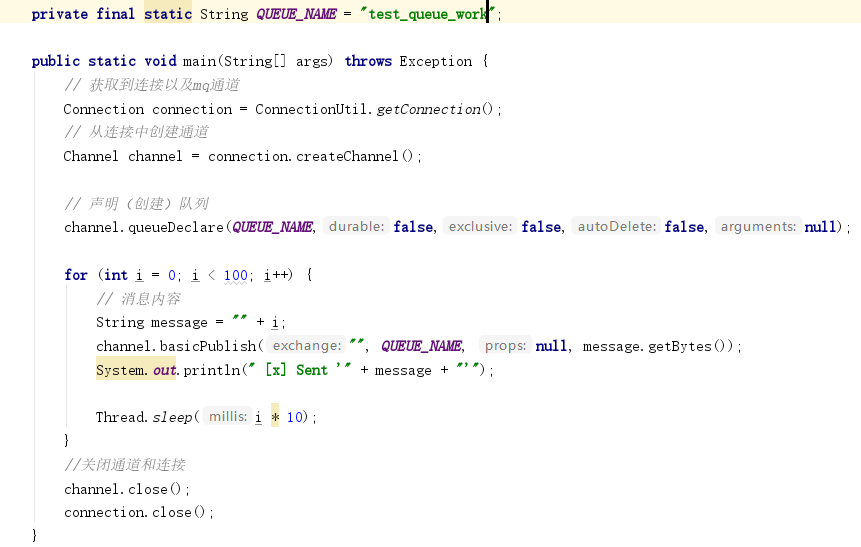
 

一个生产者、2个消费者。一个消息只能被一个消费者获取。

消费者两个



生产者一个



2个概念

* 轮询分发 ：使用任务队列的优点之一就是可以轻易的并行工作。如果我们积压了好多工作，我们可以通过增加工作者（消费者）来解决这一问题，使得系统的伸缩性更加容易。在默认情况下，RabbitMQ将逐个发送消息到在序列中的下一个消费者(而不考虑每个任务的时长等等，且是提前一次性分配，并非一个一个分配)。平均每个消费者获得相同数量的消息。这种方式分发消息机制称为Round-Robin（轮询）。
* 公平分发 ：虽然上面的分配法方式也还行，但是有个问题就是：比如：现在有2个消费者，所有的奇数的消息都是繁忙的，而偶数则是轻松的。按照轮询的方式，奇数的任务交给了第一个消费者，所以一直在忙个不停。偶数的任务交给另一个消费者，则立即完成任务，然后闲得不行。而RabbitMQ则是不了解这些的。这是因为当消息进入队列，RabbitMQ就会分派消息。它不看消费者为应答的数目，只是盲目的将消息发给轮询指定的消费者。

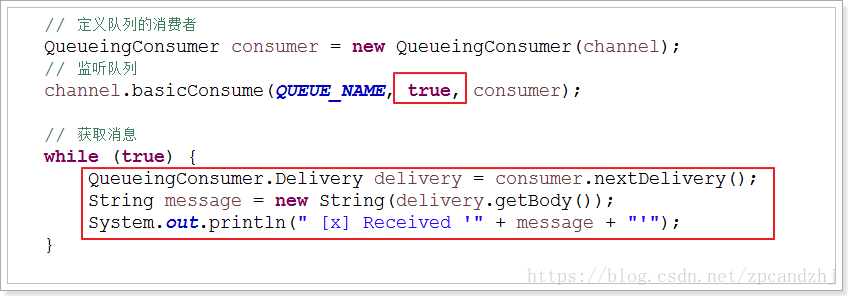
为了解决这个问题，我们使用basicQos( prefetchCount = 1)方法，来限制RabbitMQ只发不超过1条的消息给同一个消费者。当消息处理完毕后，有了反馈，才会进行第二次发送。还有一点需要注意，使用公平分发，必须关闭自动应答，改为手动应答。

### 5.5.消息的确认模式

消费者从队列中获取消息，服务端如何知道消息已经被消费呢？

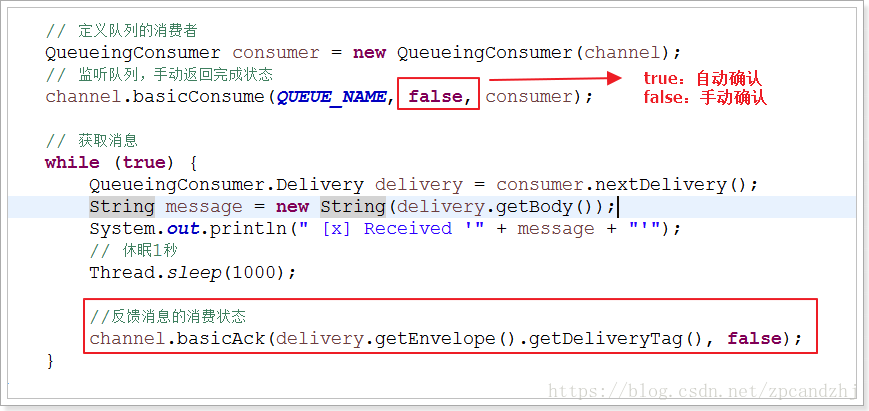
模式1：自动确认

只要消息从队列中获取，无论消费者获取到消息后是否成功消息，都认为是消息已经成功消费。

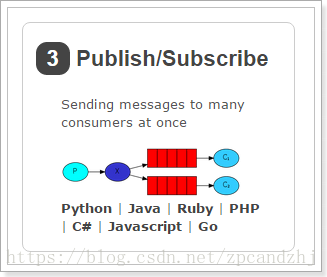


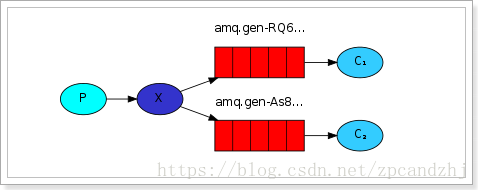
模式2：手动确认

消费者从队列中获取消息后，服务器会将该消息标记为不可用状态，等待消费者的反馈，如果消费者一直没有反馈，那么该消息将一直处于不可用状态。



### 5.6.订阅模式





解读：

1、1个生产者，多个消费者

2、每一个消费者都有自己的一个队列

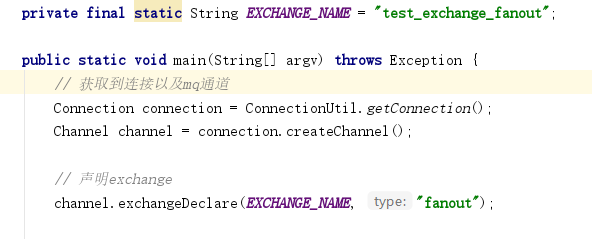
3、生产者没有将消息直接发送到队列，而是发送到了交换机

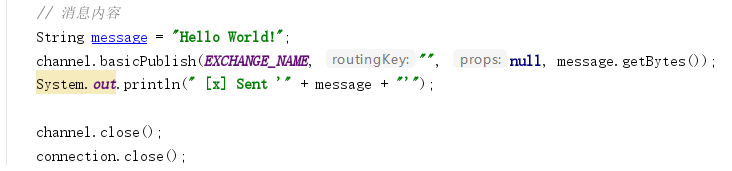
4、每个队列都要绑定到交换机

5、生产者发送的消息，经过交换机，到达队列，实现，一个消息被多个消费者获取的目的

注意：一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费

消息的生产者，向交换机中发送消息。

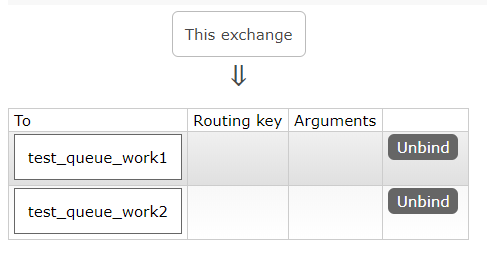




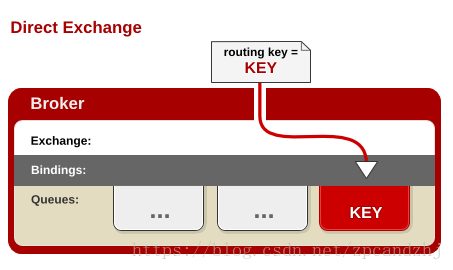
消费者

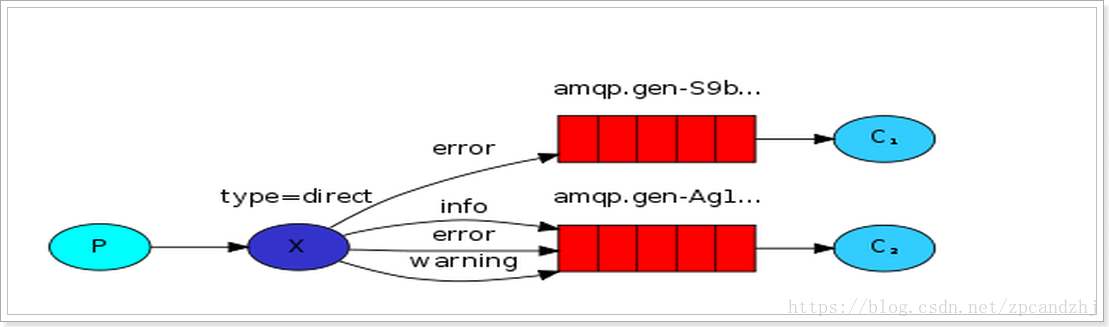


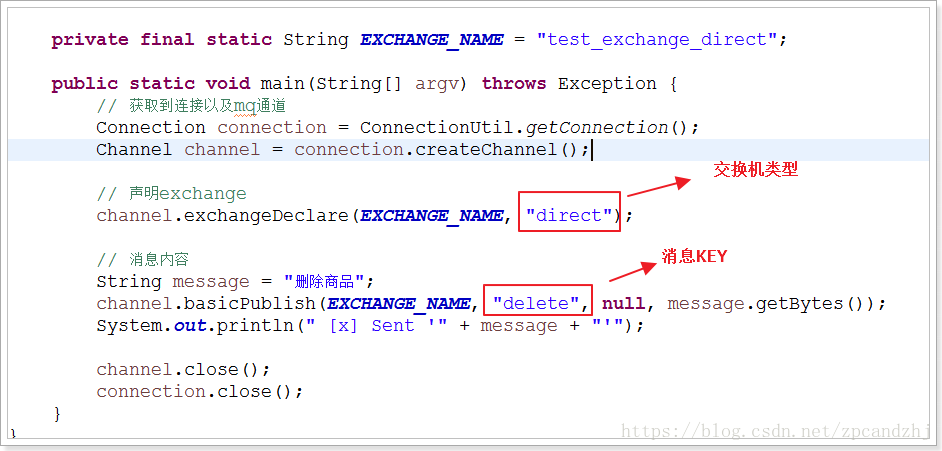
查看队列和交换机的绑定关系：

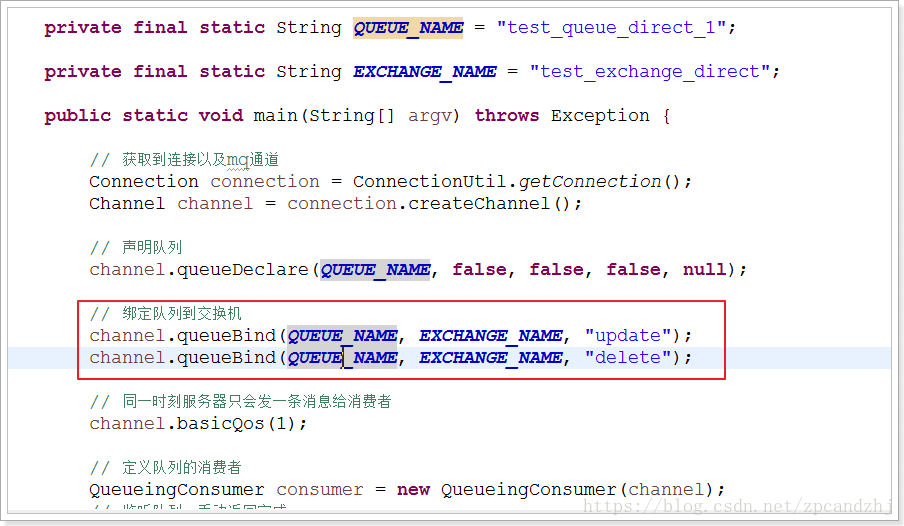


### 5.7.路由模式



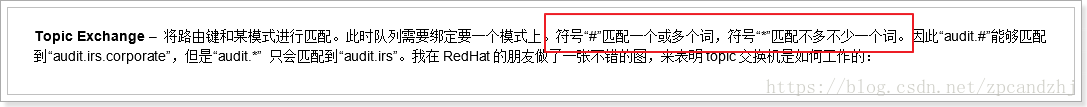


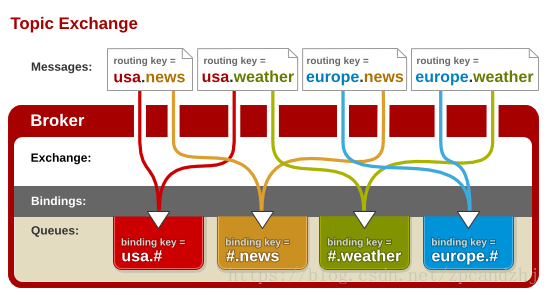


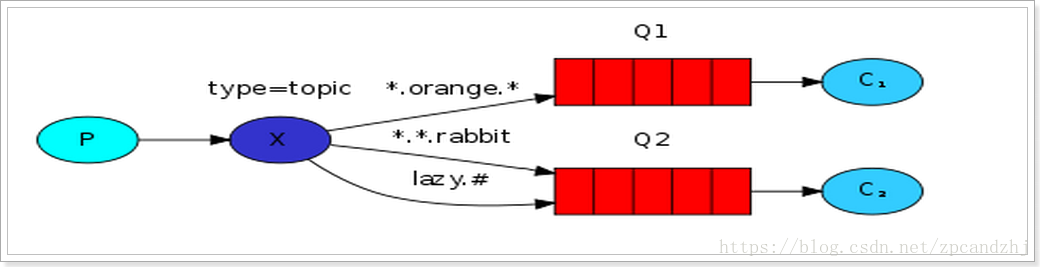




### 5.8.主题模式（通配符模式）

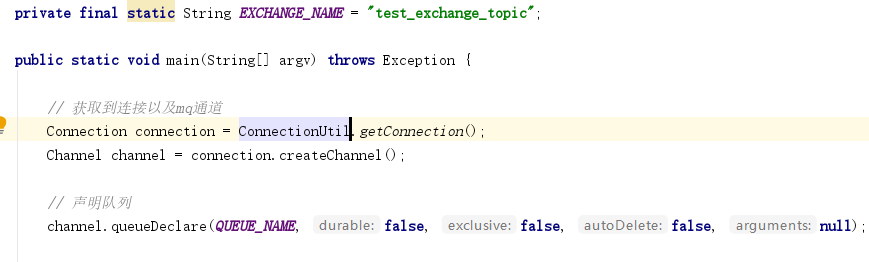






同一个消息被多个消费者获取。一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费到消息。







**Kafka**

Kafka 是一个分布式流式处理平台。活动流数据是几乎所有站点在对其网站使用情况做报表时都要用到的数据中最常规的部分。活动数据包括页面访问量（Page View）、被查看内容方面的信息以及搜索情况等内容。这种数据通常的处理方式是先把各种活动以日志的形式写入某种文件，然后周期性地对这些文件进行统计分析。运营数据指的是服务器的性能数据（CPU、IO 使用率、请求时间、服务日志等等数据)。运营数据的统计方法种类繁多。流平台具有三个关键功能：

* 消息队列：发布和订阅消息流，这个功能类似于消息队列，这也是 Kafka 也被归类为消息队列的原因。
* 容错的持久方式存储记录消息流： Kafka 会把消息持久化到磁盘，有效避免了消息丢失的风险·。
* 流式处理平台： 在消息发布的时候进行处理，Kafka 提供了一个完整的流式处理类库。

Kafka 将记录流（流数据）存储在 topic 中。每个记录由一个键、一个值、一个时间戳组成。Kafka 主要有两大应用场景：

* 消息队列：建立实时流数据管道，以可靠地在系统或应用程序之间获取数据。
* 数据处理：构建实时的流数据处理程序来转换或处理数据流。

**Kafka 消息模型**

**队列模型：早期的消息模型**

使用队列（Queue）作为消息通信载体，满足生产者与消费者模式，一条消息只能被一个消费者使用，未被消费的消息在队列中保留直到被消费或超时。

**发布-订阅模型:Kafka 消息模型**

发布-订阅模型主要是为了解决队列模型存在的问题。发布订阅模型（Pub-Sub） 使用主题（Topic） 作为消息通信载体，类似于广播模式；发布者发布一条消息，该消息通过主题传递给所有的订阅者，在一条消息广播之后才订阅的用户则是收不到该条消息的。在发布 - 订阅模型中，如果只有一个订阅者，那它和队列模型就基本是一样的了。所以说，发布 - 订阅模型在功能层面上是可以兼容队列模型的。Kafka 采用的就是发布 - 订阅模型。RocketMQ 的消息模型和 Kafka 基本是完全一样的。唯一的区别是 Kafka 中没有队列这个概念，与之对应的是 Partition（分区）。

**Kafka 重要概念解读**

Kafka 将生产者发布的消息发送到 Topic（主题） 中，需要这些消息的消费者可以订阅这些 Topic（主题）。Kafka 比较重要的几个概念：

* Producer（生产者）：产生消息的一方。
* Consumer（消费者）：消费消息的一方。
* Broker（代理）：可以看作是一个独立的 Kafka 实例。多个 Kafka Broker 组成一个 Kafka Cluster。

每个 Broker 中又包含了 Topic 以及 Partition 这两个重要的概念：

* Topic（主题）：Producer 将消息发送到特定的主题，Consumer 通过订阅特定的 Topic(主题) 来消费消息。
* Partition（分区）：Partition 属于 Topic 的一部分。一个 Topic 可以有多个 Partition ，并且同一 Topic 下的 Partition 可以分布在不同的 Broker 上，这也就表明一个 Topic 可以横跨多个 Broker 。

Kafka 为分区（Partition）引入了多副本（Replica）机制。分区（Partition）中的多个副本之间会有一个叫做 leader 的家伙，其他副本称为 follower。我们发送的消息会被发送到 leader 副本，然后 follower 副本才能从 leader 副本中拉取消息进行同步。（生产者和消费者只与 leader 副本交互。）

Kafka 的多分区（Partition）以及多副本（Replica）机制有什么好处呢？

* Kafka 通过给特定 Topic 指定多个 Partition, 而各个 Partition 可以分布在不同的 Broker 上, 这样便能提供比较好的并发能力（负载均衡）。
* Partition 可以指定对应的 Replica 数, 这也极大地提高了消息存储的安全性, 提高了容灾能力，不过也相应的增加了所需要的存储空间。

Kafka 如何保证消息的消费顺序？

每次添加消息到 Partition(分区) 的时候都会采用尾加法。Kafka 只能为我们保证 Partition(分区) 中的消息有序，而不能保证 Topic(主题) 中的 Partition(分区) 的有序。消息在被追加到 Partition(分区)的时候都会分配一个特定的偏移量（offset）。Kafka 通过偏移量（offset）来保证消息在分区内的顺序性。