# Exchange

## 类型

1. Direct Exchange

将消息中的Routing key与该Exchange关联的所有Binding中的Routing key进行比较，如果相等，则发送到该Binding对应的Queue中。

1. Fanout Exchange

直接将消息转发到所有binding的对应queue中，这种exchange在路由转发的时候，忽略Routing key。

1. Topic Exchange

将消息中的Routing key与该Exchange关联的所有Binding中的Routing key进行对比，如果匹配上了，则发送到该Binding对应的Queue中。Topic是模糊匹配

1. headers类型

将消息中的headers与该Exchange相关联的所有Binging中的参数进行匹配，如果匹配上了，则发送到该Binding对应的Queue中。

匹配规则：

如果Binding中的

x-match = all：表示所有的键值对都匹配才能转发到消息。

x-match = any: 表示只要有键值对匹配就能转发消息。

## 默认交换器

不指定交换器类型时，交换器默认是direct类型，其name属性为空。并且默认交换器有一个重要的特性：每个队列都会使用它的队列名字作为路由关键字（routing key）去自动地绑定到默认交换器上。

换言之，对于默认的交换器，无需用队列对其进行绑定操作，因为所有的队列都会自动的与之绑定。

但是，默认交换器与所有队列都进行绑定并不意味着交换器会将消息分发给所有的队列。交换器最终会将消息转发给哪个队列在生产者生产消息的时候就已经确定了——生产者生产消息时会指定消息路由关键字，这里的路由关键字即队列名。也就是说，消息会转发给与该路由关键字同名的队列中去。

# 队列（Queues）

## Name

队列名支持最多255字节的UTF-8字符。应用程序在声明队列的时候可以自己指定队列名，或者当应用程序指定name属性为空时，代理（broker）会自动地为其生成一个唯一的队列名。

需要注意的是，以”amq.”开头的队列名是由AMQP内部使用的命名前缀，请开发者不要使用，否则将抛出403异常。

## durability属性

durability属性对应两种情况，分别是durable（持久的）和transient（短暂的）。durable类型的队列会持久化至硬盘上，所以当代理（broker）重启之后，它依然存在。相反地，当代理重启之后，transient类型的队列就消失了。

需要注意的是，队列的持久化是相对队列而言，对存储在持久化队列中的消息来说，当代理重启之后：队列还存在、消息则不存在。

所以，当broker重启之后，如果想让消息仍然存在，这就是消息持久化机制干的事了，后面再说消息属性的相关内容

# AMQP

Advanced Message Queuing Protocol（amqp）高级消息队列协议。

* **Broker**: 接收和分发消息的应用，RabbitMQ Server就是Message Broker。
* **Virtual host**: 出于多租户和安全因素设计的，把AMQP的基本组件划分到一个虚拟的分组中，类似于网络中的namespace概念。当多个不同的用户使用同一个RabbitMQ server提供的服务时，可以划分出多个vhost，每个用户在自己的vhost创建exchange／queue等。
* **Connection**: publisher／consumer和broker之间的TCP连接。断开连接的操作只会在client端进行，Broker不会断开连接，除非出现网络故障或broker服务出现问题。
* **Channel**: 如果每一次访问RabbitMQ都建立一个Connection，在消息量大的时候建立TCP Connection的开销将是巨大的，效率也较低。Channel是在connection内部建立的逻辑连接，如果应用程序支持多线程，通常每个thread创建单独的channel进行通讯，AMQP method包含了channel id帮助客户端和message broker识别channel，所以channel之间是完全隔离的。Channel作为轻量级的Connection极大减少了操作系统建立TCP connection的开销。
* **Exchange**: message到达broker的第一站，根据分发规则，匹配查询表中的routing key，分发消息到queue中去。常用的类型有：direct (point-to-point), topic (publish-subscribe) and fanout (multicast)。
* **Queue**: 消息最终被送到这里等待consumer取走。一个message可以被同时拷贝到多个queue中。
* **Binding**: exchange和queue之间的虚拟连接，binding中可以包含routing key。Binding信息被保存到exchange中的查询表中，用于message的分发依据。

# confirm确认消息

confirm应答，当broker接收到消息时，会回复一个comfirm应答。

.NET客户端中可以通过下列方法来开启confim，并等待消息发送成功。

model.confirmSelect()

bool recievesuccess=model.WaitForConfirms();

如果消息发送失败，可以进行重试或者记录日志等。

Comfirm可以确保消息发送成功。

# Return消息机制

Return Listener 用于处理一些不可路由的消息

我们的生产者，通过指定一个Exchange和Routing key，把消息送达到某个消息队列，然后我们的消费者监听队列，进行消费操作。

但是在某些情况下，如果我们在发送消息的时候，当前的Exchange不存在或者指定的Routing key路由不到，此时我们需要监听这种不可达消息，就是用Return Listener。

.NET客户端中使用model.BasicReturn事件来监听return消息。

# qos（服务质量保证）

model.BasicQos(0, 2, false);

var consumer = new EventingBasicConsumer(model);

//consumer包含了失败重连了，不需要自己实现

consumer.Received += async (ch, ea) =>

{

model.BasicAck(ea.DeliveryTag, false);

model.BasicReject(ea.DeliveryTag, true);

};

//noAck设置为false，消费者需要ack确认才表示消息消费完

string consumerTag = model.BasicConsume(queuename, false, consumer);

通过Qos服务质量保证来给消费者限流，代码中表示消费者每次消费两个消息。

通过Ack确认保证消息消费完成之后，brocker才删除消息。当multiple为true时，该消费者所有的消费消息都会ack确认。

通过Reject可以拒绝消息，后面参数表示是否让消息重新入队。

<https://www.rabbitmq.com/confirms.html>

# 消息属性和有效负载

* content type
* content encoding
* routing key
* delivery mode(persistent or not)
* message priority
* message publish timestamp
* expiration period
* publisher application id

# TTL队列/消息

TTL 是 Time To Live 的缩写，也就是生存时间

RabbitMQ支持消息的过期时间设置，在消息发送时指定

RabbitMQ支持队列的过期时间设置，从消息入队时开始计算，只要超过队列的超时时间配置，消息自动清除

# 死信队列

死信队列也是一个普通的队列，不过该队列是专门用来存放死信消息的。

消息变成死信有以下几种情况

* 消息被拒绝（basic.reject/basic.nack）并且requeue = false。
* 消息TTL过期
* 队列达到最大长度

定义一个dlx.exchange、dlx.queue，并绑定在一块，dlx.queue作为死信队列存放死信消息。

再定义一个正常消息queue，设置queue的arguments的x-dead-letter-exchange来指定交互机死信交换机，当该队列的消息变成死信时，消息会再发送到x-dead-letter-exchange所指定的死信交换机上，然后发送到死信队列里。

arguments.put("x-dead-letter-exchange", "dlx.exchange")

# 题目