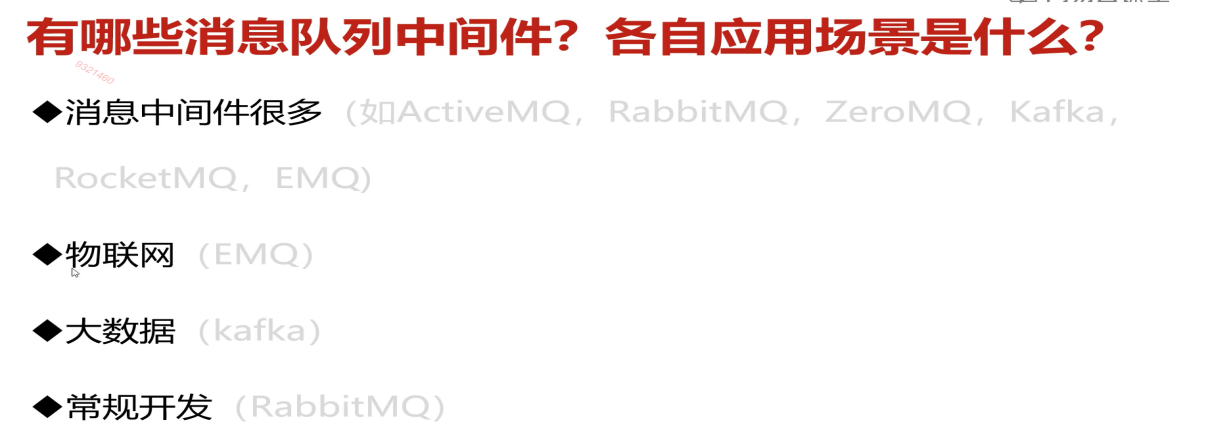
消息队列之RabbitMQ

**1常用消息队列**

Kafka在于分布式架构，RabbitMQ基于AMQP协议来实现，RocketMQ（阿里）/思路来源于kafka，改成了主从结构，在事务性可靠性方面做了优化。广泛来说，电商、金融等对事务性要求很高的，可以考虑RabbitMQ和RocketMQ，对性能要求高的可考虑Kafka。基于Apache的ActiveMQ性能最差。



**2安装RabbitMQ**

参考：<https://www.jianshu.com/p/ee9f7594212b>

#### **2.1 安装erlang**

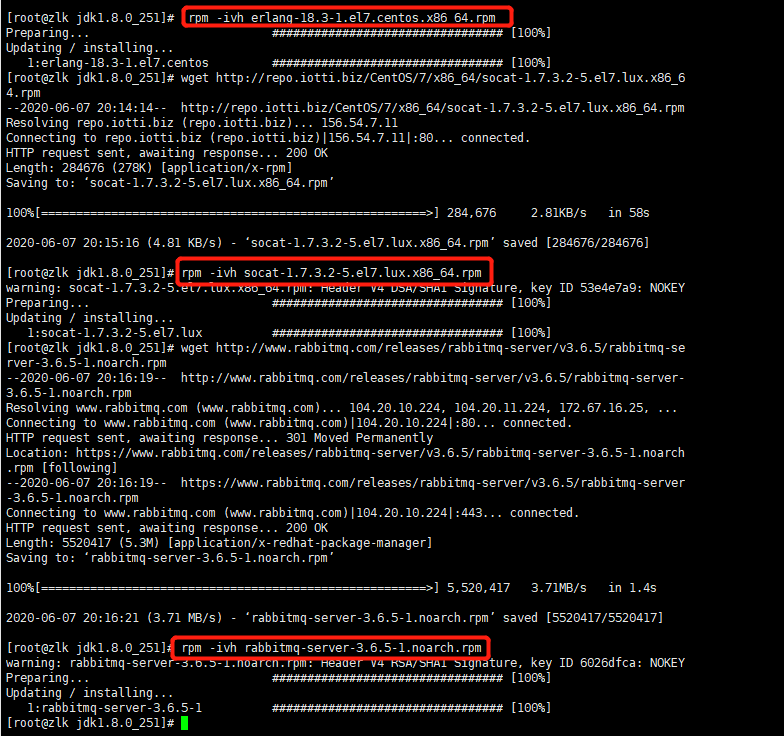
下载: wget http://www.rabbitmq.com/releases/erlang/erlang-18.3-1.el7.centos.x86\_64.rpm  
安装: rpm -ivh erlang-18.3-1.el7.centos.x86\_64.rpm

#### **2.2 安装socat**

下载: wget http://repo.iotti.biz/CentOS/7/x86\_64/socat-1.7.3.2-5.el7.lux.x86\_64.rpm  
安装: rpm -ivh socat-1.7.3.2-5.el7.lux.x86\_64.rpm

#### **2.3 安装rabbitmq**

下载: wget http://www.rabbitmq.com/releases/rabbitmq-server/v3.6.5/rabbitmq-server-3.6.5-1.noarch.rpm  
安装: rpm -ivh rabbitmq-server-3.6.5-1.noarch.rpm

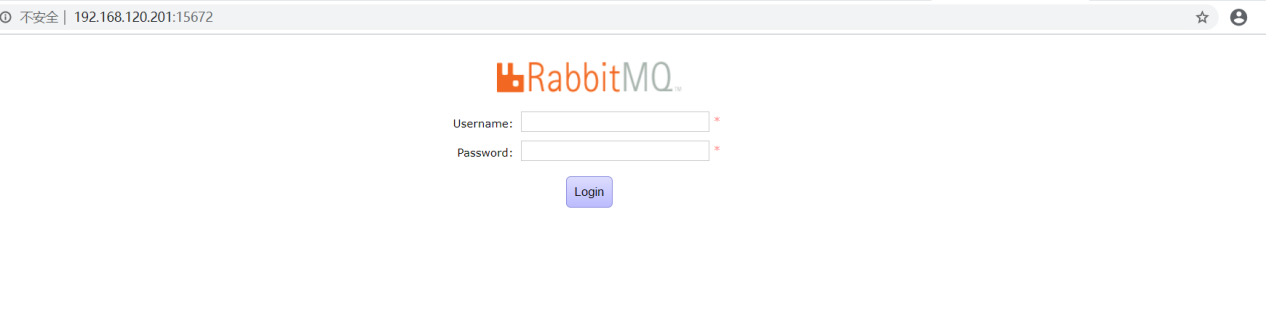
**启动服务**: service rabbitmq-server restart

rabbitmq默认端口5672。

#### **2.4 安装后台启动插件**

命令：rabbitmq-plugins enable rabbitmq\_management

安装成功后可以使用浏览器访问：ip:端口**（默认15672）**



**注意:**

如果服务器开启了防火墙, 则访问不了, 可以先关闭防火墙或者暴露端口  
centos7查看防火墙状态: firewall-cmd --state, 关闭后显示not running，开启后显示running  
关闭防火墙: systemctl stop firewalld.service

#### **2.5开启用户远程访问**

Rabbitmq从3.3.0开始, 默认用户guest只允许本机访问, 即: http://localhost:15672, 如果通过ip:port访问, 会发现Login failed, 为了让guest用户能够远程访问, 只需新建配置文件配置loopback\_users即可(rabbitmq.config配置文件需手动创建), 步骤如下:

这里显示的是没有找到配置文件，我们可以自己创建这个文件

cd /etc/rabbitmq/

vi rabbitmq.config

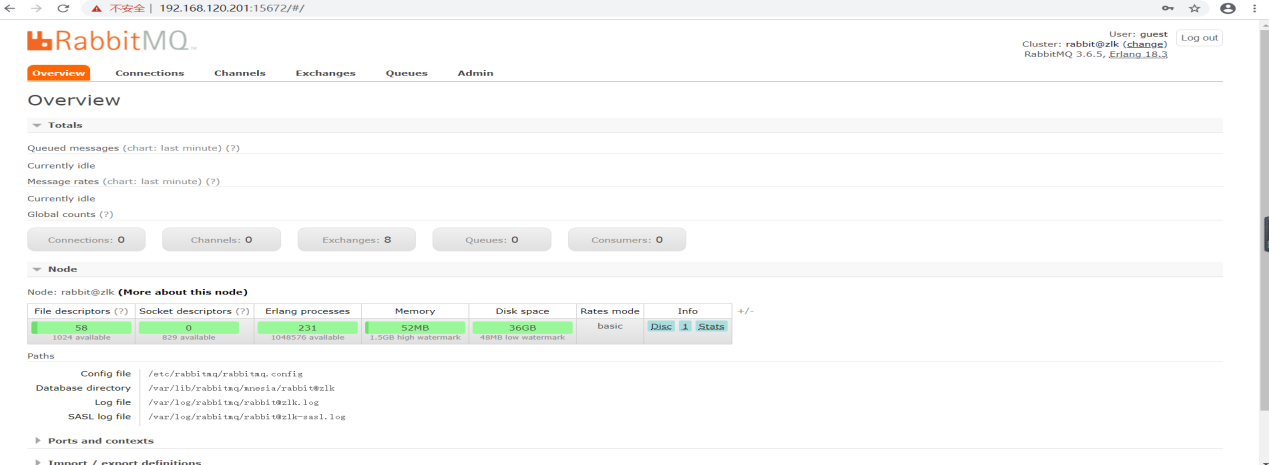
编辑内容如下：注意后面有个.

[{rabbit, [{loopback\_users, []}]}].

停止后，重启服务: service rabbitmq-server restart

这里的意思是开放使用，rabbitmq默认创建的**用户guest，密码也是guest**，这个用户默认只能是本机访问，localhost或者127.0.0.1，从外部访问需要添加上面的配置。

*目前为止, rabbitmq服务已安装完毕*



用户相关：<https://www.cnblogs.com/cwp-bg/p/10070467.html>

<https://blog.csdn.net/doubleqinyan/article/details/81081673>

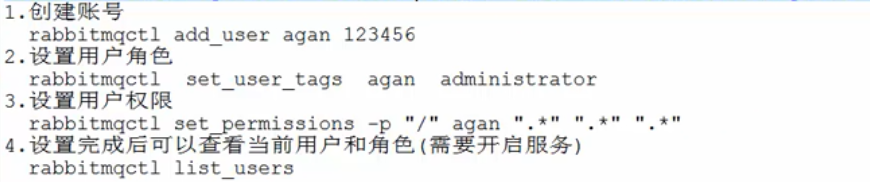
**3 RabbitMQ常用命令**

1. 查看mq服务状态：service rabbitmq-server status
2. 启动mq进程: service rabbitmq-server start
3. 停止进程：停止：service rabbitmq-server stop
4. 查看mq启动进程：ps -ef | grep rabbitmq
5. 开启管控台: rabbitmq-plugins enable rabbitmq\_management
6. 关闭管控台: rabbitmq-plugins disable rabbitmq\_management
7. 设置开机启动: chkconfig rabbitmq-server on
8. 用户相关命令

rabbitmqctl add\_user agan 123456

rabbitmqctl set\_user\_tags agan administrator

rabbitmqctl set\_permissions -p / agan '.\*' '.\*' '.\*'



**4 RabbitMQ 介绍**

基于amqp协议，使用插件后可以支持jms协议。开源，性能强。支持语言多，支持插件。

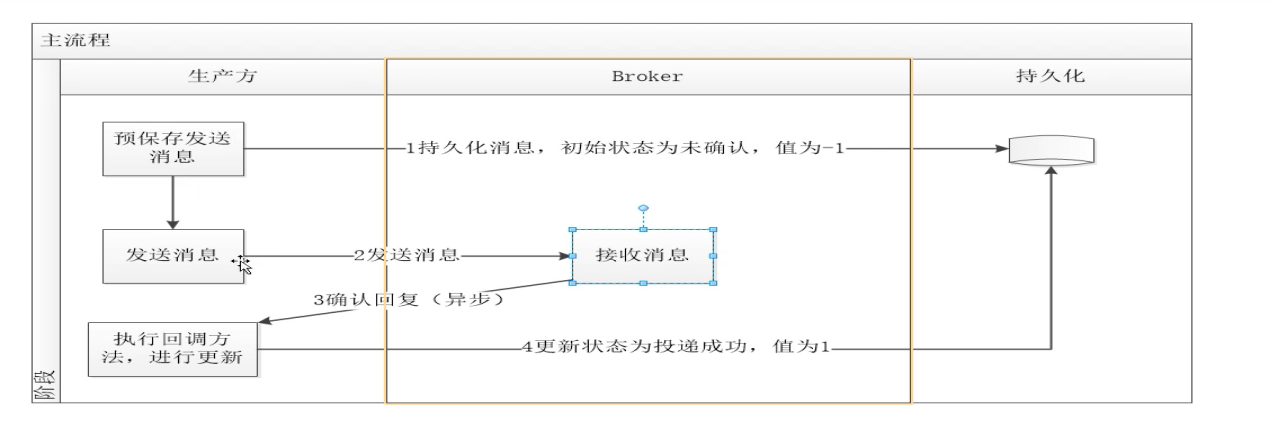
Jms:类似api定义。语言相关。

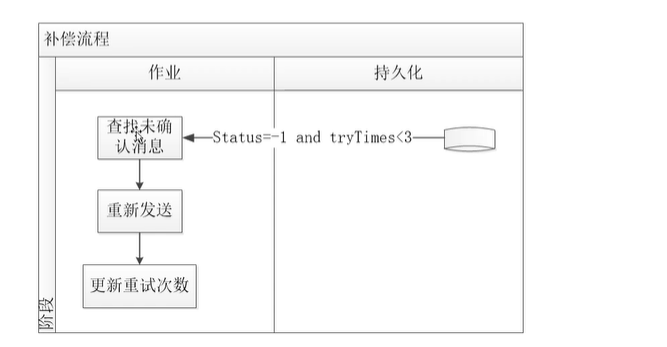
Amqp：从网络交换出发，类似http。和语言无关。

Mqtt:工作在TCP/IP上。面向物联网。

**作用**：异步、解耦与消峰。

发送的保证ack与持久化后面章节将介绍。





## 4.1工作过程

发布者（Publisher）发布消息（Message），经由交换机（Exchange）。

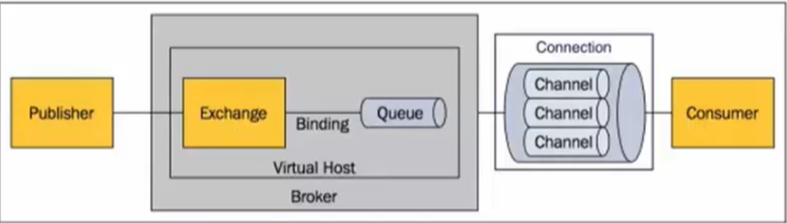
交换机根据路由规则将收到的消息分发给与该交换机绑定的队列（Queue）。

最后 AMQP 代理会将消息投递给订阅了此队列的消费者，或者消费者按照需求自行获取。

## 4.2主要有两种模式

(1)简单模式：一个生产者，一个消费者。队列对应消息全部由改消费者消费。应用场景:聊天，验证码，还有需要保证消费顺序时使用。

(2)work模式：一个生产者，多个消费者。每个消费者获取到的**消息唯一（资源竞争）**。应用场景:红包;大项目中的资源调度(任务分配系统不需知道哪一个任务执行系统在空闲,直接将任务扔到消息队列中,空闲的系统自动争抢)。



<https://blog.csdn.net/fxq8866/article/details/62049393?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.nonecase>

## 4.3 常用名词介绍

### 4.3.1 消息 message

消息由消息头与消息体构成，是不透明的，包含路由键（routing-key）、优先权（priority）、

是否需要持久化存储（delivery-mode）。

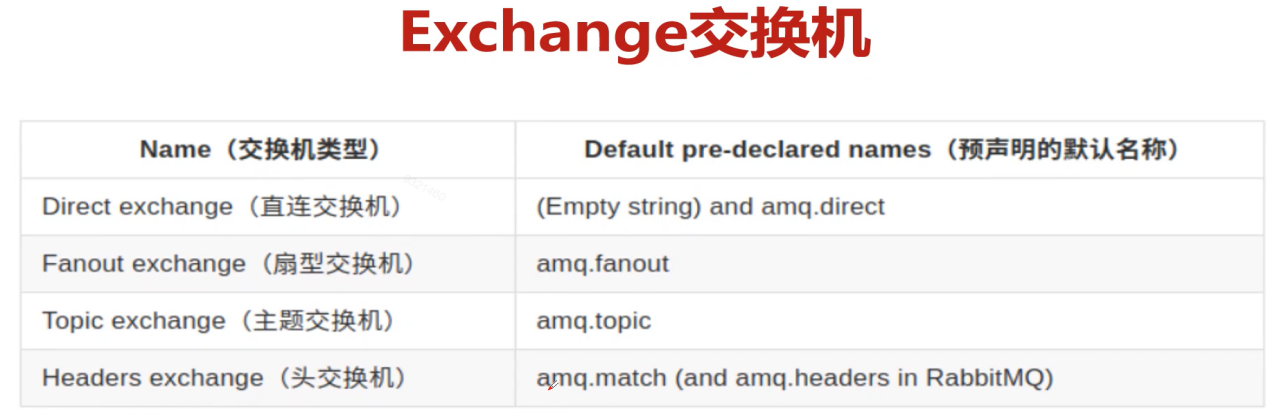
### 4.3.2 生产者 publisher

生产者是向交换器（exchange）发布消息的客户端应用。

### 4.3.3 消费者 consumer

消费者者是从队列（queue）获取信息的客户端应用。

### \*\*\*4.3.4交换器exchange（重点）



#### **(1)发布-订阅（direct）**

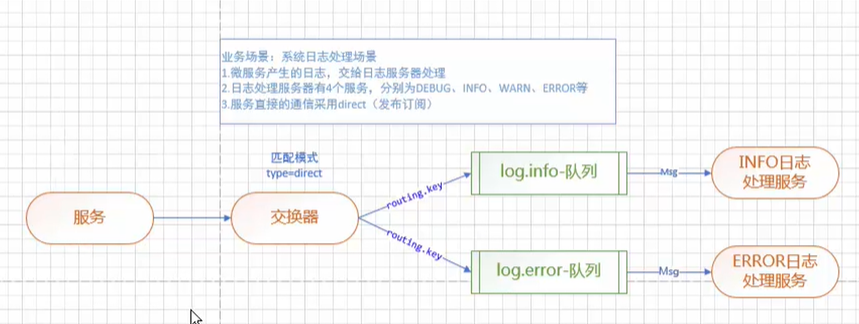
路由键完全匹配。消费者路由键（完全匹配）绑定队列与交换器。生产者路由键绑定交换器与消息。也是rabbitmq默认交换器（默认的direct交换器，也可以拿用队列名称当做路由键）。

  处理路由键。需要将一个队列绑定到交换器（通过路由键）上，要求该消息与一个特定的路由键完全匹配。这是一个完整的匹配。如果一个队列绑定到该交换器上要求路由键 “test”，则只有被标记为“test”的消息才被转发，不会转发test.aaa，也不会转发dog.123，只会转发test。

**应用场景**：用户通知，当用户充值成功或转账完成系统通知用户，通知方式有短信、邮件多种方法。队列少时适用，适用与消息在队列间相对于独立的情况。

**Work模式下**，多个消费者时。队列消费消息也会保证唯一消费（注：不含重复消息情况。指一个队列，多个消费者时，消息不会被队列重复消费）。但是与路由键(完全匹配)和交换器绑定的队列可以有多个（都会接受到对应的完整信息），就是说可以有多个队列（消费者：路由键和交换器绑定完全匹配的队列）来接收完整消息（生产者：路由键和交换器绑定的消息，路由键需要完全匹配）。

当接收端订阅者有多个的时候，direct会轮询公平的分发给每个订阅者（订阅者消息确认正常）



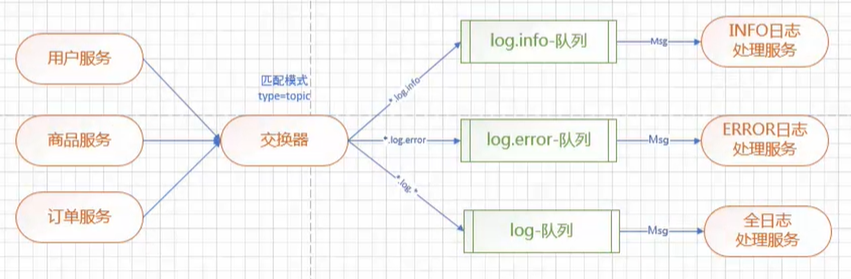
#### **(2)主题（topic）**

也是通过路由键（模糊匹配，\*一个，#一个或者多个）规则绑定交换器与队列。消费者路由键（规则匹配）绑定队列与交换器。生产者路由键绑定交换器与消息。

将路由键和某模式进行匹配。此时队列需要绑定要一个模式上。符号“#”匹配一个或多个词，符号“\*”匹配不多不少一个词。因此“audit.#”能够匹配到“audit.irs.corporate”类型，但是“audit.\*” 只会匹配到“audit.irs”类型。

**应用场景**：日志推送。队列有一点规则整理时适用，适用与消息在在某些队列独立，在某些队列会交叉消费的情况（因为规则匹配路由键）。

**Work模式下**，多个消费者时，队列消费消息也会保证唯一消费（注：不含重复消息情况。指一个队列，多个消费者时，消息不会被队列重复消费）。但是与路由键（规则匹配）和交换器绑定的队列可以有多个（都会接受到对应的完整信息），就是说可以有多个队列（消费者：路由键（模糊匹配，\*一个，#一个或者多个）和交换器绑定的队列）来接收完整消息（生产者：路由键和交换器绑定的消息，消费者队列模糊匹配上规则才行）。



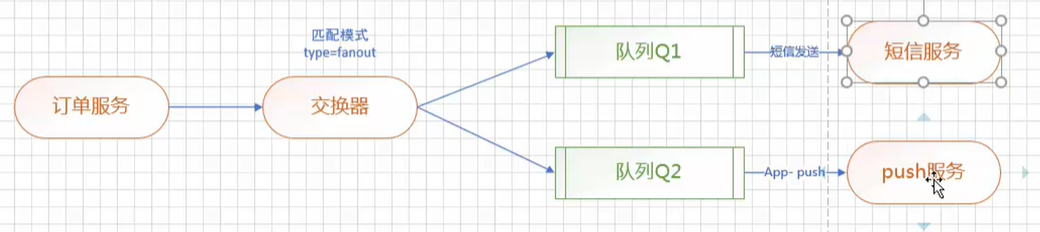
#### **(3)广播（fanout）**

消息将群发到当前交换器下绑定的所有队列上。该交换器队列与交换器不通过路由器绑定。 消费者队列直接绑定到交换器，生产者消息直接绑定到交换器。

 不处理路由键。你只需要简单的将队列绑定到交换器上。一个发送到交换器的消息都会被转发到与该交换器绑定的所有队列上。很像子网广播，每台子网内的主机都获得了一份复制的消息。Fanout交换器转**发消息是最快的**。

**应用场景**：消息推送，订单，红包等。适用于队列可以归为一个交换器类型的情况，且该交换器下所有队列需要消费消息是完全一样的；

**Work模式下**，多个消费者时，队列消费消息也会保证唯一消费（注：不含重复消息情况。指一个队列，多个消费者时，消息不会被队列重复消费）。但是交换器绑定的队列可以有多个（都会接受到对应的完整信息），就是说可以有多个队列（消费者：交换器绑定的队列）来接收完整消息（生产者：交换器绑定的消息）。



#### **(4)Headers**

设置header attribute参数类型的交换机,使用比较少。

### 4.3.5 绑定 binding

绑定是交换器和队列之间的关联，一个绑定就是基于路由键来绑定交换器与消息队列的规则。

### 4.3.6 队列 queue

队列就是消费者与生产者之间的一个中转站。生产者将信息发送到队列。消费者消费成功信息后，mq将当前对应消息剔除。

### 4.3.7 路由键 routing-key

决定消息去那个队列的规则。消息通过路由键与交换器绑定。消息发到交换器后，交换器会去找当前路由器下是不是有与该路由规则匹配的队列。匹配消息放入对应队列。不匹配则消息进入黑洞。

### 4.3.8 连接 connection

连接可以理解为rabbitmq服务器与应用服务器间进行的TCP连接。

### 4.3.9 信道 channel

是一条虚拟的连接，一个TCP中可以有多个信道（1减少tcp连接断开的消耗，2提升并发性能（TCP连接有限，一条TCP却信道可以有很多））。

TCP一旦打开就会创建 AMQP的信道。无论是发布消息、接收消息与调队列都是由信道完成。

### 4.3.10 虚拟主机 virtual host

表示一批交换器、队列相关对象。共享相同的身份认证与加密环境，本质上就是一个mq服务器。

### 4.3.11 消费服务器实体 broker

消费服务器实体。

**5 RabbitMQ主要问题**

## 5.1 消息丢失与确认（安全到达与安全消费）

**生产者丢失消息、消息列表丢失消息（持久化）、消费者丢失消息（ACK，持久化）；**

### 5.1.1 rabbitmq没拿到消费者的ack确认。（消费者挂掉、消费者抛异常、网络异常）

**出现问题**：rabbitmq会一直向消费者发送，导致内存溢出。

**解决方案**：先开启队列的持久化。autoDelete = "true"。保证消费不丢失。（为了保证在 RabbitMQ 退出或者 crash 了数据不丢失，需要将 Queue 和 Message 持久化。）

**解决方案1**：将消费者的接口中异常处理掉，try-catch

**两种方式：**

（1）catch异常后，手动发送到指定队列，然后使用channel给rabbitmq确认消息已消费

channel.basicNack(delivery.getEnvelope().getDeliveryTag(), false, true);

（2）给Queue绑定死信队列，使用nack（requque为false）确认消息消费失败

**解决方案2**：消费者配置开启重试与重试次数。到达重试次数后。Rabbitmq不再向消费者发送消息。

启用手动确认模式可以解决这个问题:消费者丢失

①自动确认模式，消费者挂掉，待ack的消息回归到队列中。消费者抛出异常，消息会不断的被重发，直到处理成功。不会丢失消息，即便服务挂掉，没有处理完成的消息会重回队列，但是异常会让消息不断重试。

②手动确认模式，如果消费者来不及处理就死掉时，没有响应ack时会重复发送一条信息给其他消费者；如果监听程序处理异常了，且未对异常进行捕获，会一直重复接收消息，然后一直抛异常；如果对异常进行了捕获，但是没有在finally里ack，也会一直重复发送消息(重试机制)。

③不确认模式，acknowledge="none" 不使用确认机制，只要消息发送完成会立即在队列移除，无论客户端异常还是断开，只要发送完就移除，不会重发。

### 5.2 确保生产者消息准确发送到rabbitmq。

<https://blog.csdn.net/qq_29914837/article/details/93376741>

对于消息发送方来说：

生产者发送一条消息，正常情况下是通过交换机发送到队列中，再由消费者接受队列中消息，由消费者进行逻辑操作。

但是RabbitMQ在收到消息后，还需要有一段时间才能将消息存入磁盘之中。

并且也并不是为每条消息都做fsync的处理，可能仅仅保存到cache中而不是物理磁盘上。如果RabbitMQ broker正巧发生了崩溃，因为消息还没来得及保存磁盘，消息将会丢失。

  （1）、发送的消息怎么样才算失败或成功？

当消息无法路由到任何队列时，确认消息路由失败。

消息成功路由时，当需要发送的队列都发送成功后，进行确认消息，对于持久化队列意味着写入磁盘，对于镜像队列意味着所有镜像接收成

  （2）、如何确定发送的消息是否成功？

## RabbitMQ引入发送端消息确认机制，主要通过**AMQP事务和publisher Confirm机制**。

**AMQP事务使用**

RabbitMQ支持事务(transaction)，RabbitMQ中与事务机制有关的方法有三个：txSelect(), txCommit()以及txRollback()。

（1）txSelect用于将当前channel设置成transaction模式，通过调用tx.select方法开启事务模式。

（2）txCommit用于提交事务。当开启了事务模式后，只有当一个消息被所有的镜像队列保存完毕后，RabbitMQ才会调用tx.commit-ok返回给客户端。

（3）txRollback用于回滚事务，在通过txSelect开启事务之后，我们便可以发布消息给broker代理服务器了，如果txCommit提交成功了，则消息一定到达了broker了，如果在txCommit执行之前broker异常崩溃或者由于其他原因抛出异常，这个时候我们便可以捕获异常通过txRollback回滚事务了。

关键代码：

channel.txSelect();

//ConfirmConfig.exchangeName(交换机名称)

//ConfirmConfig.routingKey(路由键)

//message （消息内容）

channel.basicPublish(ConfirmConfig.exchangeName,

ConfirmConfig.routingKey, MessageProperties.PERSISTENT\_TEXT\_PLAIN, message));

channel.txCommit();

事务确实能够解决producer与broker之间消息确认的问题，只有消息成功被broker接受，事务提交才能成功，否则我们便可以在捕获异常进行事务回滚操作同时进行消息重发。

**事务机制的缺点** ：

使用事务机制的话会降低RabbitMQ的性能。

会导致生产者和RabbitMq之间产生同步(等待确认)，这也违背了我们使用RabbitMq的初衷。所以一般很少采用.

发送确认分为两步，一是确认是否到达交换器，二是确认是否到达队列。

**confirm模式的实现原理**

生产者将信道设置成confirm模式，一旦信道进入confirm模式，所有在该信道上面发布的消息都会被指派一个唯一的ID(从1开始)，一旦消息被投递到所有匹配的队列之后，broker就会发送一个确认给生产者（包含消息的唯一ID）,这就使得生产者知道消息已经正确到达目的队列了，如果消息和队列是可持久化的，那么确认消息会将消息写入磁盘之后发出，broker回传给生产者的确认消息中deliver-tag域包含了确认消息的序列号，此外broker也可以设置basic.ack的multiple域，表示到这个序列号之前的所有消息都已经得到了处理。

**ConfirmCallback和ReturnCallback**

对于ConfirmCallback来说：

如果消息没有到exchange,则confirm回调,ack=false

如果消息到达exchange,则confirm回调,ack=true

对于ReturnCallback来说：

exchange到queue成功,则不回调return

exchange到queue失败,则回调return(需设置mandatory=true,否则不回回调,消息就丢了)

比如路由不到队列时触发回调

### 5.3 rabbitmq挂掉

## 5.2 消息持久化

1）**消息未消费完，消费者进程中断**导致消息丢失一部分。设置autoDelete = "false".

未消费队列会缓存到队列中（数据会存放在缓存中）。消费者启动后会继续消费。

@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.error}",autoDelete = "true"),  
 exchange = @Exchange(value = "${mq.config.exchange}",type = ExchangeTypes.*DIRECT*),  
 key = "${mq.config.queue.error.routing.key}"  
 )  
)

****交换器的持久化、队列的持久化和消息的持久化****

**交换器的持久化**

交换器的持久化是通过声明队列时，将durable参数设置为true实现的。如果交换器不设置持久化，那么rabbitmq服务重启之后，相关的交换器元数据将会丢失，不过消息不会丢失，只是不能将消息发送到这个交换器中了，建议将交换器设置为持久化

**队列的持久化**

队列的持久化是通过声明队列时，将durable参数设置为true实现的。如果队列不设置持久化，那么rabbitmq服务重启之后，相关的队列元数据将会丢失，而消息是存储在队列中的，所以队列中的消息也会被丢失

### **消息的持久化**

### 队列的持久化只能保证其队列本身的元数据不会被丢失，但是不能保证消息不会被丢失。所以消息本身也需要被持久化，可以在投递消息前设置AMQP.BasicProperties的属性deliveryMode为2即可：

注：

发送方：持久化交换器。持久化消息。

接收方：持久化交换器，持久化队列，

### **问题**

描述：将交换器、队列和消息都设置持久化之后就能保证数据不会被丢失吗？当然不是，多个方面：

* 消费者端: 消费者订阅队列将autoAck设置为true,虽然消费者接收到了消息，但是没有来得及处理就宕机了，那数据也会丢失，解决方案就是以为手动确认接收消息，待处理完消息之后，手动删除消息
* 在rabbitmq服务端，如果消息正确被发送，但是rabbitmq未来得及持久化，没有将数据写入磁盘，服务异常而导致数据丢失，解决方案，可以通过rabbitmq集群的方式实现消息中间件的高可用性

## 5.3 重复消费问题

## 5.4 怎么保证消息的顺序

生产者按顺序放入一个mQ队列让一个消费者消费。

如果存在多个消费者，那么就让每个消费者对应一个queue，然后把要发送 的数据全都放到一个queue，这样就能保证所有的数据只到达一个消费者从而保证每个数据到达数据库都是顺序的。  
 rabbitmq：拆分多个queue，每个queue一个consumer，就是多一些queue而已，确实是麻烦点；或者就一个queue但是对应一个consumer，然后这个consumer内部用内存队列做排队，然后分发给底层不同的worker来处理

## 5.5 mq挂掉导致问题(里面还有未消费消息)

<https://blog.csdn.net/u013887008/article/details/90341014?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>

持久化

## 5.6 mq削峰

<https://www.cnblogs.com/jack1995/p/10908818.html>

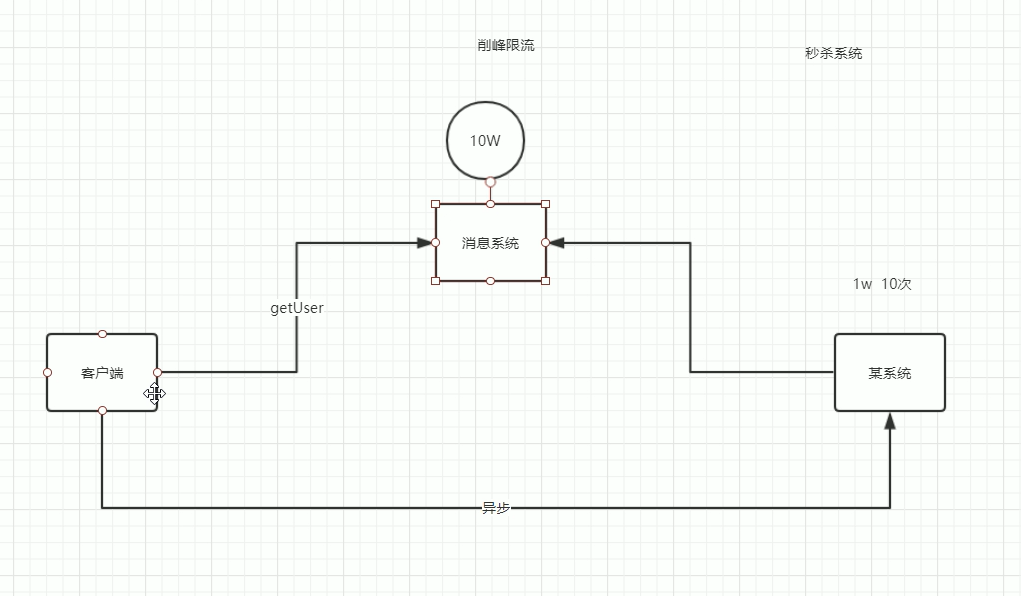
<https://blog.csdn.net/caiqing116/article/details/84640688>

**如秒杀**

对每个listener在初始化的时候设置的并发消费者的个数为5，每个 consumer 可以预监听消费拉取的消息数量为 5 个，如果同一时间处理不完，会将其缓存在 mq 的客户端等待处理！（rmq.channelCacheSize=25）

可以控制队列大小（但是会），超过库存的直接丢弃或者放到死信队列（垃圾队列中）。

首先写个临时程序连接mq消息，消费数据，来一个丢弃一个，快速消费积压的消息，降低mq消息的压力，在晚上的时候，采取批量重新导入的方法，把丢失的数据重新找回来

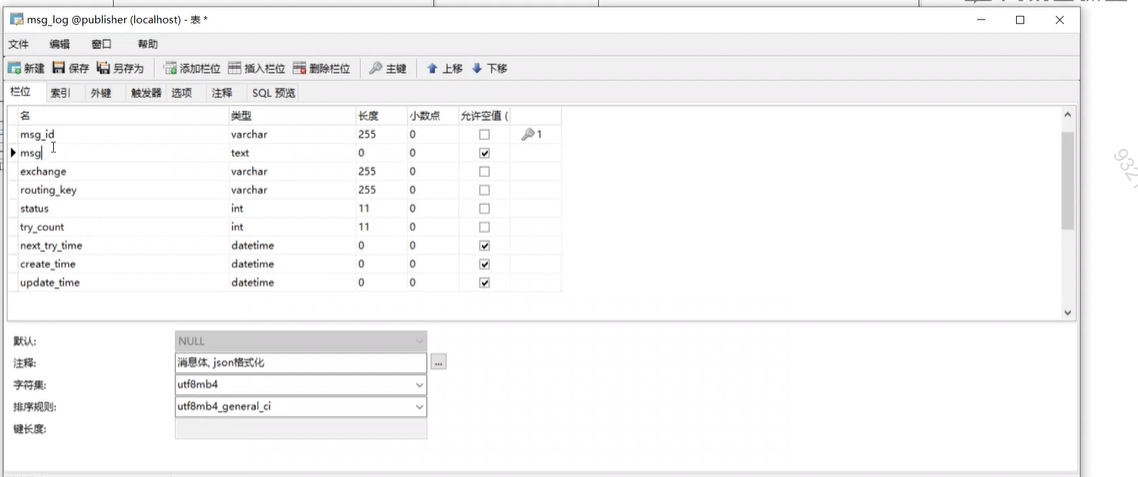


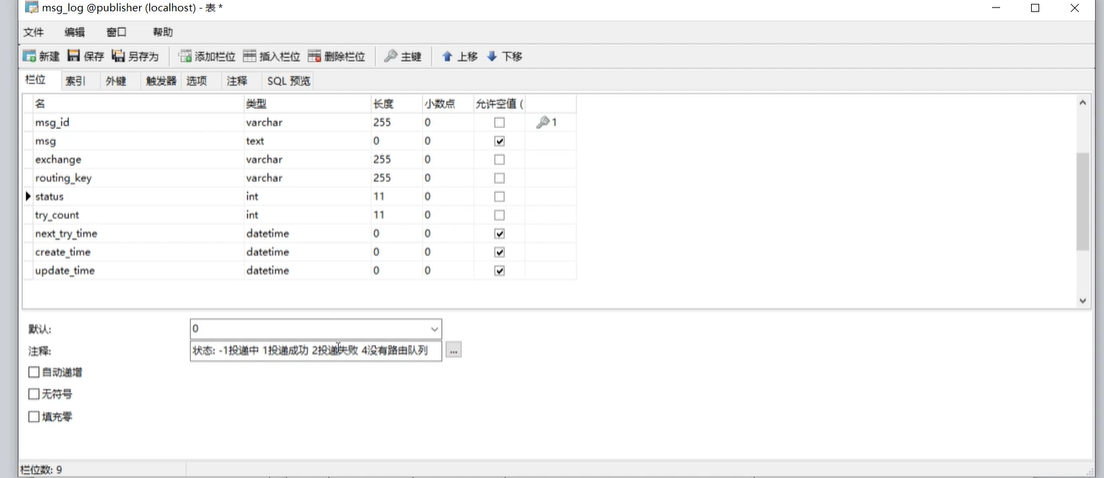
## 5.7 为什么使用信道不直接使用TCP?

一个tcp可以有很多信道，减少tcp连接消耗。提高性能（tcp连接在有限）。

## 5.8 消息可靠投递（生产者）

使用confirm与数据库记录并重试。使用定时任务去跑补偿（需要控制时间与重试次数）。





**5.8 消息可靠消费**

Ack手动确认机制与有限重试机制。死信队列。Redis去重。

//死信队列https://www.jianshu.com/p/9d496fde24b2

// https://www.cnblogs.com/piaolingzxh/p/5448927.html

1. 不捕获异常或者捕获异常（ack自动返回成功），只是设置重试。重试结束消息会从队列删除。（不论是否成功队列中消息都删除，存在问题）

package com.zlk.mq.direct;  
  
import org.springframework.amqp.core.ExchangeTypes;  
import org.springframework.amqp.rabbit.annotation.\*;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息(error日志处理)  
 \*  
 \** ***@RabbitListener*** *绑定队列  
 \* <p>  
 \* autoDelete = "true" 临时队列，autoDelete = "false" 则开启持久化  
 \* type = ExchangeTypes.DIRECT 交换器类型 消息-订阅  
 \* key 路由键  
 \*/*@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.error}", autoDelete = "false"),  
 exchange = @Exchange(value = "${mq.config.exchange}", type = ExchangeTypes.*DIRECT*),  
 key = "${mq.config.queue.error.routing.key}"  
 )  
)  
public class ErrorReceiver {  
  
 */\*\*  
 \* 从消息队列接收error日志  
 \*/* @RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("从消息队列接收error日志：" + msg);  
 try{  
 Integer s = 10/0;  
 }catch (Exception ex){  
  
 }  
 }

@RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("从消息队列接收error日志：" + msg);  
 Integer s = 10/0;  
 }  
  
}