第一节:rabbitmq安装以及入门

初识RabbitMq:

rabbitmq**是一个开源的消息代理和队列服务器，通过普通的协议**(Amqp**协议**)**来完成不同应用之间的数据共享**

(消费生产和消费者 可以跨语言平台)

rabbitmq**是通过**elang**语言来开发的基于**amqp**协议**

一：各大互联网公司为什么选择Rabbitmq

1)**比如滴滴，美团，携程，去哪儿**

2)**开源，性能好，稳定性保证,**

3)**提供了消息的可靠性投递（**confirm**），返回模式**

4)**与**sping amqp **整合和完美，提供丰富的**api

5)**集群模式十分丰富**(HA**模式　镜像队列模型**)

6)保证数据不丢失的情况下，保证很好的性能

二：Rabbitmq高性能是如何做到的

1）**使用的语言是**elang**语言**(**通常使用到交互机上**)，elang**的语言的性能能与原生**socket**的延迟效果**.

2)消息入队的延时已经消息的消费的响应很快

三:什么是AMQP协议(Advanced message queue protocol) 高级消息队列协议

1:)**是一个二进制协议,**

2）amqp **是一个应用层协议的规范（定义了很多规范）,可以有很多不同的消息中间件产品（需要遵循该规范）**

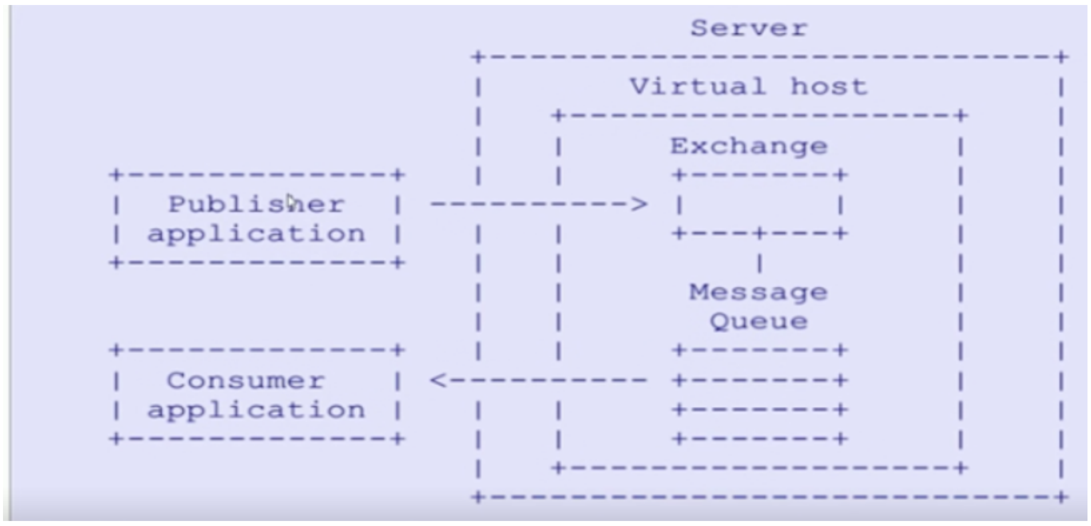
server：**是消息队列节点**

virtual host:**虚拟注解**

exchange **交换机**(**消息投递到交换机上**)

message queue **(被消费者监听消费)**

交互机和队列是有一个绑定的关系



四:AMQP**的核心概念**

1:server :**又称为**broker，**接受客户端连接，实现**amqp**实体服务**

2:Connection: **连接,应用程序与**brokder**建立网络连接**

3：channel：**网络通道，几乎所有的操作都是在**channel**中进行的，是进行消息对象的通道，客户端可以建立多个通道，每一个**channel**表示一个会话任务**

4:Message: **服务器和应用程序之间传递数据的载体，有**properties（**消息属性,用来修饰消息,比如消息的优先级,延时投递**）**和**Body（**消息体**）

5:virtual host(**虚拟主机**): **是一个逻辑概念,最上层的消息路由，一个虚拟主机中可以包含多个**exhange **和** queue **但是一个虚拟主机中不能有名称相同的**exchange **和**queue

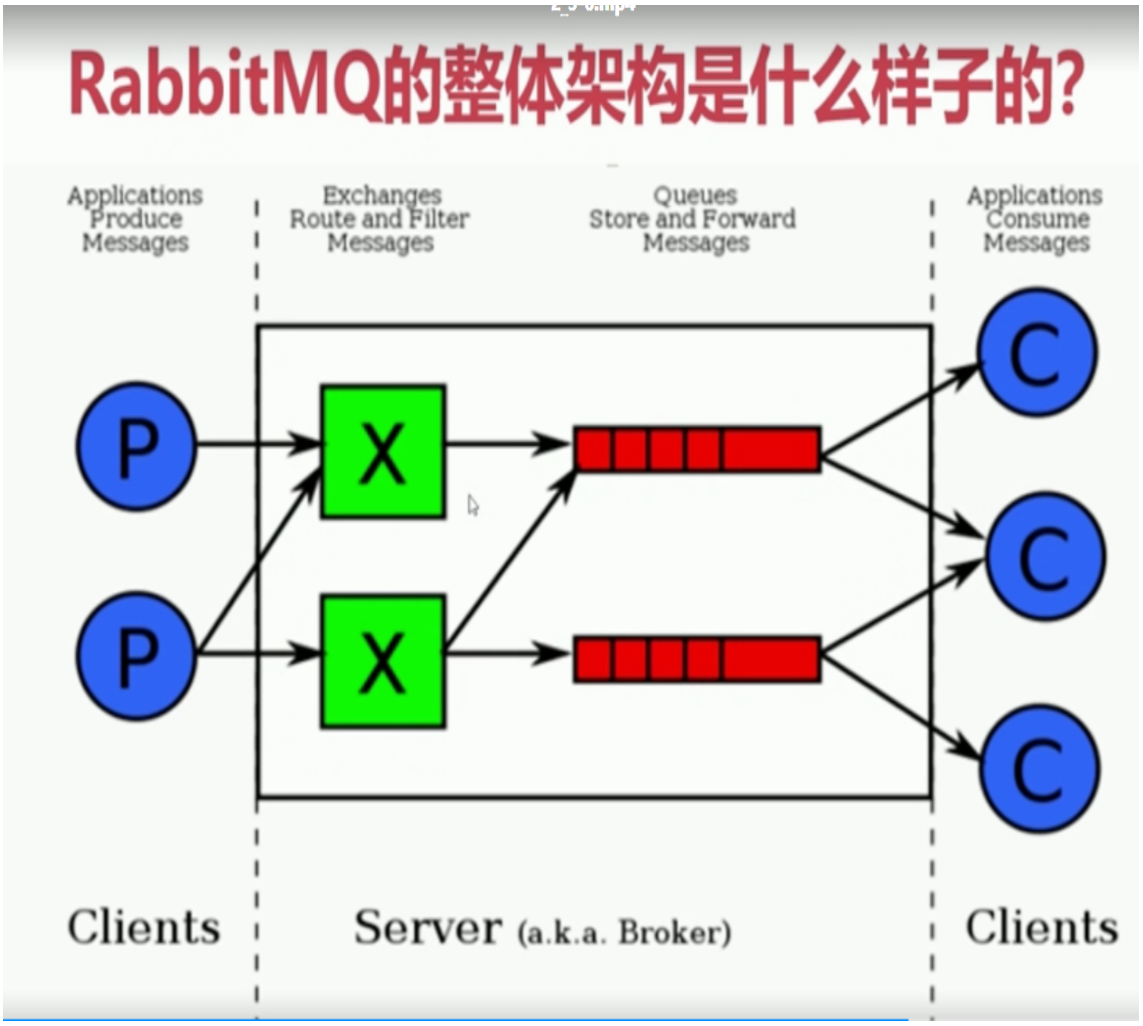
6:exchange **交换机**: **消息直接投递到交换机上，然后交换机根据消息的路由**key **来路由到对应绑定的队列上**

7:baingding: **绑定** exchange **与**queue**的虚拟连接**,bingding**中可以包含**route_key

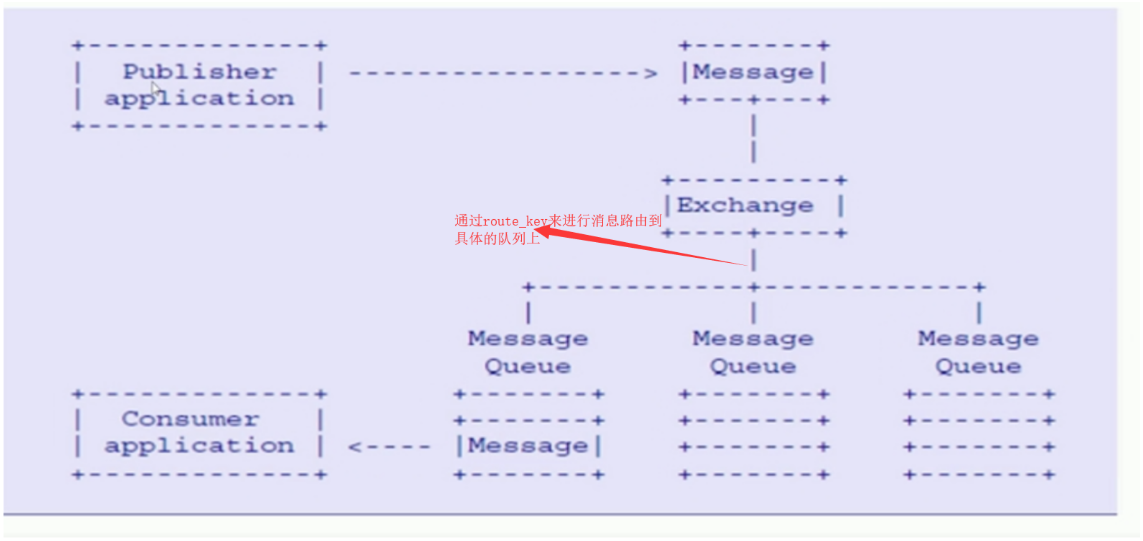
8：route_key **路由**key，**他的作用是在交换机上通过**route_key**来把消息路由到哪个队列上**

9:queue：**队列，用于来保存消息的载体，有消费者监听，然后消费消息**

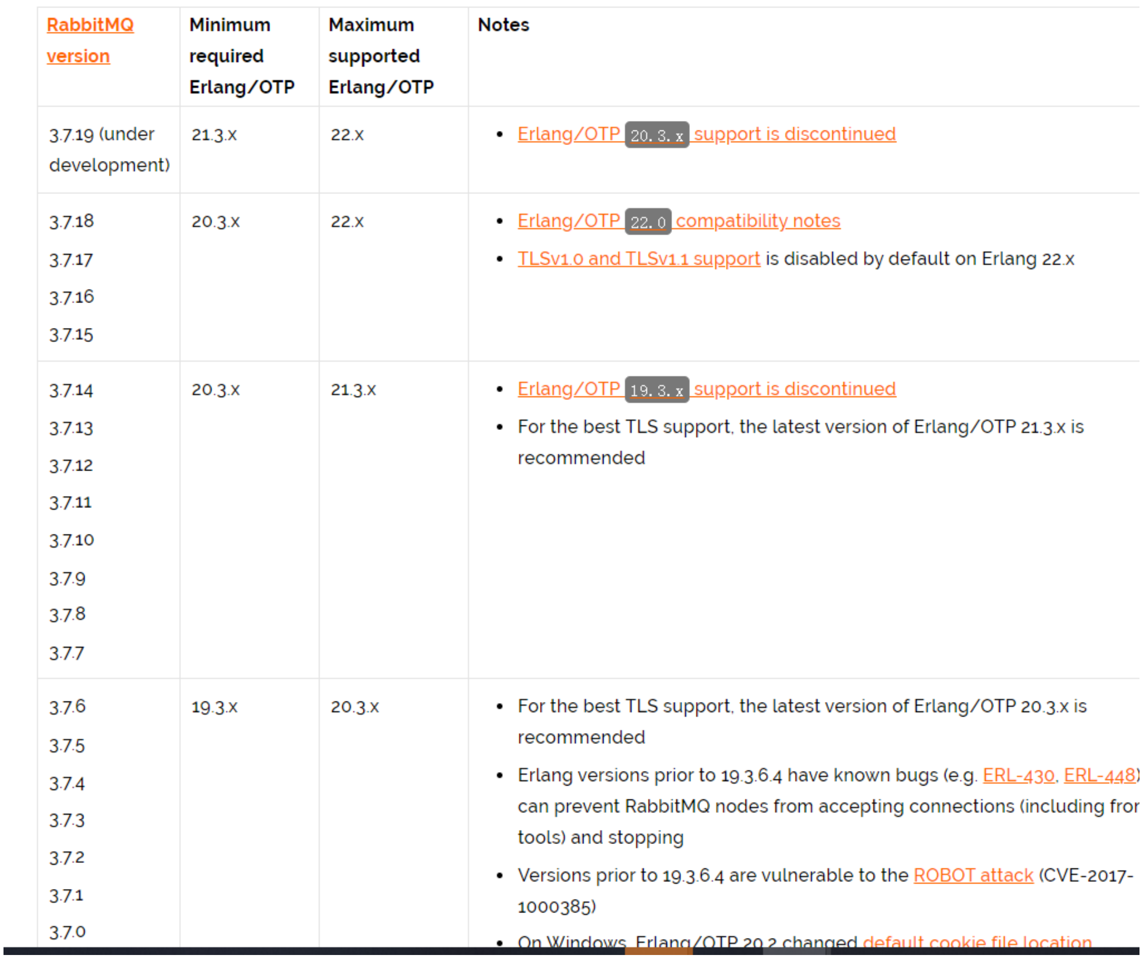
五:Rabbitmq**的整体架构模型**


RabbitMQ的整体架构是什么样子的?
Applications Produce Messages
Exchanges Route and Filter Messages
Queues Store and Forward Messages
Applications Consume Messages
P
P
X
X
C
C
C
Clients
Server (a.k.a. Broker)
Clients

六:rabbitmq**的消息是如何流转的**



七:rabbitmq**的安装和使用**



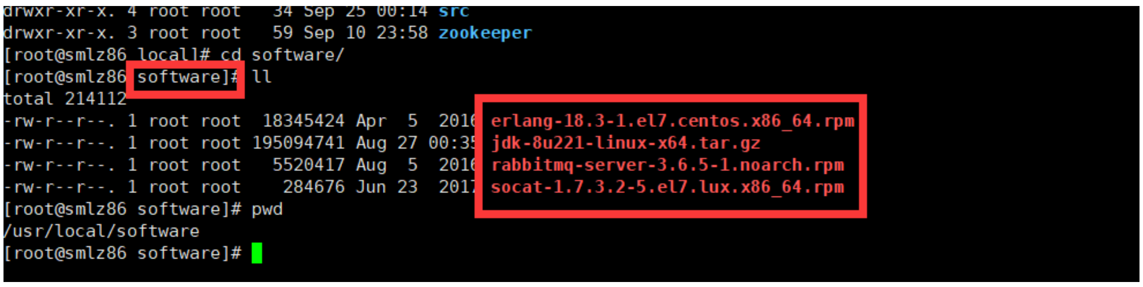
RabbitMQ version	Minimum required Erlang/OTP	Maximum supported Erlang/OTP	Notes
3.7.19 (under development)	21.3.x	22.x	• Erlang/OTP `20.3.x` support is discontinued
3.7.18 3.7.17 3.7.16 3.7.15	20.3.x	22.x	• Erlang/OTP `22.0` compatibility notes • TLSv1.0 and TLSv1.1 support is disabled by default on Erlang 22.x
3.7.14 3.7.13 3.7.12 3.7.11 3.7.10 3.7.9 3.7.8 3.7.7	20.3.x	21.3.x	• Erlang/OTP `19.3.x` support is discontinued • For the best TLS support, the latest version of Erlang/OTP 21.3.x is recommended
3.7.6 3.7.5 3.7.4 3.7.3 3.7.2 3.7.1 3.7.0	19.3.x	20.3.x	• For the best TLS support, the latest version of Erlang/OTP 20.3.x is recommended • Erlang versions prior to 19.3.6.4 have known bugs (e.g. ERL-430, ERL-448) can prevent RabbitMQ nodes from accepting connections (including fror tools) and stopping • Versions prior to 19.3.6.4 are vulnerable to the ROBOT attack (CVE-2017-1000385) • On Windows, Erlang/OTP 20.2 changed default cookie file location

①:安装rabbitmq**所需要的依赖包**

yum install build-essential openssl openssl-devel unixODBC unixODBC-devel make gcc gcc-c++ kernel-devel m4 ncurses-devel tk tc xz

②:下载安装包 (PS:**老师的下载包的目录是** cd /usr/local/software)

```
wget www.rabbitmq.com/releases/erlang/erlang-18.3-1.el7.centos.x86_64.rpm
wget http://repo.iotti.biz/CentOS/7/x86_64/socat-1.7.3.2-5.el7.lux.x86_64.rpm
wget www.rabbitmq.com/releases/rabbitmq-server/v3.6.5/rabbitmq-server-3.6.5-1.noarch.rpm
```



```
drwxr-xr-x. 4 root root    34 Sep 25 00:14 src
drwxr-xr-x. 3 root root    59 Sep 10 23:58 zookeeper
[root@smlz86 local]# cd software/
[root@smlz86 software]# ll
total 214112
-rw-r--r--. 1 root root  18345424 Apr  5  2016 erlang-18.3-1.el7.centos.x86_64.rpm
-rw-r--r--. 1 root root 195094741 Aug 27 00:35 jdk-8u221-linux-x64.tar.gz
-rw-r--r--. 1 root root   5520417 Aug  5  2016 rabbitmq-server-3.6.5-1.noarch.rpm
-rw-r--r--. 1 root root    284676 Jun 23  2017 socat-1.7.3.2-5.el7.lux.x86_64.rpm
[root@smlz86 software]# pwd
/usr/local/software
[root@smlz86 software]#
```

③:安装服务命令

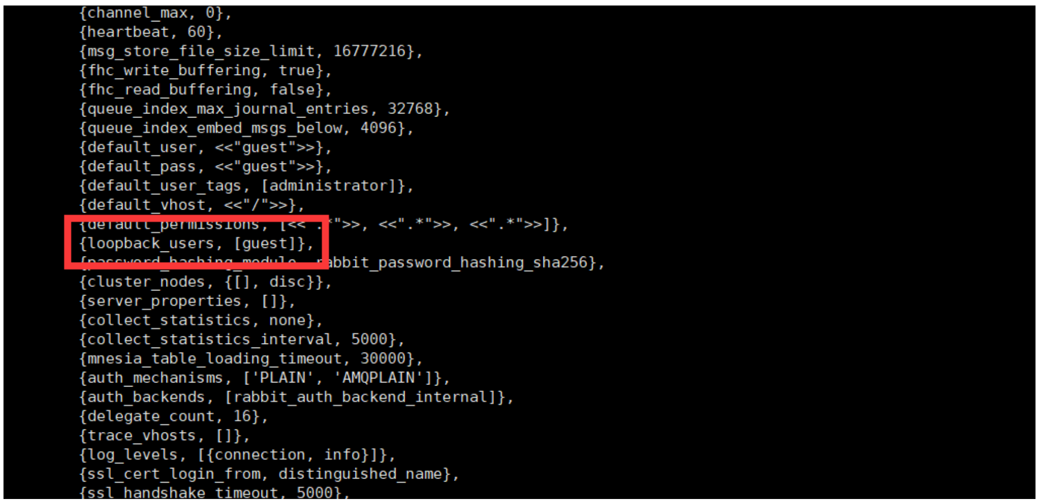
```
#第一步:安装erlang语言环境
rpm -ivh erlang-18.3-1.el7.centos.x86_64.rpm
#第二步:安装socat加解密软件
rpm -ivh socat-1.7.3.2-5.el7.lux.x86_64.rpm
#第三步:最后安装rabbitmq
```

```
rpm -ivh rabbitmq-server-3.6.5-1.noarch.rpm
```

④:修改集群用户与连接心跳检测

注意修改vim /usr/lib/rabbitmq/lib/rabbitmq_server-3.6.5/ebin/rabbit.app**文件**

修改：loopback_users 中的 <<"guest">>,只保留guest（不修改只能通过localhost访问）

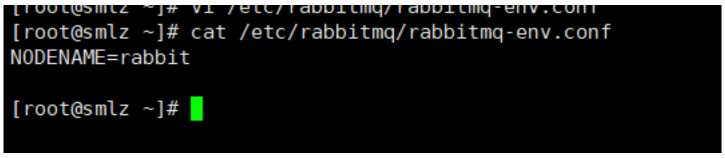


```
        {channel_max, 0},
        {heartbeat, 60},
        {msg_store_file_size_limit, 16777216},
        {fhc_write_buffering, true},
        {fhc_read_buffering, false},
        {queue_index_max_journal_entries, 32768},
        {queue_index_embed_msgs_below, 4096},
        {default_user, <<"guest">>},
        {default_pass, <<"guest">>},
        {default_user_tags, [administrator]},
        {default_vhost, <<"/">>},
        {default_permissions, [<<".*">>, <<".*">>, <<".*">>]},
        {loopback_users, [guest]},
        {password_hashing_module, rabbit_password_hashing_sha256},
        {cluster_nodes, {[], disc}},
        {server_properties, []},
        {collect_statistics, none},
        {collect_statistics_interval, 5000},
        {mnesia_table_loading_timeout, 30000},
        {auth_mechanisms, ['PLAIN', 'AMQPLAIN']},
        {auth_backends, [rabbit_auth_backend_internal]},
        {delegate_count, 16},
        {trace_vhosts, []},
        {log_levels, [{connection, info}]},
        {ssl_cert_login_from, distinguished_name},
        {ssl_handshake_timeout, 5000},
```

⑤:修改 本机系统文件

a:**修改** vim /etc/rabbitmq/rabbitmq-env.conf

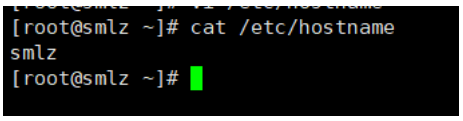
添加: NODENAME=rabbit



```
[root@smlz ~]# vi /etc/rabbitmq/rabbitmq-env.conf
[root@smlz ~]# cat /etc/rabbitmq/rabbitmq-env.conf
NODENAME=rabbit

[root@smlz ~]# 
```

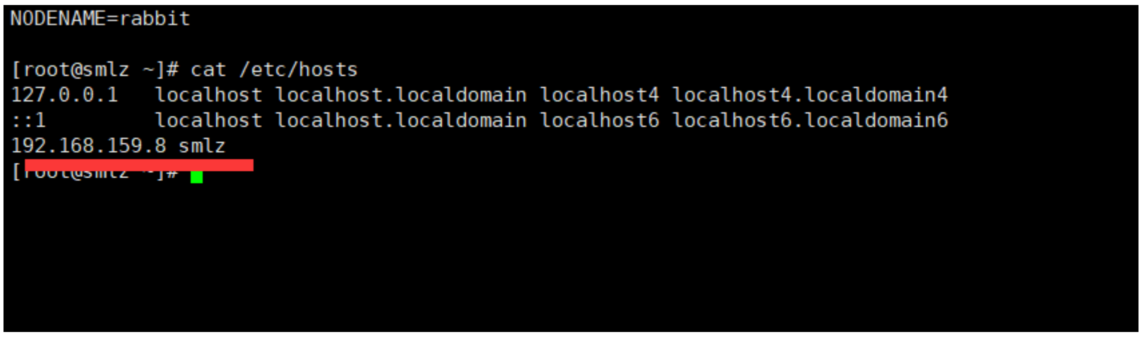
b:**修改** vim /etc/hostname

老师的是smlz **你自己的根据情况来**



```
[root@smlz ~]# vi /etc/hostname
[root@smlz ~]# cat /etc/hostname
smlz
[root@smlz ~]# 
```

c:**修改本地** vim /etc/hosts**文件**



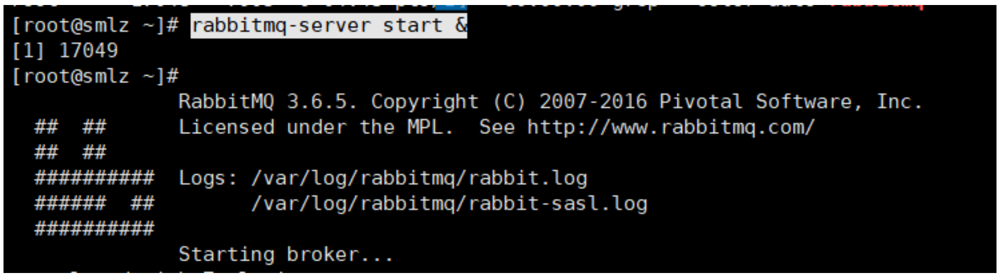
```
NODENAME=rabbit

[root@smlz ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.159.8 smlz
[root@smlz ~]# 
```

⑥:验证服务器是可用的

rabbitmq-server start &

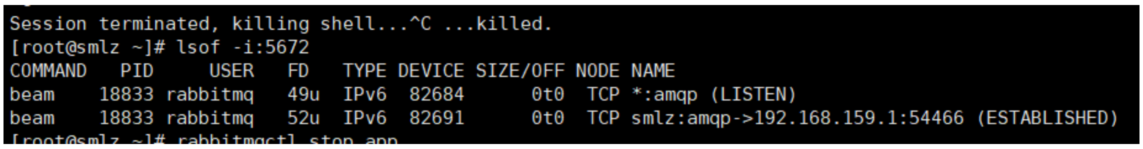
执行管控台插件:(不然不能在浏览器方法)

rabbitmq-plugins enable rabbitmq_management



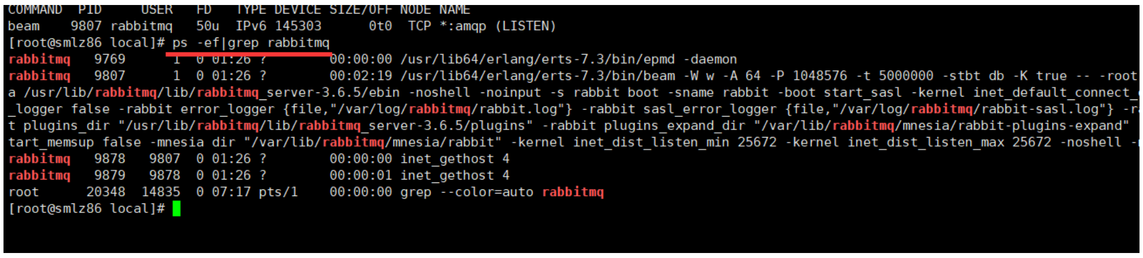
```
[root@smlz ~]# rabbitmq-server start &
[1] 17049
[root@smlz ~]#
              RabbitMQ 3.6.5. Copyright (C) 2007-2016 Pivotal Software, Inc.
  ##  ##      Licensed under the MPL.  See http://www.rabbitmq.com/
  ##  ##
  ##########  Logs: /var/log/rabbitmq/rabbit.log
  ######  ##        /var/log/rabbitmq/rabbit-sasl.log
  ##########
              Starting broker...
```

检查端口: lsof -i:5672



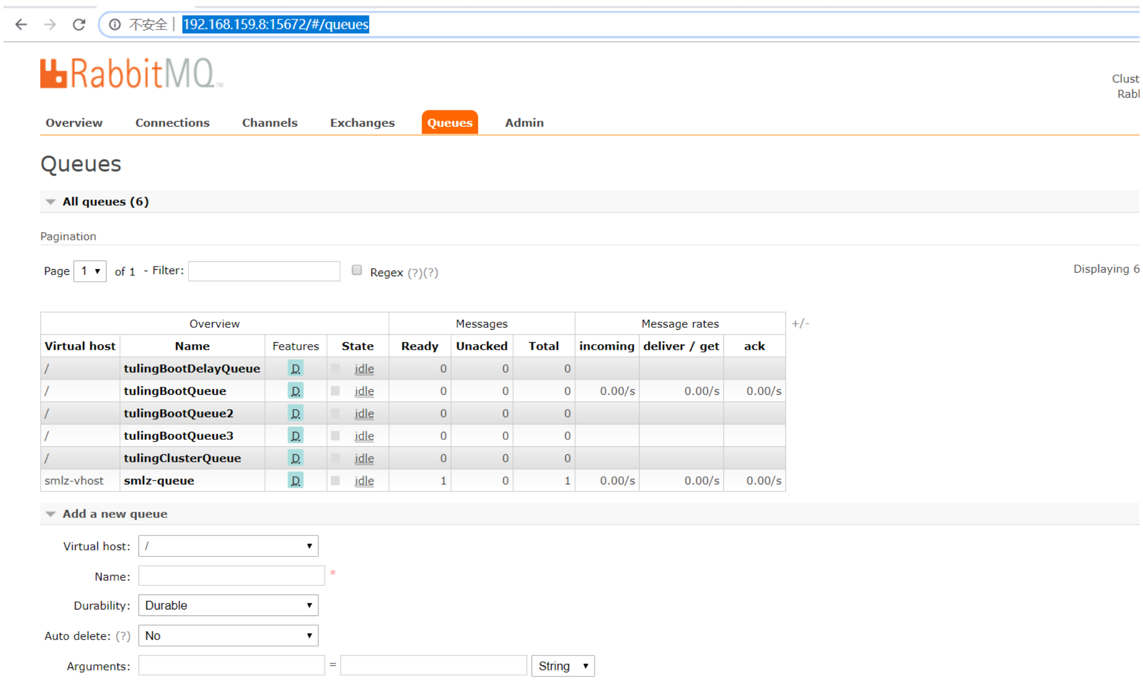
```
Session terminated, killing shell...^C ...killed.
[root@smlz ~]# lsof -i:5672
COMMAND   PID     USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
beam    18833 rabbitmq   49u  IPv6  82684      0t0  TCP *:amqp (LISTEN)
beam    18833 rabbitmq   52u  IPv6  82691      0t0  TCP smlz:amqp->192.168.159.1:54466 (ESTABLISHED)
```

通过 ps -ef|grep rabbitmq



```
COMMAND  PID     USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
beam    9807 rabbitmq   50u  IPv6 145303      0t0  TCP *:amqp (LISTEN)
[root@smlz86 local]# ps -ef|grep rabbitmq
rabbitmq   9769      1  0 01:26 ?        00:00:00 /usr/lib64/erlang/erts-7.3/bin/epmd -daemon
rabbitmq   9807      1  0 01:26 ?        00:02:19 /usr/lib64/erlang/erts-7.3/bin/beam -W w -A 64 -P 1048576 -t 5000000 -stbt db -K true -- -root
a /usr/lib/rabbitmq/lib/rabbitmq_server-3.6.5/ebin -noshell -noinput -s rabbit boot -sname rabbit -boot start_sasl -kernel inet_default_connect_
_logger false -rabbit error_logger {file,"/var/log/rabbitmq/rabbit.log"} -rabbit sasl_error_logger {file,"/var/log/rabbitmq/rabbit-sasl.log"} -r
t plugins_dir "/usr/lib/rabbitmq/lib/rabbitmq_server-3.6.5/plugins" -rabbit plugins_expand_dir "/var/lib/rabbitmq/mnesia/rabbit-plugins-expand"
tart_memsup false -mnesia dir "/var/lib/rabbitmq/mnesia/rabbit" -kernel inet_dist_listen_min 25672 -kernel inet_dist_listen_max 25672 -noshell -
rabbitmq   9878   9807  0 01:26 ?        00:00:00 inet_gethost 4
rabbitmq   9879   9878  0 01:26 ?        00:00:01 inet_gethost 4
root      20348  14835  0 07:17 pts/1    00:00:00 grep --color=auto rabbitmq
[root@smlz86 local]#
```

访问地址:http://192.168.159.8: 15672



Queues

All queues (6)

Pagination

Page 1 of 1 - Filter: Regex (?)(?) Displaying 6

Overview				Messages			Message rates		
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	tulingBootDelayQueue	D	idle	0	0	0			
/	tulingBootQueue	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	tulingBootQueue2	D	idle	0	0	0			
/	tulingBootQueue3	D	idle	0	0	0			
/	tulingClusterQueue	D	idle	0	0	0			
smlz-vhost	smlz-queue	D	idle	1	0	1	0.00/s	0.00/s	0.00/s

Add a new queue

Virtual host:	/
Name:	*
Durability:	Durable
Auto delete: (?)	No
Arguments:	= String

*****具体安装3.6.5的文档:https://www.cnblogs.com/sky-cheng/p/10709104.html

******卸载文档:https://www.cnblogs.com/kingsonfu/p/11023967.html

插件地址:https://www.rabbitmq.com/community-plugins.html

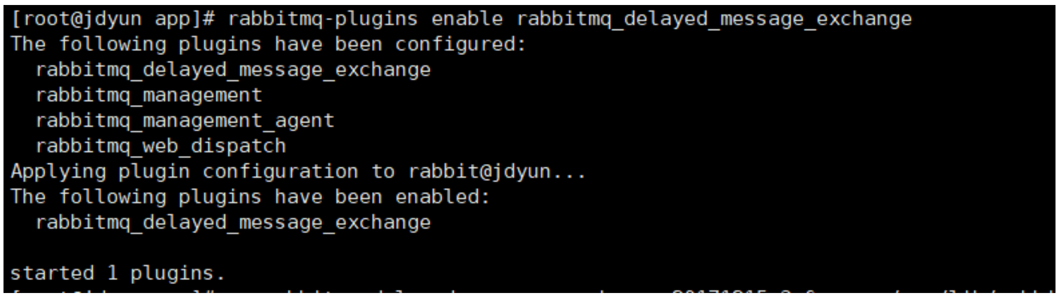
延时插件地址 :https://dl.bintray.com/rabbitmq/community-plugins/3.6.x/rabbitmq_delayed_message_exchange/

下载延时插件:wget https://dl.bintray.com/rabbitmq/community-plugins/3.6.x/rabbitmq_delayed_message_exchange/rabbitmq_delayed_message_exchange-20171215-3.6.x.zip

解压延时插件：unzip rabbitmq_delayed_message_exchange-20171215-3.6.x.zip

把延时插件拷贝到指定目录下:cp rabbitmq_delayed_message_exchange-20171215-3.6.x.ez /usr/lib/rabbitmq/lib/rabbitmq_server-3.7.5/plugins

启动延时插件: rabbitmq-plugins enable rabbitmq_delayed_message_exchange

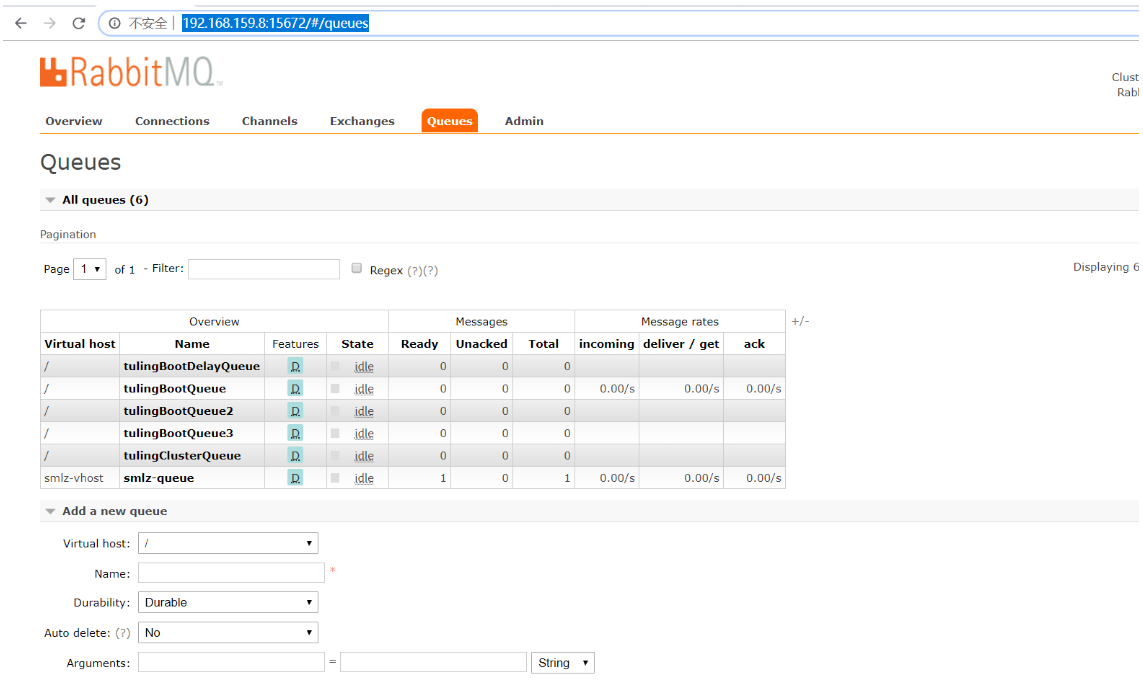


```
[root@jdyun app]# rabbitmq-plugins enable rabbitmq_delayed_message_exchange
The following plugins have been configured:
  rabbitmq_delayed_message_exchange
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@jdyun...
The following plugins have been enabled:
  rabbitmq_delayed_message_exchange

started 1 plugins.
```

八:命令行和管控台

开启管控台插件 rabbitmq-plugus rabbitmq_management 来开启管控台

测试连接： http://ip:15672(**来访问）用户名密码** guest/guest



不安全 | 192.168.159.8:15672/#/queues

RabbitMQ

Overview Connections Channels Exchanges Queues Admin

Queues

All queues (6)

Pagination

Page 1 of 1 - Filter: Regex (?)(?)

Displaying 6

Overview				Messages			Message rates		
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	tulingBootDelayQueue	D	idle	0	0	0			
/	tulingBootQueue	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	tulingBootQueue2	D	idle	0	0	0			
/	tulingBootQueue3	D	idle	0	0	0			
/	tulingClusterQueue	D	idle	0	0	0			
smlz-vhost	smlz-queue	D	idle	1	0	1	0.00/s	0.00/s	0.00/s

Add a new queue

Virtual host:	/
Name:	
Durability:	Durable
Auto delete: (?)	No
Arguments:	= String

8.1)**管理控制台命令**:

a:**起停服务命令**

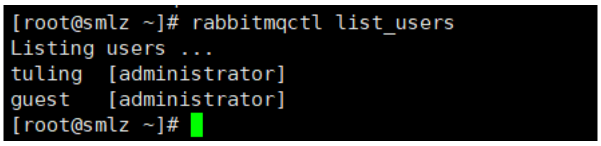
启动服务 rabbitmqctl start_app(**启动**rabbitmq**节点 保证需要**erlang**虚拟机节点起来才能执行**)

停止服务 rabbitmqctl stop_app(**停止**rabbtimq**节点，但是不会停止**erlang**节点**) rabbitmqctl stop(**都会停止**)

查看服务状态 rabbtimqctl status

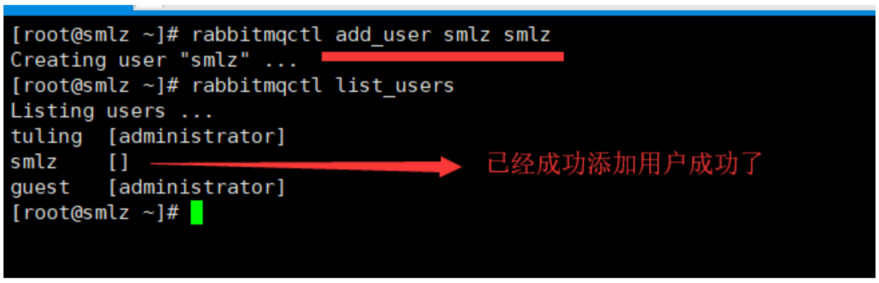
b:**用户操作命令**

查看所有用户列表: rabbitmq list_users



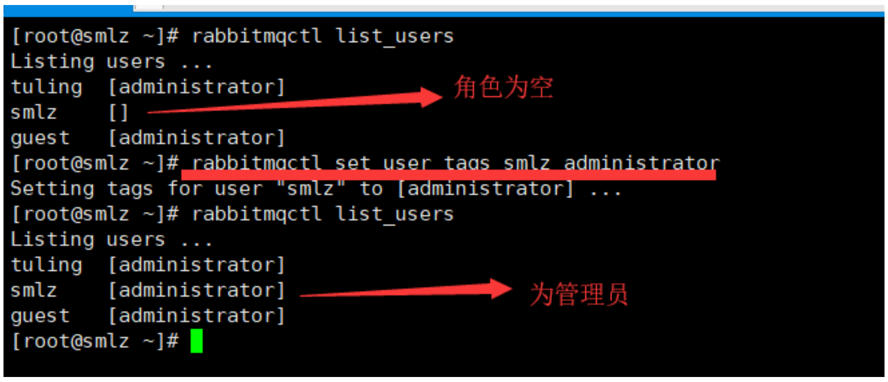
```
[root@smlz ~]# rabbitmqctl list_users
Listing users ...
tuling  [administrator]
guest   [administrator]
[root@smlz ~]#
```

添加用户 rabbitmqctl add_user smlz smlz



```
[root@smlz ~]# rabbitmqctl add_user smlz smlz
Creating user "smlz" ...
[root@smlz ~]# rabbitmqctl list_users
Listing users ...
tuling  [administrator]
smlz    []
guest   [administrator]
[root@smlz ~]#
```

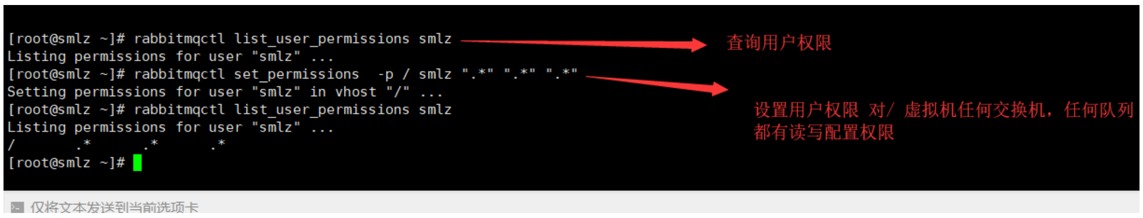
设置rabbitmq**用户的角色** rabbitmqctl set_user_tags smlz administrator



```
[root@smlz ~]# rabbitmqctl list_users
Listing users ...
tuling  [administrator]
smlz    []
guest   [administrator]
[root@smlz ~]# rabbitmqctl set_user_tags smlz administrator
Setting tags for user "smlz" to [administrator] ...
[root@smlz ~]# rabbitmqctl list_users
Listing users ...
tuling  [administrator]
smlz    [administrator]
guest   [administrator]
[root@smlz ~]#
```

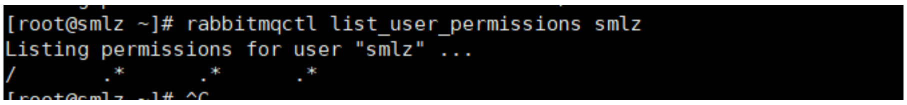
为用户设置权限: rabbitmqctl set_permissions -p / smlz ".*" ".*" ".*"

rabbitmqctl set_permissions -p <**虚拟机**> <**用户名**> ".*" ".*" ".*"



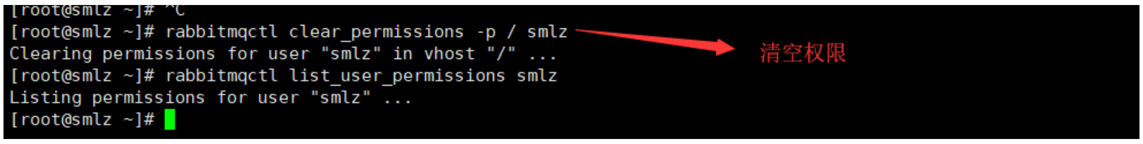
```
[root@smlz ~]# rabbitmqctl list_user_permissions smlz
Listing permissions for user "smlz" ...
[root@smlz ~]# rabbitmqctl set_permissions  -p / smlz ".*" ".*" ".*"
Setting permissions for user "smlz" in vhost "/" ...
[root@smlz ~]# rabbitmqctl list_user_permissions smlz
Listing permissions for user "smlz" ...
/       .*      .*      .*
[root@smlz ~]#
```

列出用户权限: rabbitmqctl list_user_permissions smlz



```
[root@smlz ~]# rabbitmqctl list_user_permissions smlz
Listing permissions for user "smlz" ...
/       .*      .*      .*
[root@smlz ~]# ^C
```

清除用户权限 rabbitmqctl clear_permissions -p <**虚拟机**> <**用户名**>



```
[root@smlz ~]# ^C
[root@smlz ~]# rabbitmqctl clear_permissions -p / smlz
Clearing permissions for user "smlz" in vhost "/" ...
[root@smlz ~]# rabbitmqctl list_user_permissions smlz
Listing permissions for user "smlz" ...
[root@smlz ~]#
```

删除用户 rabbitmqctl delete_user root

修改密码 rabbitmqctl change_password **用户名 新密码**

c:**虚拟主机操作**

rabbitmqctl add_vhost /cloudmall **增加一个虚拟主机**

rabbitmqctl list_vhosts; **查看所有的虚拟主机**

rabbitmqctl list_permissions -p /cloudmall **查看虚拟主机的权限**

rabbitmqctl delete_vhost /cloudmall **删除虚拟主机**

D:**操作队列命令**

rabbitmqctl list_queues **查询所有队列**

rabbitmqctl -p vhostpath purge_queue blue **清除队列消息**

E:**高级命令**

rabbitmqctl reset **移除所有数据 该命令需要在** rabbitmqctl stop_app**命令之后才执行(也就是说 在服务停止后)**

rabbitmqctl join_cluster <cluster_node> [--ram] **组成集群命令**

rabbitmqctl cluster_status **查看集群状态**

rabbitmqctl change_cluster_node_type dist|ram **修改集群节点存储数据模式**

rabbitmqctl forget_cluster_node [--offline]**忘记节点 (摘除节点)**

rabbitmqctc rename_cluster_node oldnode1 newnode1 oldnode2 newnode2 **修改节点名称**

九:**消费者 生产者模型(使用**java**连接**mq)

A:**生产者**

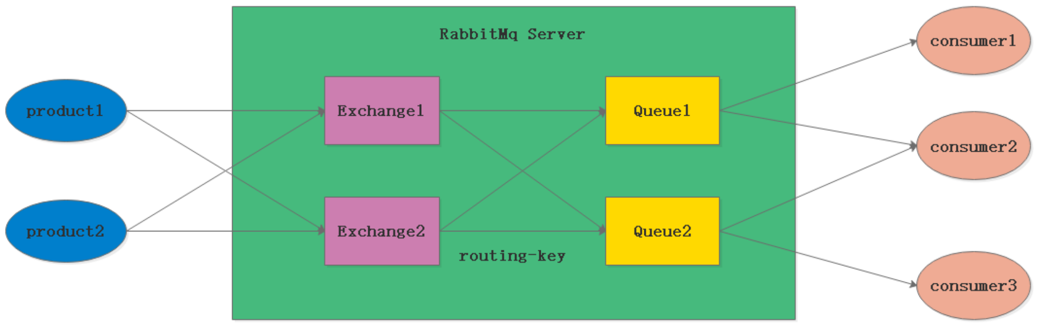
```
public class RabbitmqProducter {

    public static void main(String[] args) throws IOException, TimeoutException {
        //1:创建连接工厂
        ConnectionFactory connectionFactory = new ConnectionFactory();

        //2设置连接工厂的属性
        connectionFactory.setHost("192.168.159.8");
        connectionFactory.setPort(5672);
        connectionFactory.setVirtualHost("tuling");
        connectionFactory.setUsername("smlz");
        connectionFactory.setPassword("smlz");

        //3:通过连接工厂创建连接对象
        Connection connection = connectionFactory.newConnection();

        //4:通过连接创建channel
        Channel channel = connection.createChannel();

        //5:通过channel发送消息
        for(int i=0;i<5;i++) {
            String message = "hello--"+i;
            /**
             * 老师以前讲过说我们的消息会发送的exchange上,
             * 但是在这里我们没有指定交换机?那我们的消息发送到哪里了? ? ? ?
             * The default exchange is implicitly bound to every queue, with a routing key equal to the queue name.
             * It is not possible to explicitly bind to, or unbind from the default exchange. It also cannot be deleted.
             * 说明:加入我们消息发送的时候没有指定具体的交换机的话，那么就会发送到rabbimtq指定默认的交换机上,
             * 那么该交换机就会去根据routing_key 查找对应的queueName 然后发送的该队列上.
             *
             */
            channel.basicPublish("","tuling-queue-01",null,message.getBytes());
        }

        //6:关闭连接
        channel.close();
        connection.close();
    }
```

B：**消费者**:

```
public class RabbitmqConsumer {

    public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {

        //创建连接工厂
        ConnectionFactory connectionFactory  = new ConnectionFactory();
        connectionFactory.setHost("192.168.159.8");
        connectionFactory.setPort(5672);
        connectionFactory.setVirtualHost("tuling");
        connectionFactory.setUsername("smlz");
        connectionFactory.setPassword("smlz");

        //创建连接
        Connection connection = connectionFactory.newConnection();

        //创建一个channel
        Channel channel = connection.createChannel();

        //声明队列
        String queueName = "tuling-queue-01";
        /**
         * queue:队列的名称
         * durable:是否持久化, 队列的声明默认是存放到内存中的，如果rabbitmq重启会丢失，如果想重启之后还存在就要使队列持久
         * 保存到Erlang自带的Mnesia数据库中，当rabbitmq重启之后会读取该数据库
         * exclusive:当连接关闭时connection.close()该队列是否会自动删除;
         * 二：该队列是否是私有的private，如果不是排外的，可以使用两个消费者都访问同一个队列,
         * 没有任何问题，如果是排外的，会对当前队列加锁，其他通道channel是不能访问的，如果强制访问会报异常
         * com.rabbitmq.client.ShutdownSignalException: channel error; protocol method: #method<channel.close>(reply-c
         * 一般等于true的话用于一个队列只能有一个消费者来消费的场景
         * autodelete:是否自动删除，当最后一个消费者断开连接之后队列是否自动被删除，可以通过RabbitMQ Management，
         * 查看某个队列的消费者数量，当consumers = 0时队列就会自动删除
         */
        channel.queueDeclare(queueName,true,false,true,null);

        //创建消费者
        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
        channel.basicConsume(queueName,true,queueingConsumer);

        while (true) {
            QueueingConsumer.Delivery delivery = queueingConsumer.nextDelivery();
            String reserveMsg = new String(delivery.getBody());
            System.out.println("消费消息:"+reserveMsg);
        }
    }
}
```

十:Rabbitmq**交换机详解**

1：**作用**:**接受生产者的消息，然后根据路由键 把消息投递到跟交换机绑定的对应的队列上**



2：**交换机的属性**:

Name:**交换机的名称**

Type:**交换机的类型**,direct,topic,fanout,headers

Durability:**是否需要持久化**

autodelete:**假如没有队列绑定到该交换机，那么该交换机会自动删除**

Internal:**当前交换机是否用户**rabbitmq**内部使用不常用,默认为**false

Argurements:**扩展参数,用户扩展**AMQP **定制化协议**

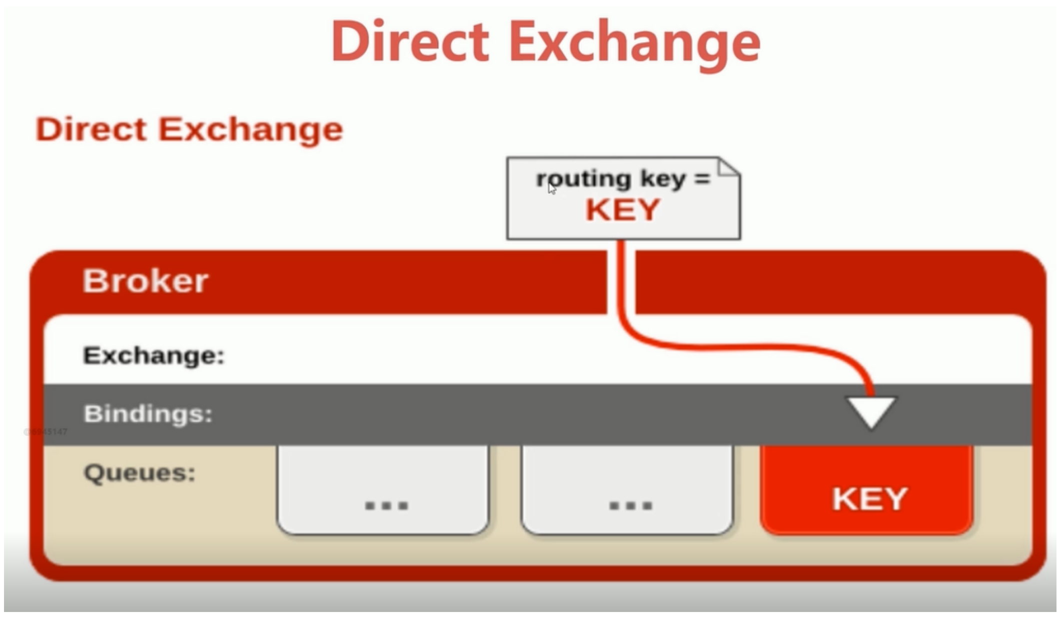
3：**交换机的类型**

3.1)**直连交换机**:direct exchange

所以发送的direct exhchange **的消息都会被投递到与**routekey**名称(与队列名称)相同的**queue**上**

:direct模式下，可以使用**rabbitmq**自定**exchange----> default exchange **所以不需要交换机和任何队列绑定,**

消息将会投递到route_key名称和队列名称相同的队列上



代码演示:

直连交换机生产者

```
public class DirectExchangeProductor {

    public static void main(String[] args) throws IOException, TimeoutException {

        //创建连接工厂
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setHost("192.168.159.8");
        connectionFactory.setPort(5672);
        connectionFactory.setVirtualHost("tuling");
        connectionFactory.setUsername("smlz");
        connectionFactory.setPassword("smlz");

        //创建连接
        Connection connection = connectionFactory.newConnection();

        //创建channel
        Channel channel = connection.createChannel();

        //定义交换机名称
        String exchangeName = "tuling.directchange";

        //定义routingKey
        String routingKey = "tuling.directchange.key";

        //消息体内容
        String messageBody = "hello tuling ";
        channel.basicPublish(exchangeName,routingKey,null,messageBody.getBytes());




    }
}
```

直接交换机消费者

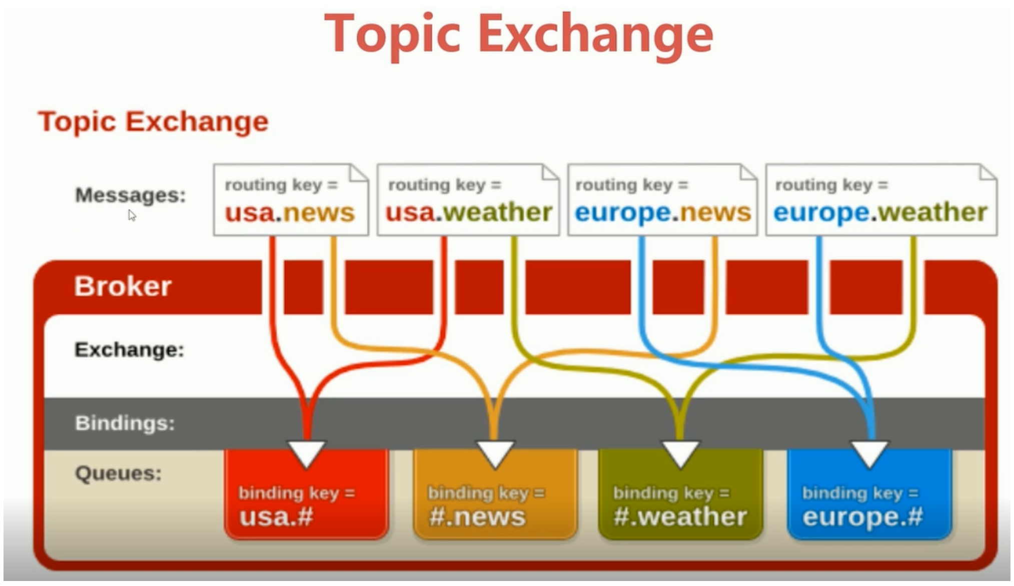
```
public class DirectExchangeConsumer {

    public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {

        //创建连接工厂
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setHost("192.168.159.8");
        connectionFactory.setPort(5672);
        connectionFactory.setVirtualHost("tuling");
        connectionFactory.setUsername("smlz");
        connectionFactory.setPassword("smlz");

        //创建连接
        Connection connection = connectionFactory.newConnection();

        //创建channel
        Channel channel = connection.createChannel();

        String exchangeName = "tuling.directchange";
        String exchangeType = "direct";
        String queueName = "tuling.directqueue";
        String routingKey = "tuling.directchange.key";
        /**
         * 声明一个交换机
         * exchange:交换机的名称
         * type:交换机的类型 常见的有direct,fanout,topic等
         * durable:设置是否持久化。durable设置为true时表示持久化，反之非持久化.持久化可以将交换器存入磁盘，在服务器重启的时
         * autodelete:设置是否自动删除。autoDelete设置为true时，则表示自动删除。自动删除的前提是至少有一个队列或者交换器与
         * 不能错误的理解—当与此交换器连接的客户端都断开连接时，RabbitMq会自动删除本交换器
         * arguments:其它一些结构化的参数，比如：alternate-exchange
         */
        channel.exchangeDeclare(exchangeName,exchangeType,true,false,null);

        /**
         * 声明一个队列
         * durable:表示rabbitmq关闭删除队列
         * autodelete:表示没有程序和队列建立连接 那么就会自动删除队列
         *
         */
        channel.queueDeclare(queueName,true,false,false,null);

        /**
         * 队里和交换机绑定
         */
        channel.queueBind(queueName,exchangeName,routingKey);

        /**
         * 创建一个消费者
         */
        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);

        /**
         * 开始消费
         */
        channel.basicConsume(queueName,true,queueingConsumer);

        while (true) {
            QueueingConsumer.Delivery delivery = queueingConsumer.nextDelivery();
            String reciverMessage = new String(delivery.getBody());
            System.out.println("消费消息:-----"+reciverMessage);
        }

    }
}
```

3.2)**主题交换机** TopicExchange

就是在队列上绑到top **交换机上的路由**key **可以是通过通配符来匹配的通配符的规则是**

比如: log.# **:** **可以匹配一个单词 也可以匹配多个单词** **比如** log.# **可以匹配**log.a log.a.b log.a.b

log.* **可以匹配一个单词** **比如** log.* **可以匹配**log.a **但是不可以匹配**log.a.b



代码演示 :topic exchange **生产者**

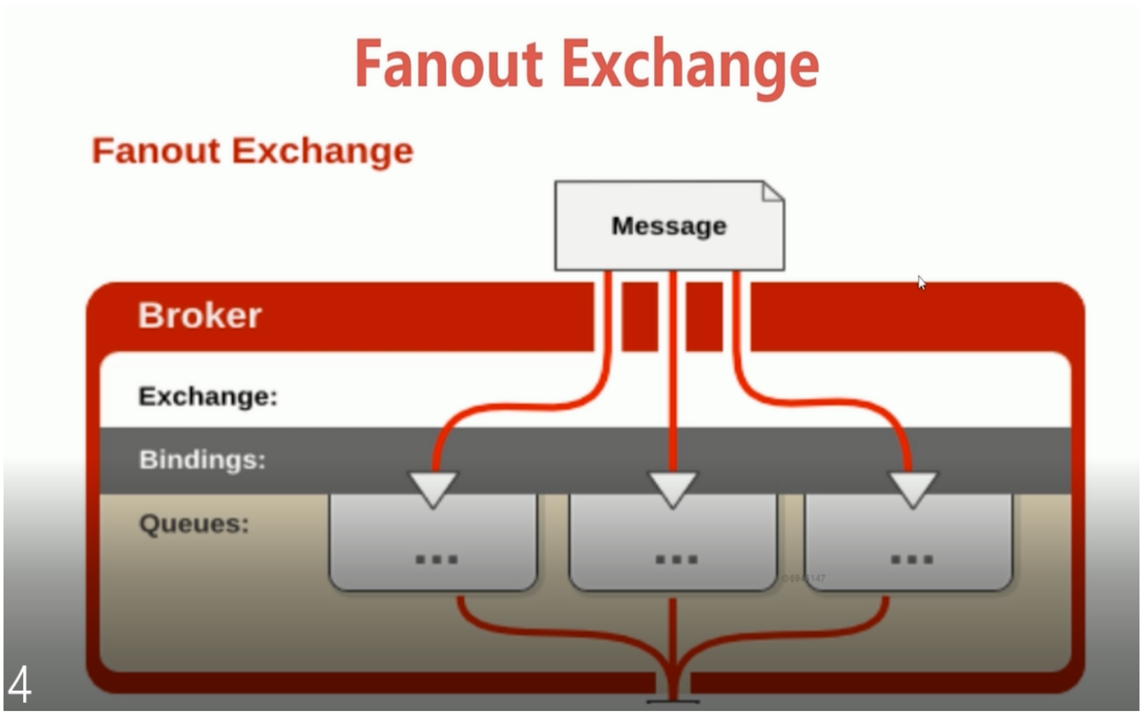
```
public static void main(String[] args) throws IOException, TimeoutException {
ConnectionFactory connectionFactory = new ConnectionFactory();
 connectionFactory.setVirtualHost("cloudmall");
 connectionFactory.setHost("47.104.128.12");
 connectionFactory.setPort(5672);
Connection connection = connectionFactory.newConnection();
Channel channel = connection.createChannel();
String topExchangeName = "top.exchange";
String routingKey1 = "top.key.1";
 String routingKey2 = "top.key.2";
channel.basicPublish(topExchangeName,routingKey1,null,"测试交换机".getBytes());
 channel.basicPublish(topExchangeName,routingKey2,null,"测试交换机".getBytes());
channel.close();
 connection.close();
}
```

代码演示: topic exhcange **消费者**

```
public class Top4Consumer {
public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {
 ConnectionFactory connectionFactory = new ConnectionFactory();
 connectionFactory.setVirtualHost("cloudmall");
 connectionFactory.setHost("47.104.128.12");
 connectionFactory.setPort(5672);
Connection connection = connectionFactory.newConnection();
Channel channel = connection.createChannel();
String topicExchangeName = "top.exchange";
 //声明一个交换机
 channel.exchangeDeclare(topicExchangeName,"topic",true,true,false,null);
//声明一个队列
 channel.queueDeclare("top.queue",true,false,true,null);
//队列绑定到交换机
 channel.queueBind("top.queue",topicExchangeName,"top.#");
QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
//参数：队列名称、是否自动ACK、Consumer
 channel.basicConsume("top.queue", true, queueingConsumer);
while (true) {
 QueueingConsumer.Delivery delivery = queueingConsumer.nextDelivery();
 System.out.println(new String(delivery.getBody()));
}
 }
```

3.3)**扇形交换机**(fanout exchange)

就是消息通过从交换机到队列上不会通过路由key **所以该模式的速度是最快的 只要和交换机绑定的那么消息就会被分发到与之绑定的队列上**



代码演示 扇形交换机 模式下的生产着

```
public static void main(String[] args) throws IOException, TimeoutException {
 ConnectionFactory connectionFactory = new ConnectionFactory();
 connectionFactory.setHost("47.104.128.12");
 connectionFactory.setPort(5672);
 connectionFactory.setVirtualHost("cloudmall");
Connection connection = connectionFactory.newConnection();
 Channel channel = connection.createChannel();
String fanoutExchangeName = "test.fanout.exchange";
 String routingKey = "test.fanout.key";
 channel.basicPublish(fanoutExchangeName,routingKey,null,"測試扇形交換机。。。。。。".getBytes());
 }
```

扇形交换机模式下的消费者

```
public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {
 ConnectionFactory connectionFactory = new ConnectionFactory();
 connectionFactory.setHost("47.104.128.12");
 connectionFactory.setPort(5672);
 connectionFactory.setVirtualHost("cloudmall");
Connection connection = connectionFactory.newConnection();
 Channel channel = connection.createChannel();
//创建交换机
 channel.exchangeDeclare("test.fanout.exchange","fanout",true,true,false,null);
 //创建队列
 channel.queueDeclare("test.fanout.queue",true,false,true,null);
//绑定队列
 channel.queueBind("test.fanout.queue","test.fanout.exchange","");
QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
channel.basicConsume("test.fanout.queue",true,queueingConsumer);
while (true) {
 QueueingConsumer.Delivery delivery = queueingConsumer.nextDelivery();
 System.out.println(new String(delivery.getBody()));
}
```

}

十一: 队列,绑定虚拟主机，消息

绑定: exchange **与之间的连接关系(通过路由规则)**

队列:**用来存储消息的实体**

队列的属性: durability **消息是否被持久化**

AutoDelete :**表示最后一个监听被移除那么该队列就会被删除**

消息:**用来生产着和消费者之间传递数据的**

消息属性: **包括消息体**body **和属性** properties

常用属性:delivery mode ，headers，content_type(**消息类型**) content_encoding(**消息编码**),priporty(**消息优先级**)

correntlation_id(**最为消息唯一的**id),reply_to（**消息失败做重回队列**）,expiretion(**消息的过期时间**),message_id(**消息**id);

timestamp,type,user_id ， app_id,cluster_id等

自定义消息属性的消费端代码:

```
public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {
 ConnectionFactory connectionFactory = new ConnectionFactory();
 connectionFactory.setHost("47.104.128.12");
 connectionFactory.setPort(5672);
 connectionFactory.setVirtualHost("/");
//2 通过连接工厂创建连接
 Connection connection = connectionFactory.newConnection();
//3 通过connection创建一个Channel
 Channel channel = connection.createChannel();
//4 声明（创建）一个队列
 String queueName = "test001";
 channel.queueDeclare(queueName, true, false, false, null);
//5 创建消费者
 QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
//6 设置Channel
 channel.basicConsume(queueName, true, queueingConsumer);
while(true){
 //7 获取消息
 Delivery delivery = queueingConsumer.nextDelivery();
 String msg = new String(delivery.getBody());
 System.out.println("消费端: " + msg);
 System.out.println(delivery.getProperties());
 System.out.println(delivery.getProperties().getHeaders());
 }
}
```

自定义消息属性的生产着

```
public static void main(String[] args) throws IOException, TimeoutException {
 //创建连接工厂
 ConnectionFactory connectionFactory = new ConnectionFactory();
 connectionFactory.setHost("47.104.128.12");
 connectionFactory.setPort(5672);
 connectionFactory.setVirtualHost("/");
//创建连接
 Connection connection = connectionFactory.newConnection();
 //创建channel
 Channel channel = connection.createChannel();
 Map<String,Object> extraMap = new HashMap<>();
 extraMap.put("k1","v1");
 extraMap.put("k2","v2");
 /**
 * 附带额外信息的消息体
 */
 AMQP.BasicProperties basicProperties = new AMQP.BasicProperties.Builder()
 .deliveryMode(2)//2为持久化,1 不是持久化
 .appId("测试appid")
 .clusterId("测试集群id")
 .contentType("application/json")
 .contentEncoding("UTF-8")
 .headers(extraMap).build();
 for (int i = 0; i < 10; i++) {
 String tragetMsg = "这是我的第【"+(i+1)+"】条消息";
 channel.basicPublish("", "test001", basicProperties, tragetMsg.getBytes());
}
channel.close();
 connection.close();
 }
```