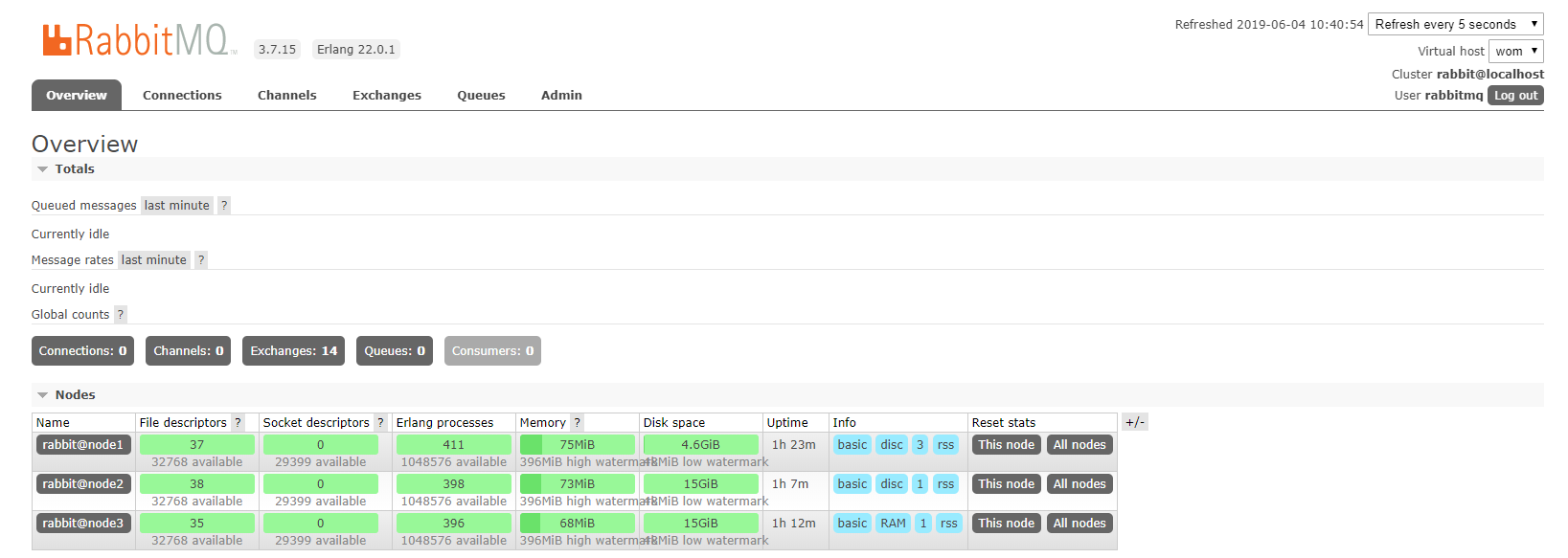
# Rabbitmq实例与操作手册

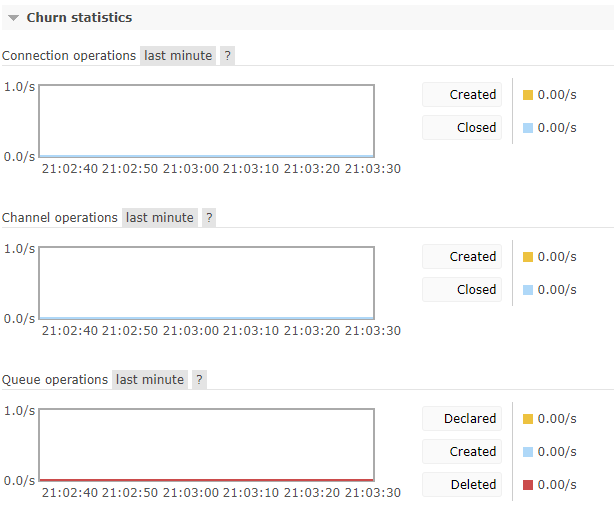
## RabbitMQ概括

## RabbitMQ 概观

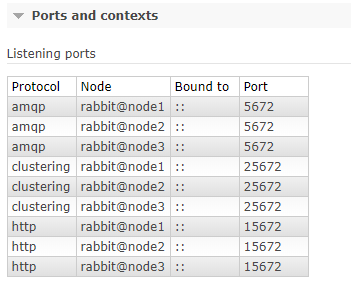


管理平台上有版本号和erlang语言的版本 用户，集群的说明。

1. 首页是overview，能看到rabbitMQ的整体情况，可以看到连接数，交换器数量，队列数量，消费者数量等。
2. 下面有“Node”项，这里面显示的是RabbitMQ的服务节点。其中列“info”disc是为磁盘节点，ram是为内存节点。
3. “Churn statistics”这里显示连接，交换器，队列最后操作时间表



1. 然后下面的“Ports and contexts”展示的是端口信息：



这里一共有三个端口，其中5672是amqp协议的端口，15672是RabbitMQ的管理工具端口(这就是我们现在操作的)，25672是做集群的端口。

1. 最下面的就是进行导入导出RabbitMQ的基本信息:

导出的是json格式的文本，内容类似我们执行“rabbitmqctl status”指令拿到的结果。

导入即是在导出文件的基础上做修改，进行导入，参数合理的话会进行基础配置的修改。

## RabbitMQ Connetions连接信息

“Connections”(连接)选项卡下面的信息：



在这里可以看客户端连接RabbitMQ服务的信息。目前尚未有客户端连接，所以上面看不到连接信息

## RabbitMQ Channels通道信息

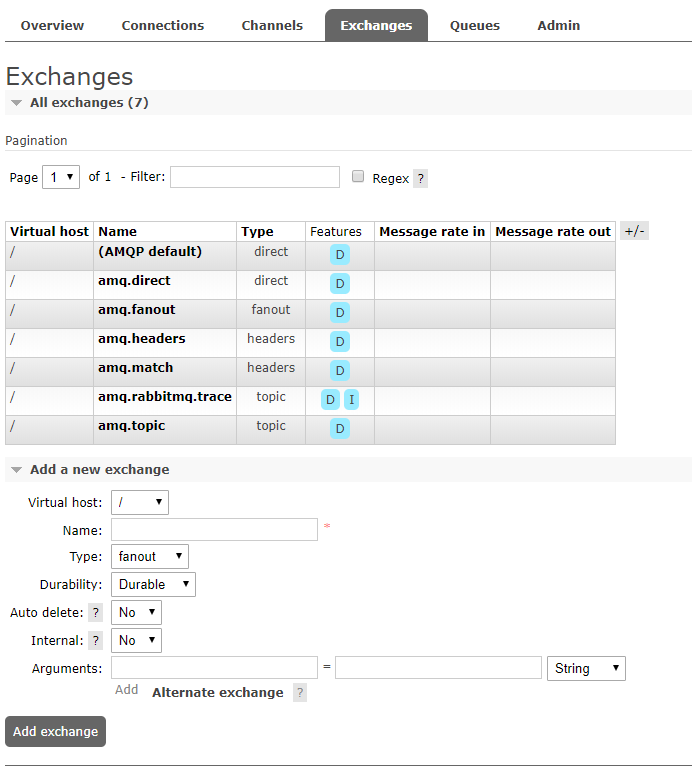
“Channels”(通道)选项卡下面的信息：



在这里可以看客户端连接RabbitMQ服务通道的信息。目前尚未有客户端连接，所以上面看不到连接通道信息

## RabbitMQ Exchanges交换器信息

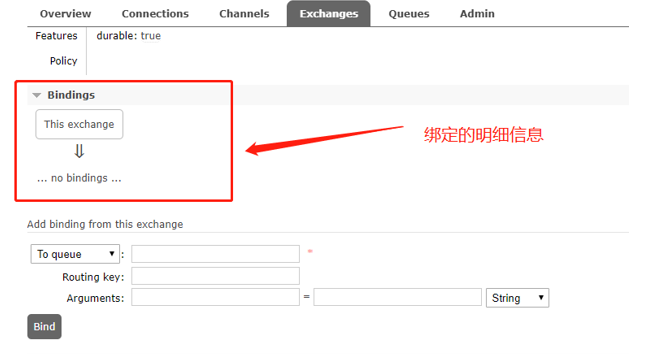
“Exchanges”(交换器)选项卡下面的信息：



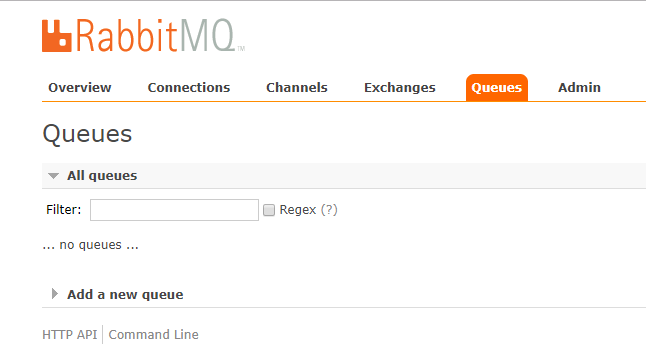
1. Virtual host:虚拟主机，相对就数据库的库名,系统有一个默认的虚拟主机“/”（默认库）. 是MQ启动时自动化生成
2. Name：新建交换器的名字；
3. Type：新建交换器的类型，交换器类型分为四种（fanout、topic、direct、headers）；
4. Durability：消息持久化，分为两种（Durable持久化、Transient暂时）；
5. Auto Delete：是否自动删除，true为自动删除，删除的前提是至少有一个队列或者交换器与这个交换器绑定，之后所有与这个交换器绑定的队列或者交换器都与此解绑,客户端都断开时自动删除；
6. Internal：是否内置，true表示内置交换器，客户端程序无法直接发送消息到这个交换器中，只能通过交换器路由到交换器这种方式；
7. Argument：其它一些结构化参数。

**选好自己想要的类型后点击add exchange即可创建新的交换器。**

**交换器绑定：单击交换器列表中的交换器名称即可进入交换器绑定界面（如图）。**



## RabbitMQ Queue队列信息

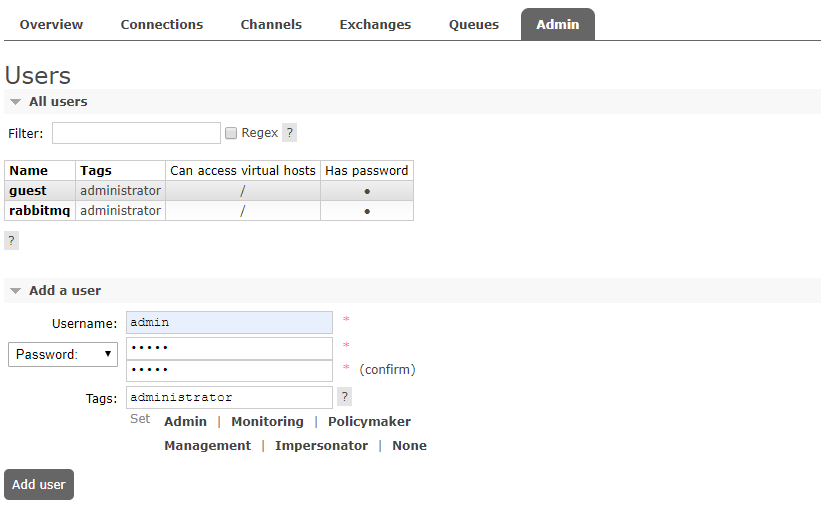


可以点击“add a new queue”增加队列

## RabbitMQ 用户处理

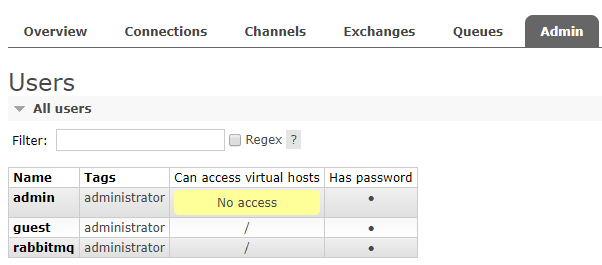
1. 增加用户信息

在管理平台添加用户,默认会提供一个默认用户guest，密码也是guest，线上环境需要创建一个新用户。首先切换到Admin标签页，可以查看或添加用户，添加用户时，可指定Tags，相当于角色，会拥有对应的权限：

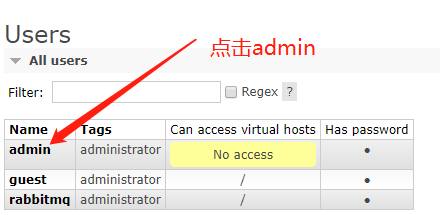


1. Name：用户名字；
2. tags：用户权限；
3. Can access virtual hosts：受权的虚拟机；
4. Has password：是有密码格式；

**填写好用户名称、密码、权限之后add user即可增加用户。**

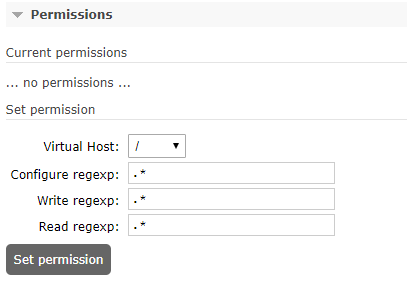


1. 修改用户信息

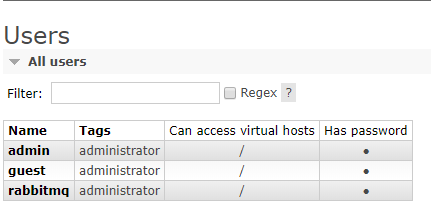


点击用户列表的用户名，可分配权限、编辑或删除用户，分配权限时，可细化到某个virtual下的某个topic，并按读、写、配置类别进行分开：

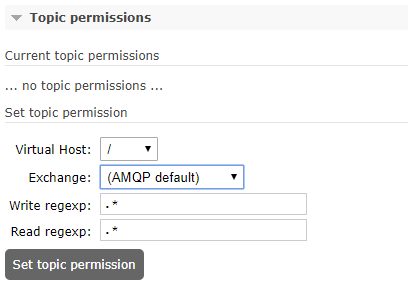
1. 设置授权的权限，分别为配置、写、读的权限



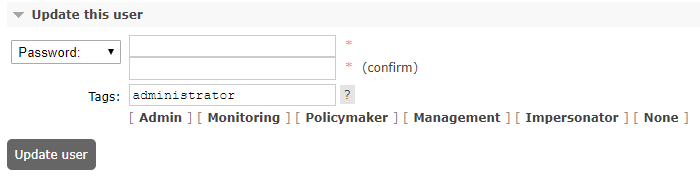
设置之后，返回时可看到



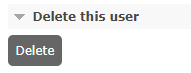
1. 分配权限时，可细化到某个virtual下的某个topic



1. 可分配权限、编辑、重置密码

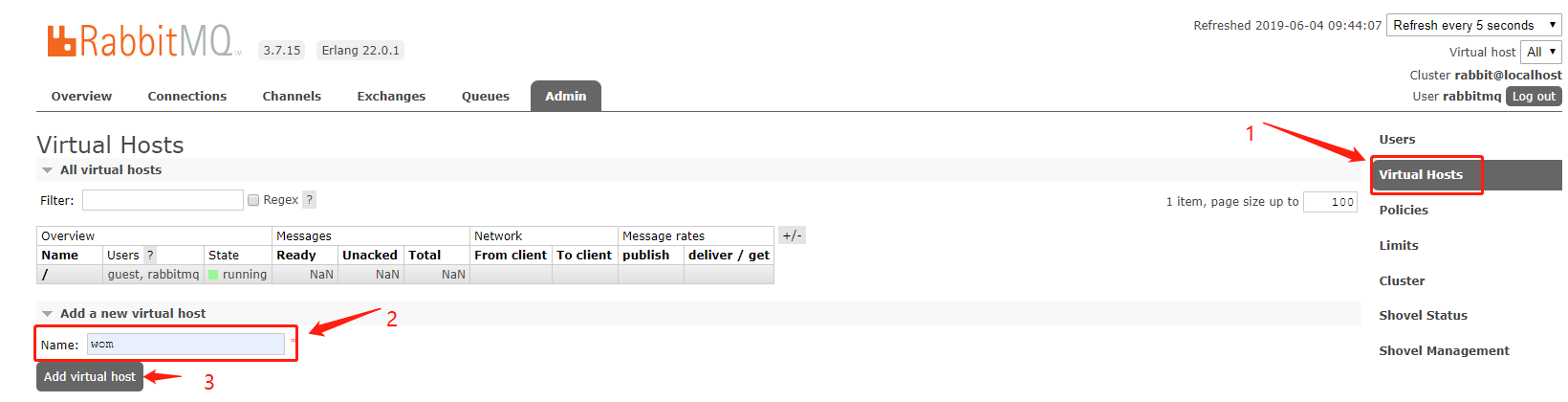


1. 删除用户

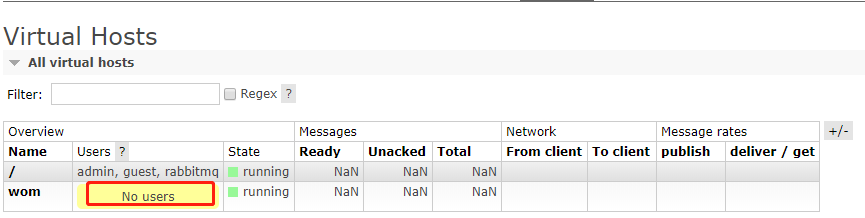


## RabbitMQ virtual-host 处理

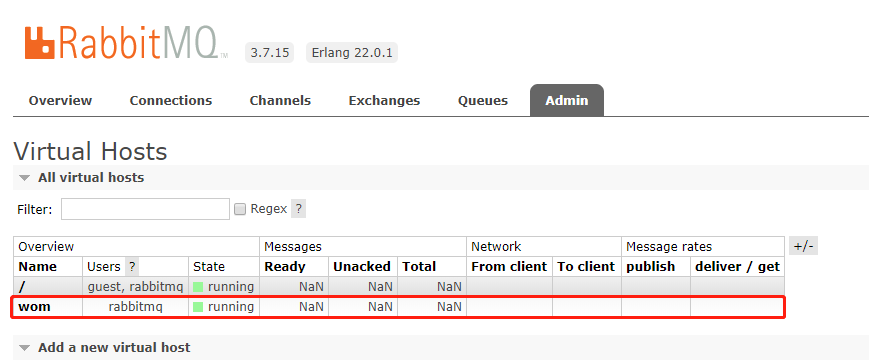
1. 增加virtual-host信息



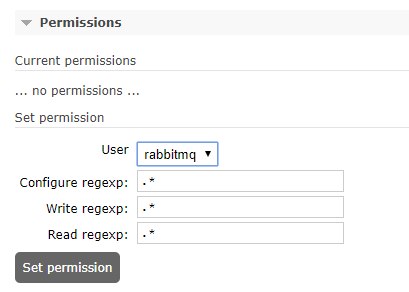
**点击“add virtual host”增虚拟主机**



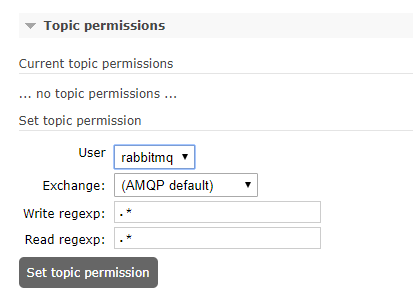
点击“wom“ 的虚拟机之后可修改virtual的信息，授权给rabbitm之后效果如下图



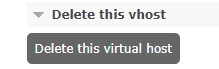
1. 修改virtual-host信息
2. 设置虚拟主机的用户配置、写、读权限



1. 分配权限时，可细化到某个user下的某个topic



1. 删除

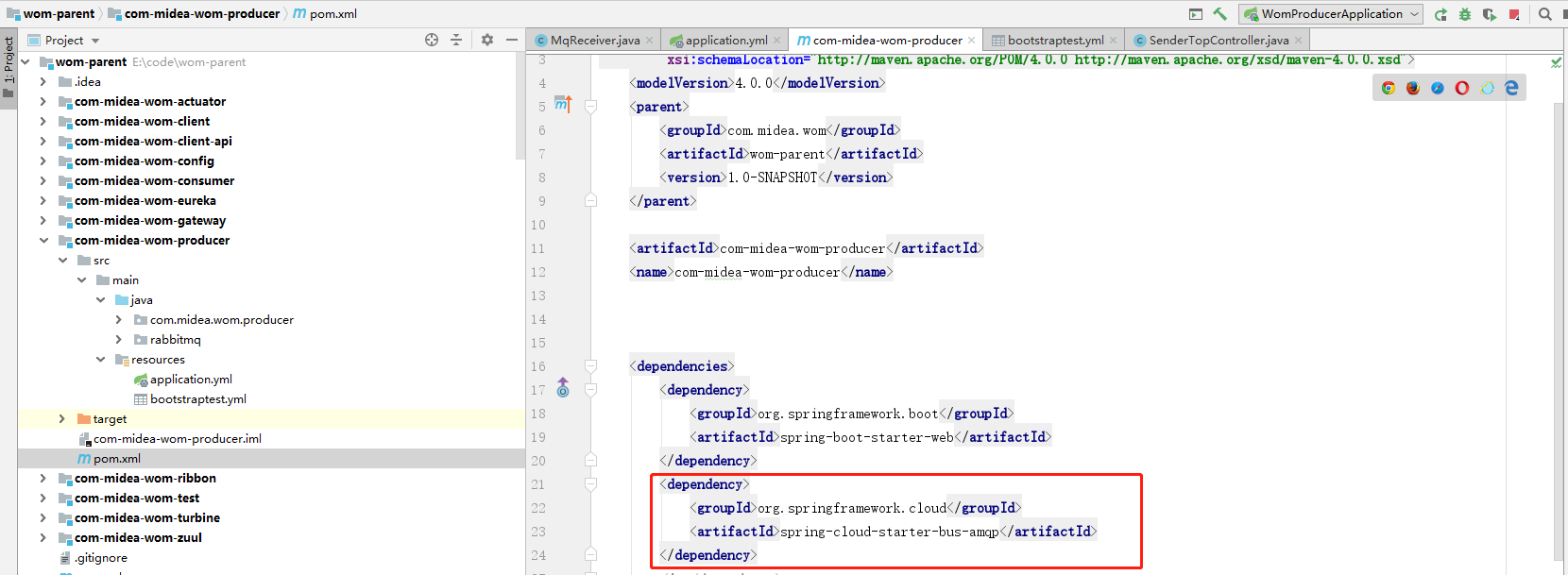


在页面中可以修改或者删除virtual-host

## 创建工程

## 创建一个生产者应用

1. 创建一个基础的Spring Boot工程，命名为：com-midea-wom-producer
2. 编辑pom.xml中的依赖关系，引入spring-cloud-starter-bus-amqp对RabbitMQ的支持，具体如下



<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-bus-amqp </artifactId>   
 </dependency>  
</dependencies>

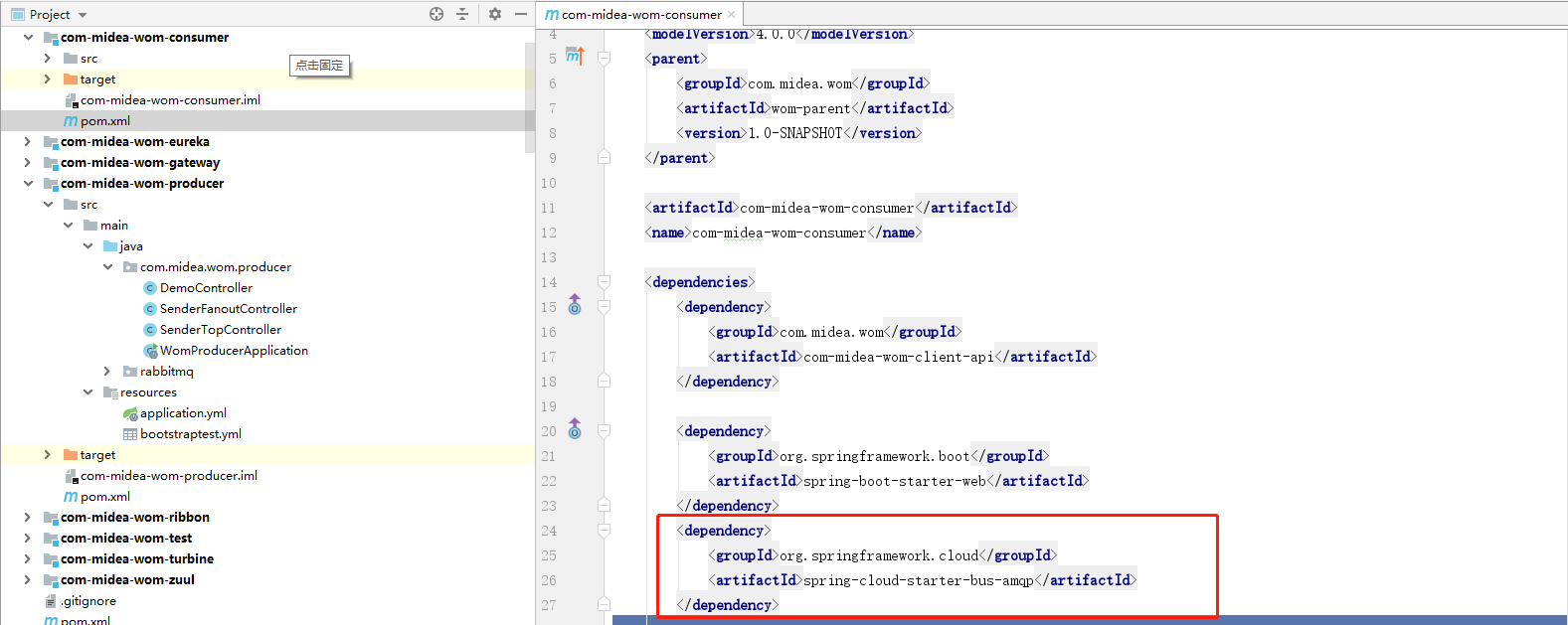
1. 配置rabbitmq连接信息



**server**:  
 **port**: 58086  
**spring**:  
 application:  
 *#应用名称* **name**: wom-producer-server  
 **rabbitmq**:  
 *#rabbitm连接地址，多个地址用逗号分开* **addresses**: amqp://10.73.179.98:5672,10.73.179.105:5672,10.73.179.109:5672  
 *#rabbitm用户* **username**: rabbitmq  
 *#rabbitmq密码* **password**: rabbitmq  
 *#虚拟机,相当于数据库的库* **virtual-host**: wom

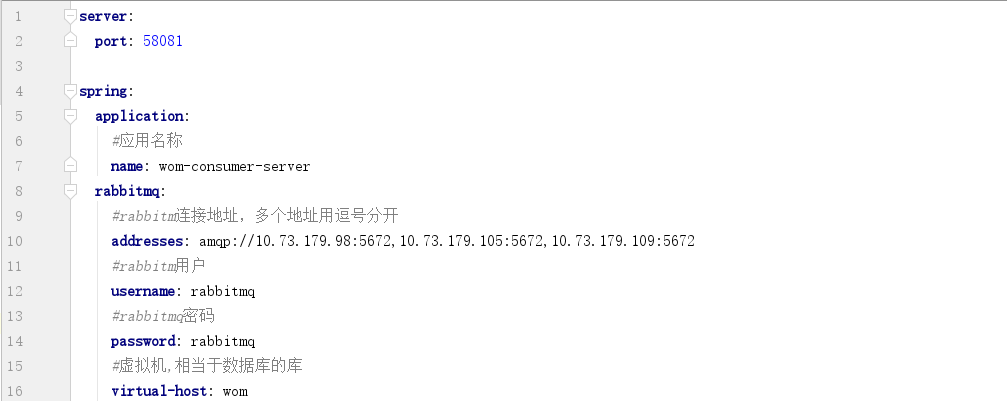
## 创建一个消费者应用

1. 创建一个基础的Spring Boot工程，命名为：com-midea-wom-consumer
2. 编辑pom.xml中的依赖关系，引入spring-cloud-starter-bus-amqp对RabbitMQ的支持，具体如下：



<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-bus-amqp </artifactId>   
 </dependency>  
</dependencies>

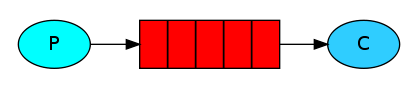
1. 配置rabbitmq连接信息



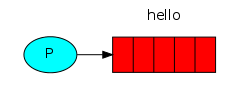
server:  
 **port**: 58081  
**spring**:  
 **application**:  
 *#应用名称* **name**: wom-consumer-server  
 **rabbitmq**:  
 *#rabbitm连接地址，多个地址用逗号分开* **addresses**: amqp://10.73.179.98:5672,10.73.179.105:5672,10.73.179.109:5672  
 *#rabbitm用户* **username**: rabbitmq  
 *#rabbitmq密码* **password**: rabbitmq  
 *#虚拟机,相当于数据库的库* **virtual-host**: wom

## 实例

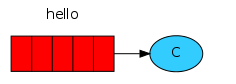
## ["Hello World!"](https://www.rabbitmq.com/tutorials/tutorial-one-python.html)



Sending

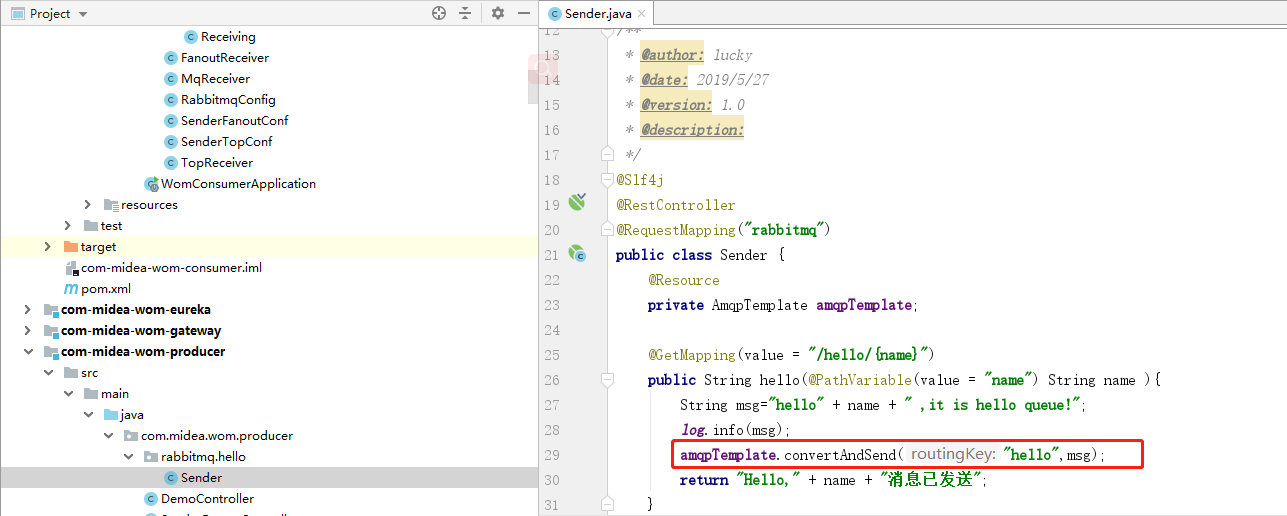


Receiving



Hello world队列是简单的一进一出队列，生产者发送一条信息之后，消费者就拉取列表数据，消息根据队列名称为“hello”产生消息通信。

* 消息发送



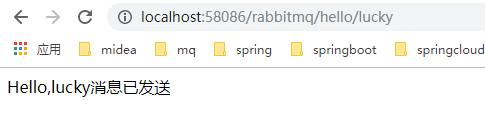
@Slf4j  
@RestController  
@RequestMapping(**"rabbitmq"**)  
**public class** Sender {  
 @Resource  
 **private** AmqpTemplate **amqpTemplate**;  
 @GetMapping(value = **"/hello/{name}"**)  
 **public** String hello(@PathVariable(value = **"name"**) String name ){  
 String msg=**"hello"** + name + **" ,it is hello queue!"**;  
 ***log***.info(msg);  
 **amqpTemplate**.convertAndSend(**"hello"**,msg);  
 **return "Hello,"** + name + **"消息已发送"**;  
 }  
}

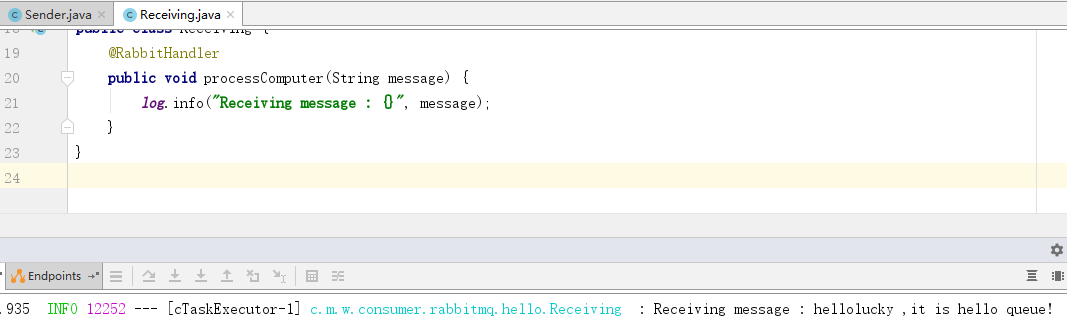
* 消息接收



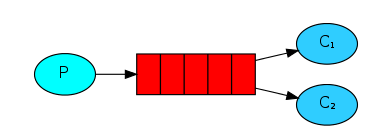
@Slf4j  
@Component  
*/\*\*  
 \* queuesToDeclare：用于判断列表value="hello是否存在，不存在时自动创建"hello"  
 \* durable:是否持久化  
 \* exclusive：是否排他性  
 \* autoDelete：是否自动删除  
 \*/*@RabbitListener(queuesToDeclare =@Queue(value = **"hello"**,durable=**"true"**,exclusive=**"false"**,autoDelete=**"false"***/\*,arguments = @Argument(name = "x-message-ttl, 10000")\*/*))  
**public class** Receiving {  
 @RabbitHandler  
 **public void** processComputer(String message) {  
 ***log***.error(**"Receiving message : {}"**, message);  
 **throw new** RuntimeException(**"Message consumer failed!"**);  
 }  
}

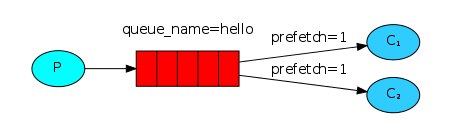
生产者推送信息后，消费方就收到队列hello的信息





## [Work queues](https://www.rabbitmq.com/tutorials/tutorial-two-python.html)





由生产者发送到队列work的消息都会被转发到与该Queue消费者的一方消费。

* 消息发送

@Slf4j  
@RestController  
@RequestMapping(**"rabbitmq"**)  
**public class** Work {  
 @Resource  
 **private** AmqpTemplate **amqpTemplate**;  
 @GetMapping(value = **"/work/{name}"**)  
 **public** String hello(@PathVariable(value = **"name"**) String name ){  
 String msg=**"hello"** + name + **" ,it is work queue!"**;  
 ***log***.info(msg);  
 **for** (**int** i = 0; i < 30; i++) {  
 **amqpTemplate**.convertAndSend(**"work"**,msg + **"context:"** + i);  
 }  
 **return "Hello,"** + name + **"消息已发送"**;  
 }  
}

* 消息接收

消费者A

@Slf4j  
@Component  
@RabbitListener(queuesToDeclare =@Queue(value = **"work"**))  
**public class** ReceivingWorkA {  
 @RabbitHandler  
 **public void** process(String message) **throws** InterruptedException {  
 Thread.*sleep*(10);{  
  
 }  
 ***log***.error(**"ReceivingA message : {}"**, message);  
 }  
}

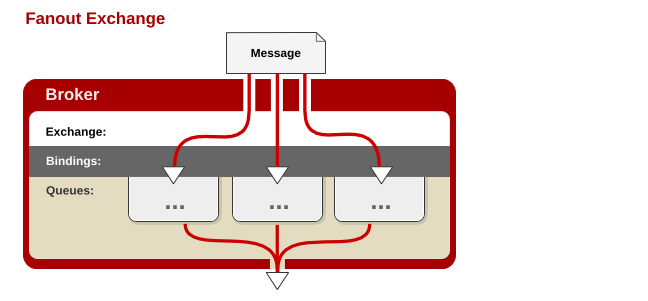
消费者B

@Slf4j  
@Component  
@RabbitListener(queuesToDeclare =@Queue(value = **"work"**))  
**public class** ReceivingWorkB {  
 @RabbitHandler  
 **public void** process(String message) **throws** InterruptedException {  
 Thread.*sleep*(1000);  
 ***log***.info(**"ReceivingB message : {}"**, message);  
 }  
}

设置消费者属性文件application.yml,在文件中增加消费缓存沲，这样在消费者A与消费者B会根据缓存沲数拉取信息。所以消费都A消费了25条信息，而A却消费了5条信息。

**spring**:  
 **rabbitmq**:  
 **listener**:  
 **simple**:  
 *#消费者每次从队列获取的消息数量。相当于消费者的缓存沲数据* **prefetch**: 5

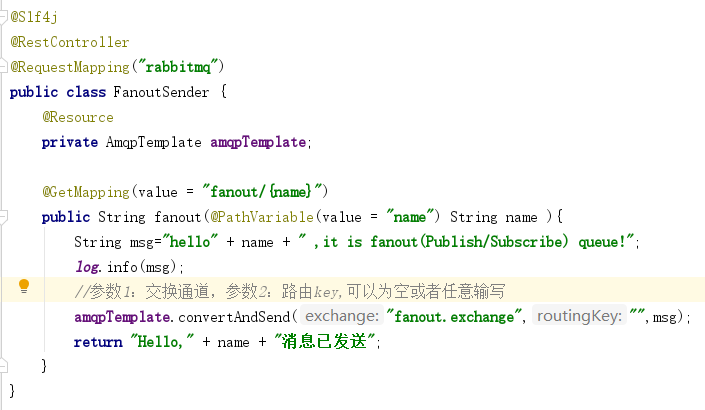
## [Publish/Subscribe](https://www.rabbitmq.com/tutorials/tutorial-three-python.html)



任何发送到Fanout Exchange 的消息都会被转发到与该Exchange绑定(Binding)的所有Queue上。

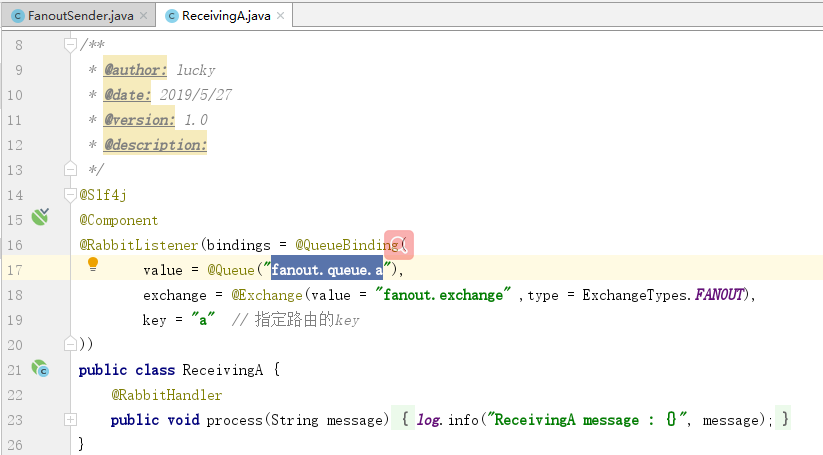
1. 可以理解为路由表的模式
2. 这种模式不需要 RouteKey
3. 这种模式需要提前将Exchange与Queue进行绑定，一个Exchange可以绑定多个Queue，一个Queue可以同多个Exchange进行绑定。
4. 如果接受到消息的Exchange没有与任何Queue绑定，则消息会被抛弃。

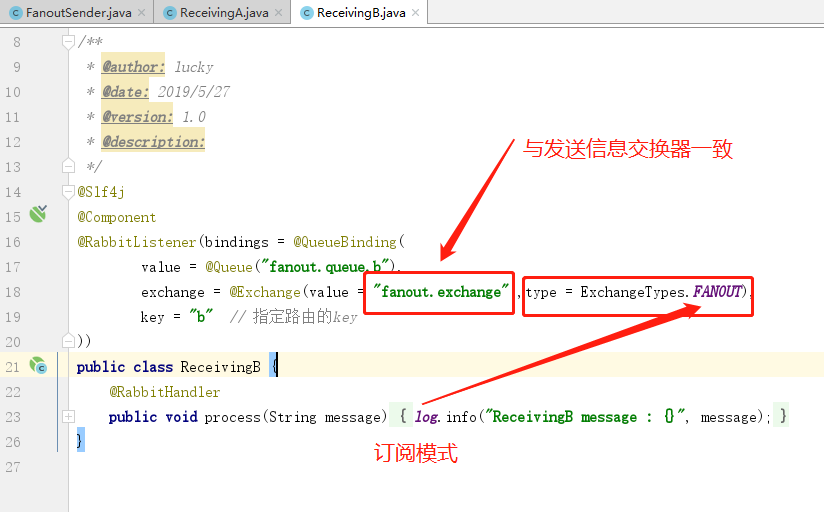
* 消息发送



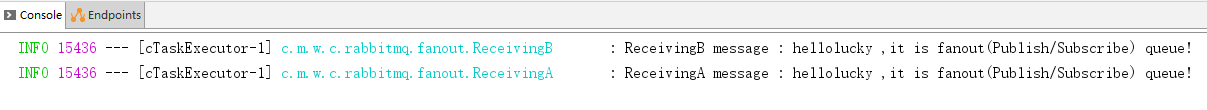
@Slf4j  
@RestController  
@RequestMapping(**"rabbitmq"**)  
**public class** FanoutSender {  
 @Resource  
 **private** AmqpTemplate **amqpTemplate**;  
 @GetMapping(value = **"fanout/{name}"**)  
 **public** String fanout(@PathVariable(value = **"name"**) String name ){  
 String msg=**"hello"** + name + **" ,it is fanout(Publish/Subscribe) queue!"**;  
 ***log***.info(msg);   
 *//参数1：交换通道，*参数*2：路由key,可以为空或者任意输写*  
 **amqpTemplate**.convertAndSend(**"fanout.exchange"**,**""**,msg);  
 **return "Hello,"** + name + **"消息已发送"**;  
 }  
}

* 消息接收

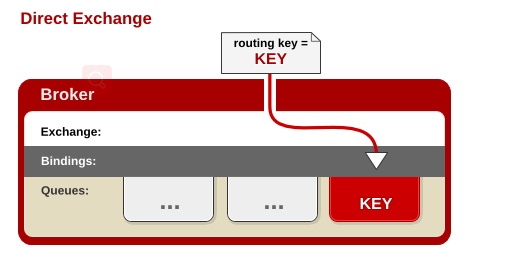




@Slf4j  
@Component  
@RabbitListener(bindings = @QueueBinding(  
 value = @Queue(**"fanout.queue.a"**),  
 exchange = @Exchange(value = **"fanout.exchange"** ,type = ExchangeTypes.***FANOUT***),  
 key = **"a"** *// 指定路由的key*))  
**public class** ReceivingA {  
 @RabbitHandler  
 **public void** process(String message) {  
 ***log***.info(**"ReceivingA message : {}"**, message);  
 }  
}



## [Routing](https://www.rabbitmq.com/tutorials/tutorial-four-python.html)



Direct Exchange 是RabbitMQ默认的交换机模式，也是最简单的模式，根据key全文匹配去寻找队列。

任何发送到Direct Exchange的消息都会被转发到RouteKey中指定的Queue。

1.一般情况可以使用rabbitMQ自带的Exchange：""(该Exchange的名字为空字符串，下文称其为default Exchange)。

2.这种模式下不需要将Exchange进行任何绑定(binding)操作

3.消息传递时需要一个RouteKey，可以简单的理解为要发送到的队列名字。

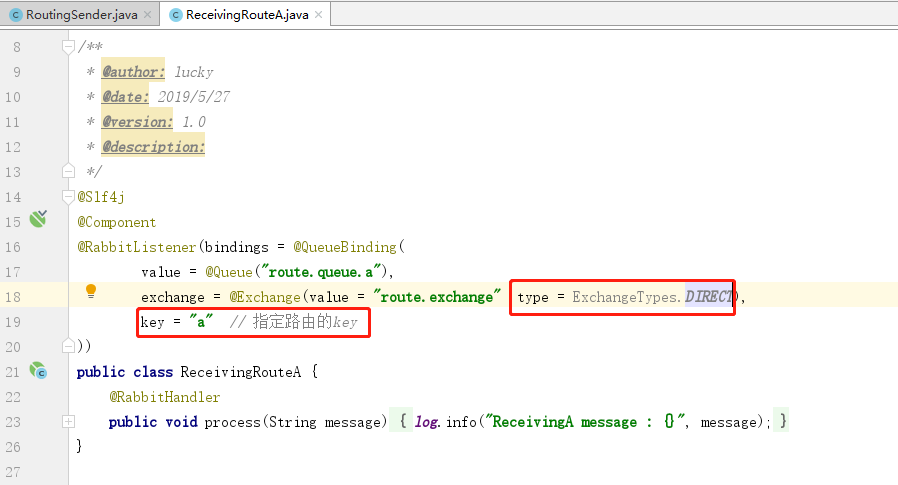
4.如果vhost中不存在RouteKey中指定的队列名，则该消息会被抛弃。  
配置队列

* 消息发送



@Slf4j  
@RestController  
@RequestMapping(**"rabbitmq"**)  
**public class** RoutingSender {  
 @Resource  
 **private** AmqpTemplate **amqpTemplate**;  
 @GetMapping(value = **"route/{name}"**)  
 **public** String fanout(@PathVariable(value = **"name"**) String name ){  
 String msg=**"hello,"** + name + **" ,it is route queue!"**;  
 ***log***.info(msg);  
 **amqpTemplate**.convertAndSend(**"route.exchange"**,name,msg);  
 **return "Hello,"** + name + **"消息已发送"**;  
 }  
}

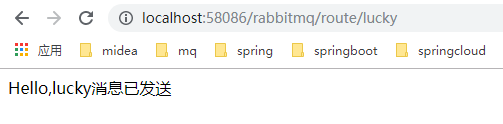
* 消息接收



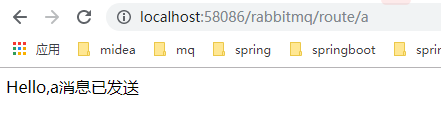
@Slf4j  
@Component  
@RabbitListener(bindings = @QueueBinding(  
 value = @Queue(**"route.queue.a"**),  
 exchange = @Exchange(value = **"route.exchange"** ,type = ExchangeTypes.**DIRECT**),  
 key = **"a"** *// 指定路由的key*))  
**public class** ReceivingRouteA {  
 @RabbitHandler  
 **public void** process(String message) {  
 ***log***.info(**"ReceivingA message : {}"**, message);  
 }  
}

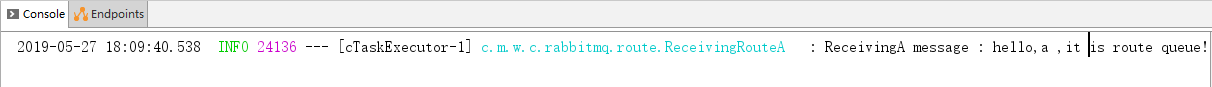


@Slf4j  
@Component  
@RabbitListener(bindings = @QueueBinding(  
 value = @Queue(**"route.queue.b"**),  
 exchange = @Exchange(value = **"route.exchange"** ,type = ExchangeTypes.***DIRECT***),  
 key = **"b"** *// 指定路由的key*))  
**public class** ReceivingRouteB {  
 @RabbitHandler  
 **public void** process(String message) {  
 ***log***.info(**"ReceivingB message : {}"**, message);  
 }  
}



生产方发送信息name为lucky时，routingKey=name,所以消费方接收不到信息。

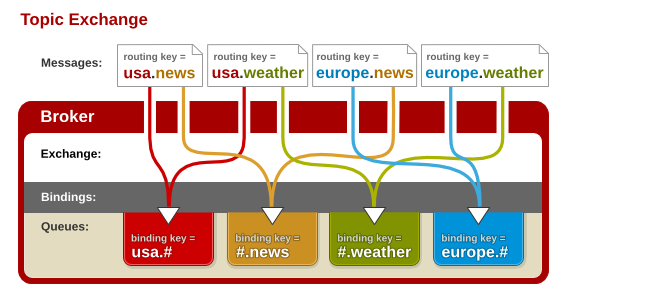




生产方发送信息name为a时，routingKey=a,所以消费方ReceivingRouteA接收到信息。

生产方发送信息name为b时，routingKey=b,所以消费方ReceivingRouteB接收到信息。

## [Topics](https://www.rabbitmq.com/tutorials/tutorial-five-python.html)



任何发送到Topic Exchange的消息都会被转发到所有关心RouteKey中指定话题的Queue上

1. 这种模式较为复杂，简单来说，就是每个队列都有其关心的主题，所有的消息都带有一个标题``(RouteKey)，Exchange会将消息转发到所有关注主题能与RouteKey模糊匹配的队列。
2. 这种模式需要RouteKey，也许要提前绑定Exchange与Queue。
3. 在进行绑定时，要提供一个该队列关心的主题，如#.log.#表示该队列关心所有涉及log的消息(一个RouteKey为MQ.log.error的消息会被转发到该队列)。
4. #表示0个或若干个关键字，表示一个关键字。如topic.能与topic.warn匹配，无法与topic.warn.timeout匹配；但是topic.#能与上述两者匹配。
5. \*表示0个或1个关键字，表示一个关键字。如topic.\*能与topic.warn匹配，无法与topic.warn.timeout匹配.
6. 同样，如果Exchange没有发现能够与RouteKey匹配的Queue，则会抛弃此消息。

* 生产者

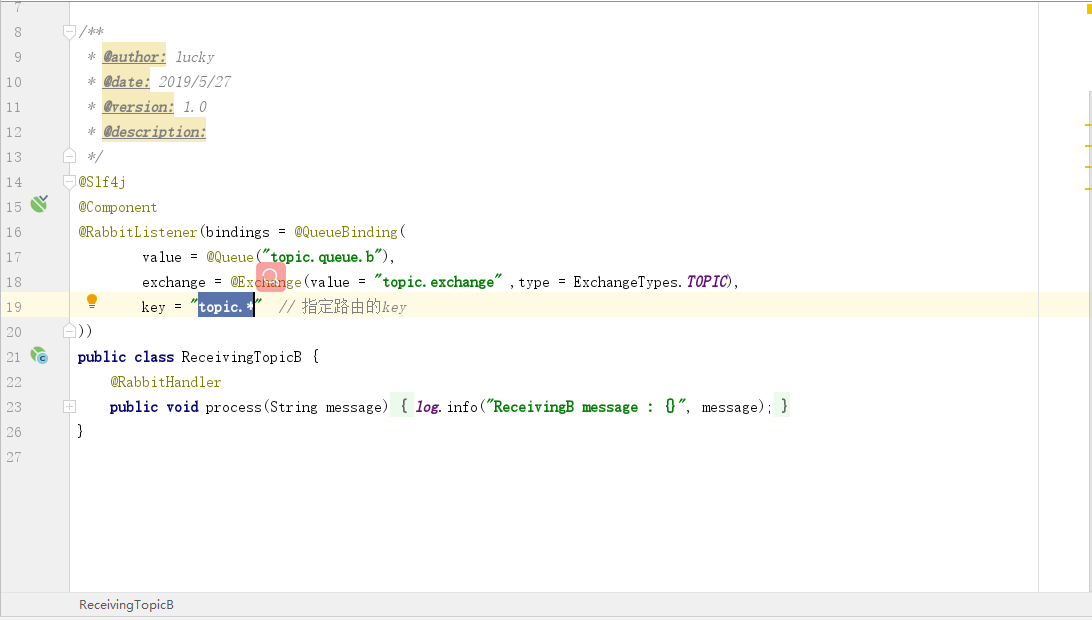


@Slf4j  
@RestController  
@RequestMapping(**"rabbitmq"**)  
**public class** TopicSender {  
 @Resource  
 **private** AmqpTemplate **amqpTemplate**;  
  
 @GetMapping(value = **"topic/{name}"**)  
 **public** String fanout(@PathVariable(value = **"name"**) String name ){  
 String msg=**"hello,"** + name + **" ,it is topic queue!"**;  
 ***log***.info(msg);  
 **amqpTemplate**.convertAndSend(**"topic.exchange"**,name,msg);  
 **return "Hello,"** + name + **"消息已发送"**;  
 }  
}

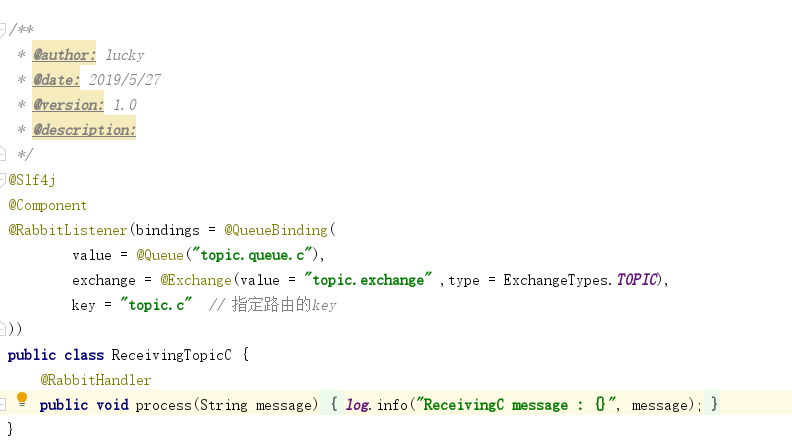
* 消费者



@Slf4j  
@Component  
@RabbitListener(bindings = @QueueBinding(  
 value = @Queue(**"topic.queue.a"**),  
 exchange = @Exchange(value = **"topic.exchange"** ,type = ExchangeTypes.***TOPIC***),  
 key = **"topic.#"** *// 指定路由的key*))  
**public class** ReceivingTopicA {  
 @RabbitHandler  
 **public void** process(String message) {  
 ***log***.info(**"ReceivingA message : {}"**, message);  
 }  
}



@Slf4j  
@Component  
@RabbitListener(bindings = @QueueBinding(  
 value = @Queue(**"topic.queue.b"**),  
 exchange = @Exchange(value = **"topic.exchange"** ,type = ExchangeTypes.***TOPIC***),  
 key = **"topic.\*"** *// 指定路由的key*))  
**public class** ReceivingTopicB {  
 @RabbitHandler  
 **public void** process(String message) {  
 ***log***.info(**"ReceivingB message : {}"**, message);  
 }  
}



@Slf4j  
@Component  
@RabbitListener(bindings = @QueueBinding(  
 value = @Queue(**"topic.queue.c"**),  
 exchange = @Exchange(value = **"topic.exchange"** ,type = ExchangeTypes.***TOPIC***),  
 key = **"topic.c"** *// 指定路由的key*))  
**public class** ReceivingTopicC {  
 @RabbitHandler  
 **public void** process(String message) {  
 ***log***.info(**"ReceivingC message : {}"**, message);  
 }  
}

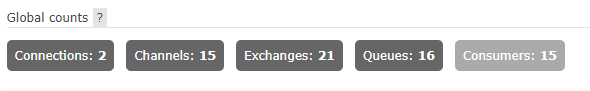
## [RPC](https://www.rabbitmq.com/tutorials/tutorial-six-python.html)（省略）

## 应用与RabbitMQ关系

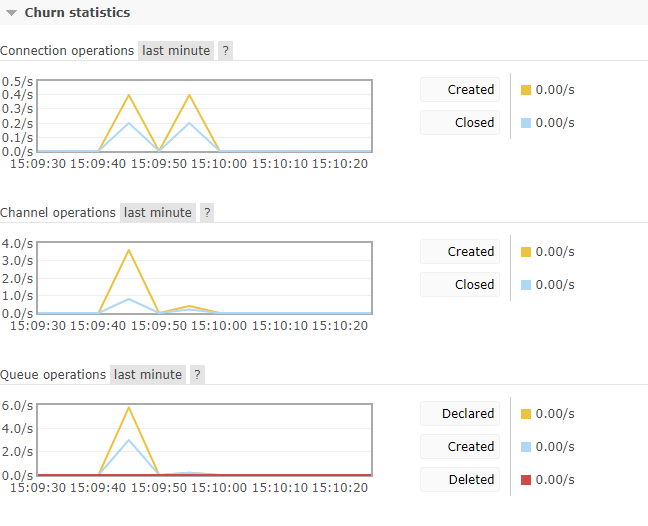
**生产者与消费者都启动之后的变化**

## RabbitMQ 概观

1. 下面有“Global counts”选项的连接、通道、交换器、队列信息发生了变化

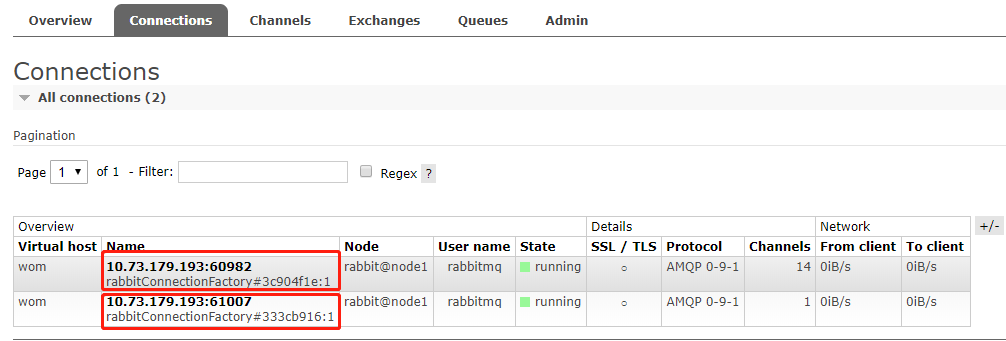


1. “Churn statistics”这里显示连接，交换器，队列最后操作时间表

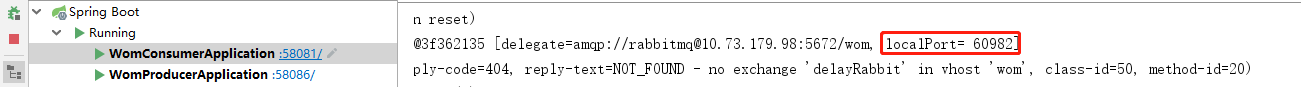


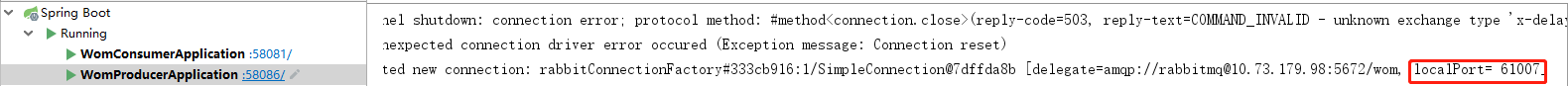
## RabbitMQ Connetions连接信息

1. “Connections”(连接)选项卡下面的信息：与Global counts的连接数为2一致。



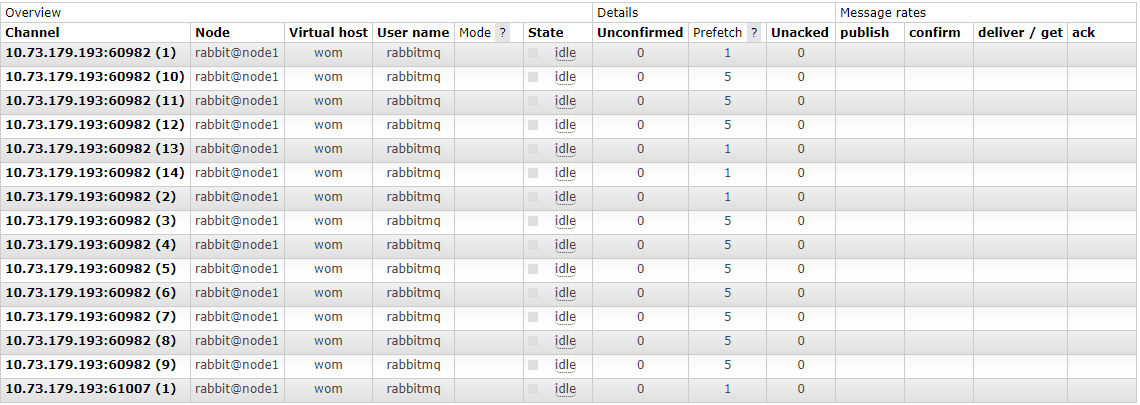
1. Connections name 的接口与生产者与消费者的应用接口与连接的应用一致。



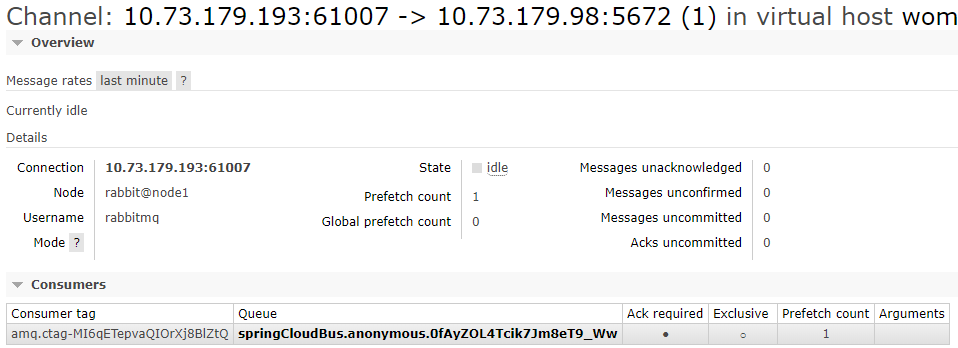


## RabbitMQ Channels通道信息

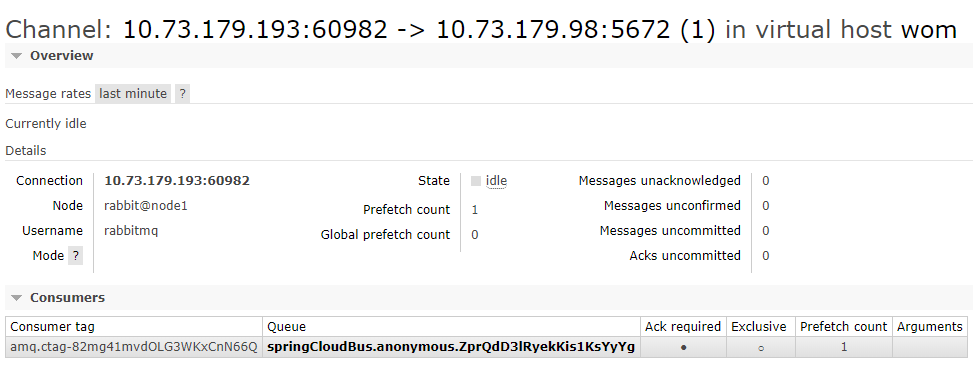
1. “Channels”(通道)选项卡下面的信息：与Global counts的通道数为15一致。



1. 生产者的通道只有一条，点击“10.73.179.98:61007 (1)”进去查看，列队属于spring-cloud-starter-bus-amqp，在生产者启动时自动创建的。

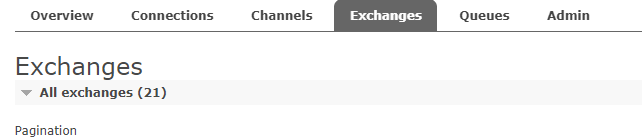


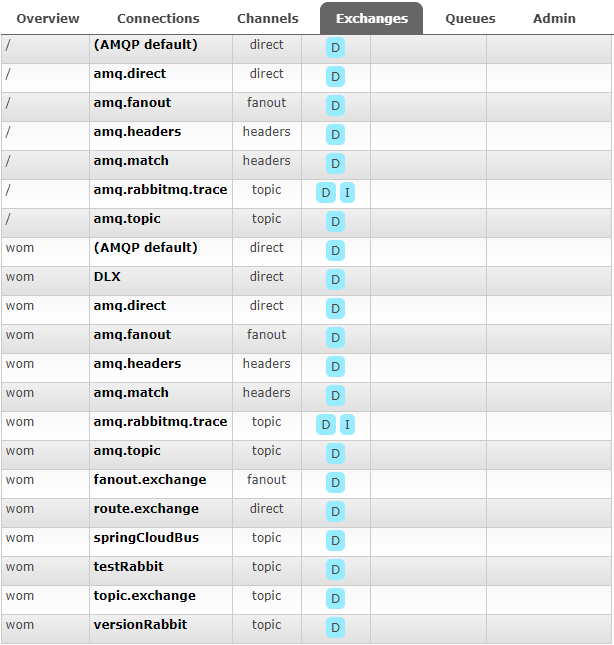
1. 同样，消费者通道也有一条属于cloud-starter-bus-amqp自动创建，点击“10.73.179.98:60982 (1)”结果如下图。



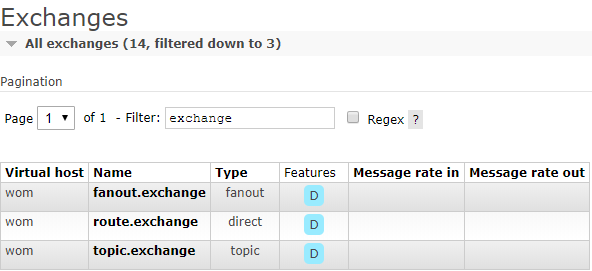
## RabbitMQ Exchanges交换器信息

1. “Exchanges”(交换器)选项卡下面的信息：与Global counts的交换器数为21一致。



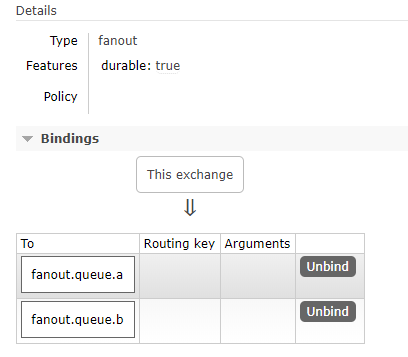


在生产者与消费者配置文件时，设置virtual-host: wom，所以生产者与消费都的交换器者存放在wom中：分别是[Publish/Subscribe](https://www.rabbitmq.com/tutorials/tutorial-three-python.html)的fanout.exchange、Routing的route.exchange与Topics的topic.exchange。根据条件exchange过虑结果

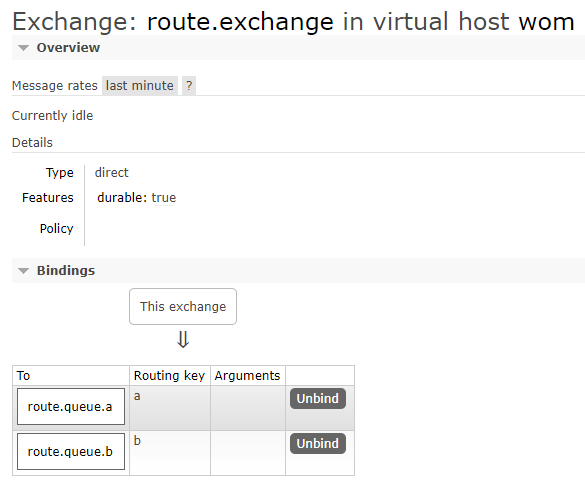


1. 点击列“name” 的交换器查看明细，可以查看详细信息与绑定队列

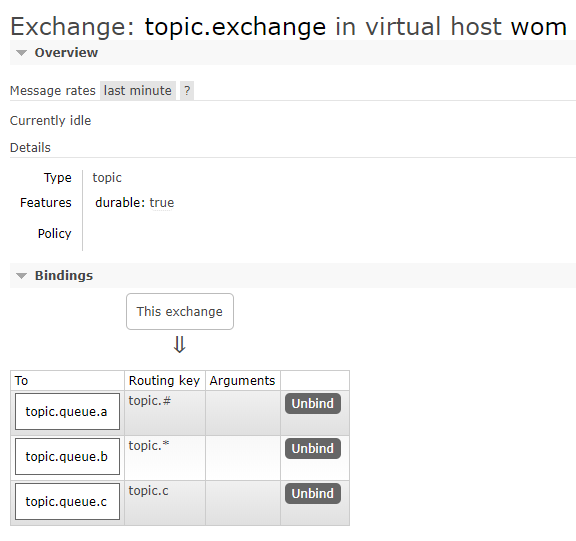
fanout.exchange明细



route.exchange 明细

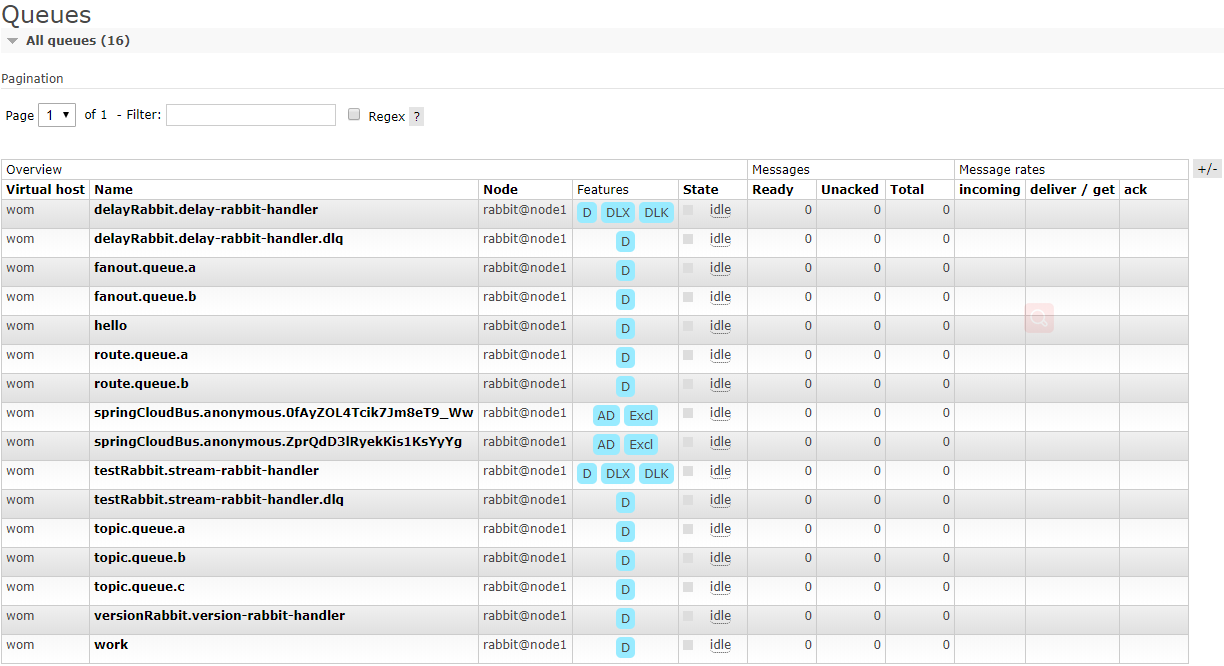


topic.exchange明细

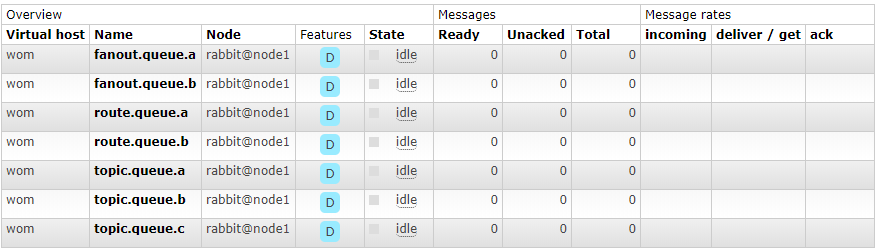


## RabbitMQ Queue队列信息

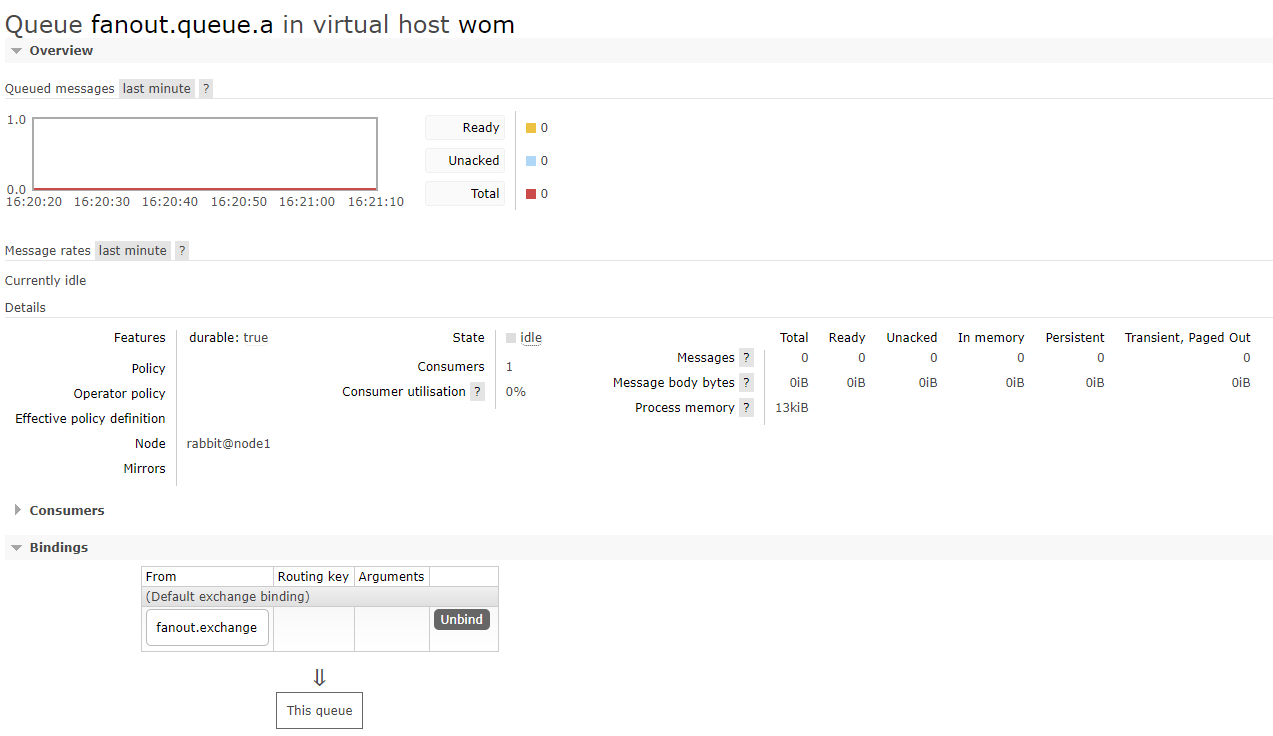
1. “Queue”(队列)选项卡下面的信息：与Global counts的交换器数为16一致。

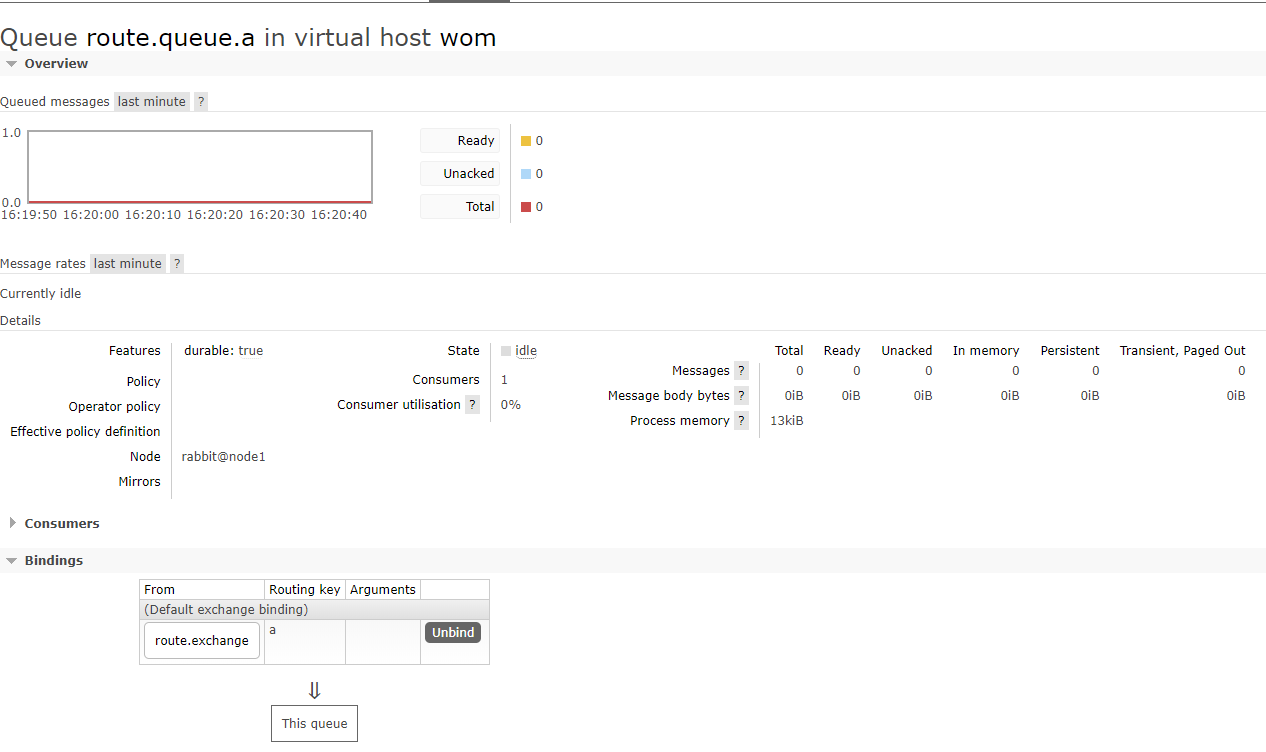


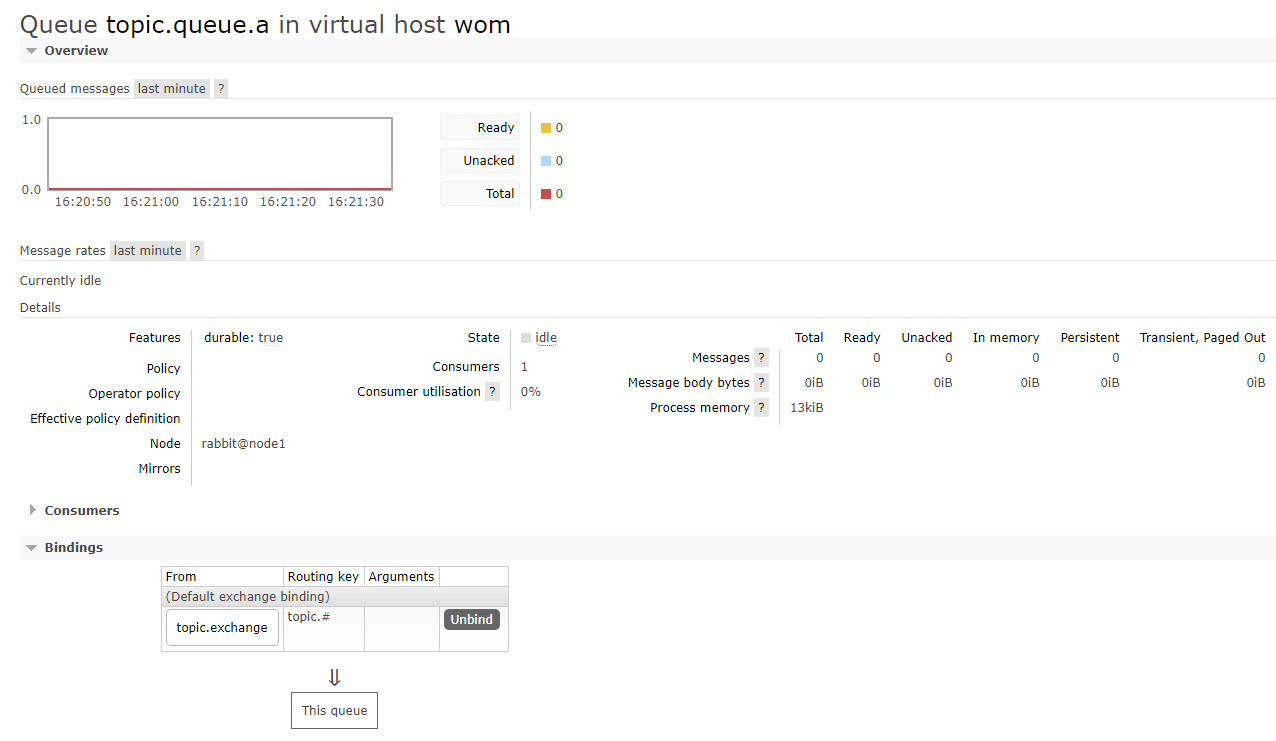
1. 分别是[Publish/Subscribe](https://www.rabbitmq.com/tutorials/tutorial-three-python.html)的fanout.exchange、Routing的route.exchange与Topics的topic.exchange交换器的队列

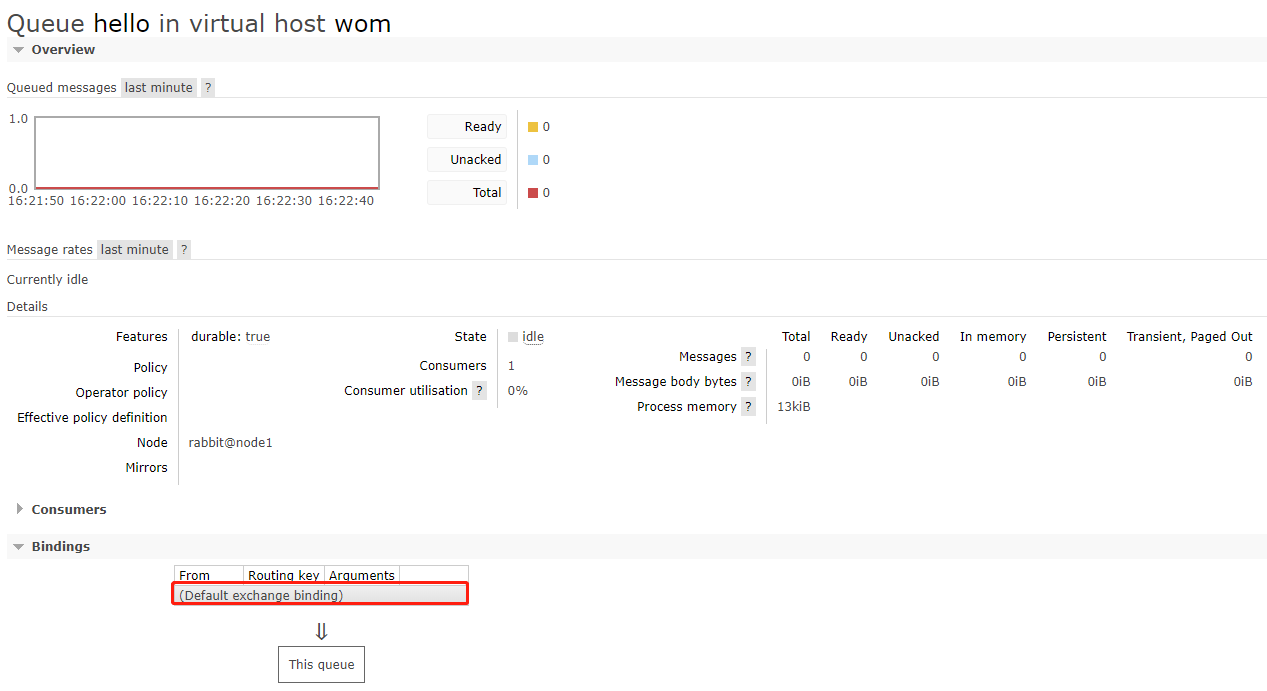


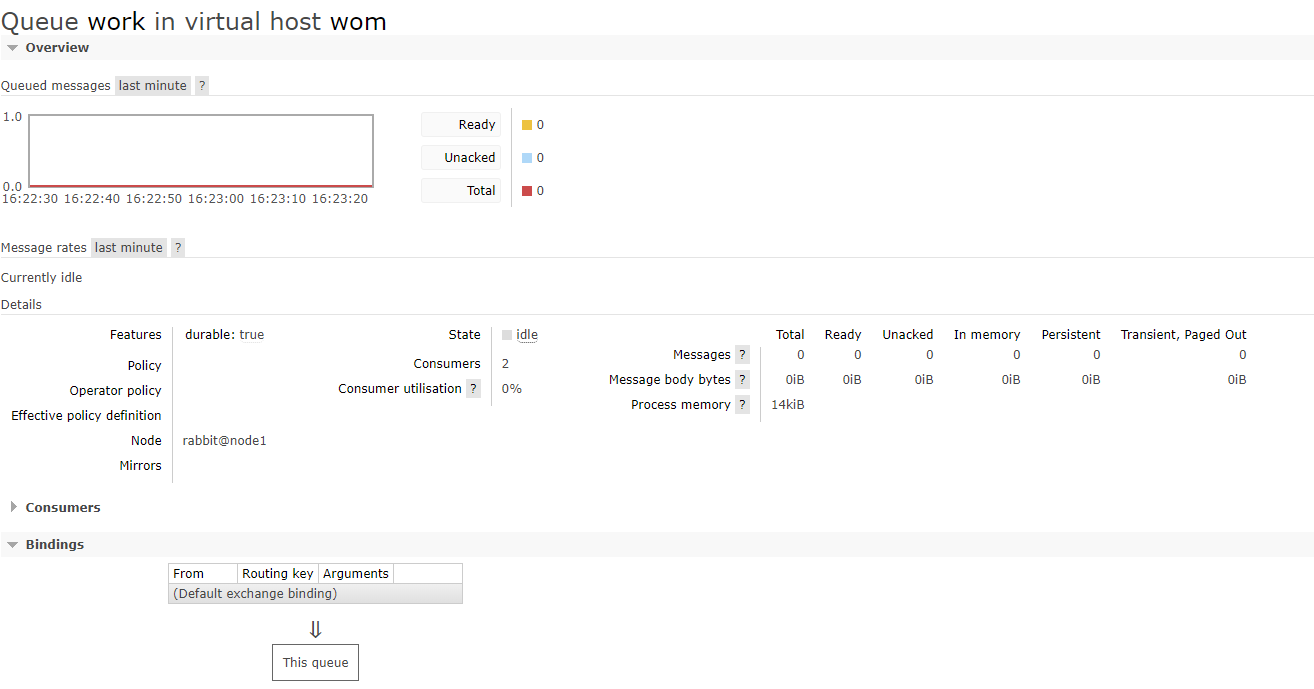
1. hello work实例 的hello 与 work 实例的work队列
2. 分别查看fanout.exchange、route.exchange、topic.exchange、hello、work











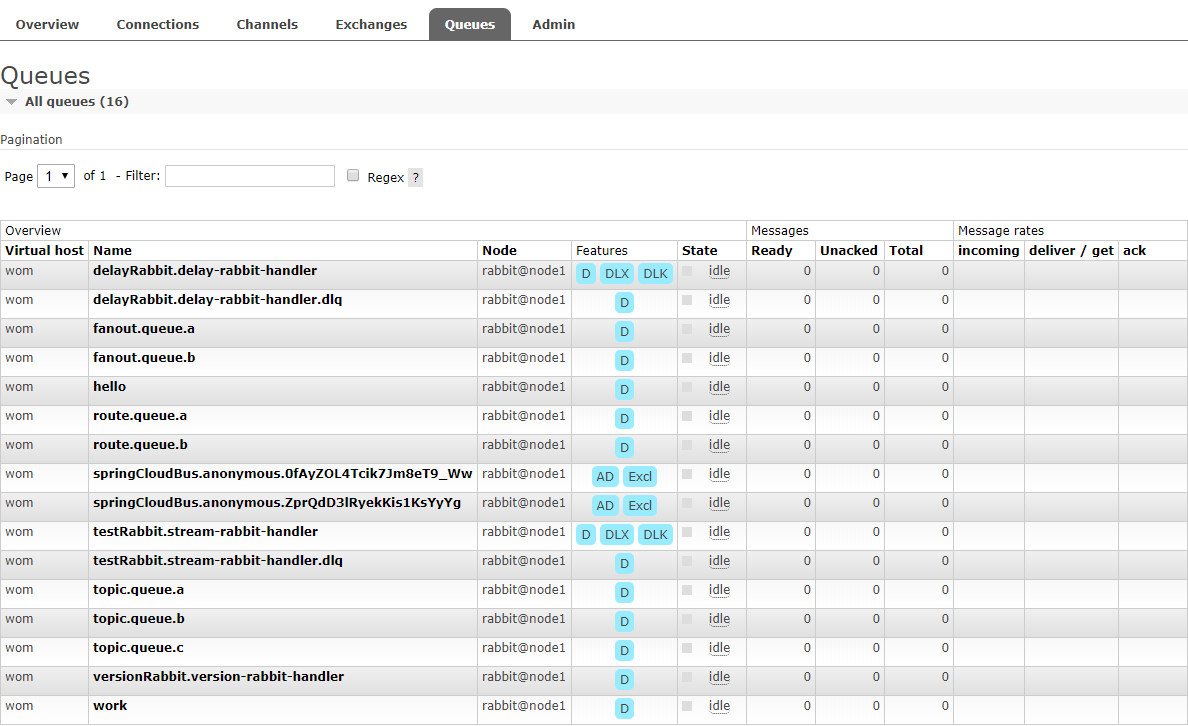
## RabbitMQ队列

## 持久化

 　RabbitMQ的队列和消息存放于内存中，在出现任何异常情况，导致RabbitMQ异常中止，内存中的数据将会丢失。若要在RabbitMQ从异常恢复后仍然保存有异常中止前的消息，就需将消息持久化，即将消息同时保存于磁盘中。RabbitMQ可将消息保存到Erlang自带的Mnesia数据库中，当RabbitMQ重启之后会读取该数据库。

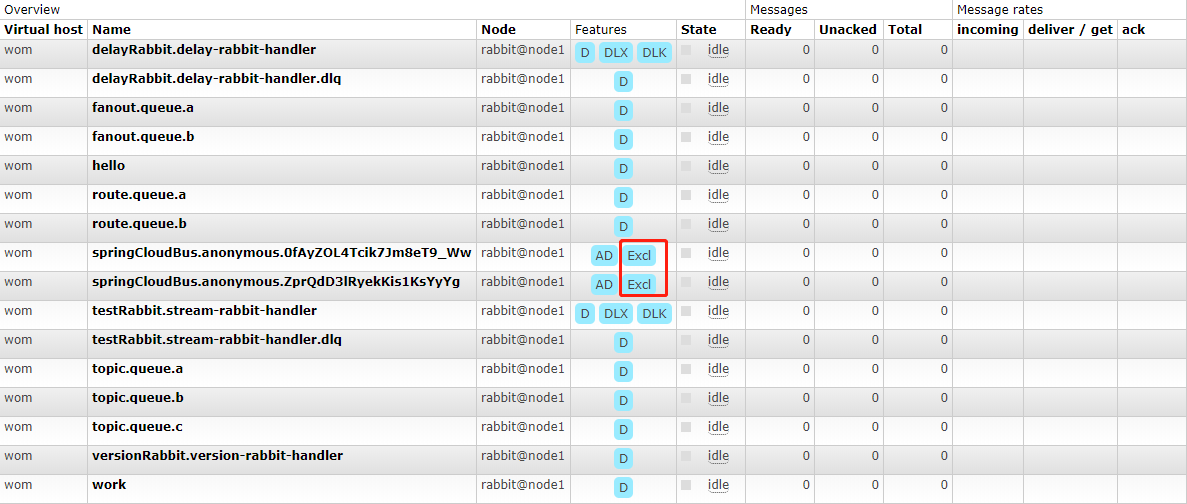
 　声明队列的方法中，“durable”参数即指定队列是否是持久化的。使消息持久化，需要队列和消息都是持久化的，并且通常交换机也应该是持久化的。RabbitMQ的默认交换机“(AMQP default)”是持久化的，对于与其绑定的队列，将队列声明为持久化的队列，并发送持久化的消息，即可将消息持久化。

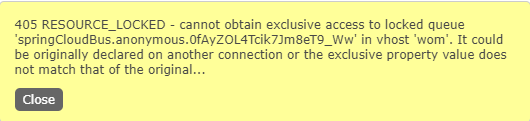
队列列表中，特性（Features）列的“D”即表示该队列是持久化的（Durable）。可以验证，若队列不是持久化的，或发送的消息未设置持久化的属性，在RabbitMQ重启后，消息都会丢失。



## 排他

 　对于排他队列，只有创建它的连接有权访问，连接断开后，排他队列将自动删除。这种队列适用于一个队列仅对应一个客户端收发消息的场景。在声明队列时，将exclusive参数设置为true即可声明一个排他队列。  
 　可通过以下程序验证排他队列的性质。在程序运行至等待控制台输入时，进入Web管理界面，可以看到，队列列表中出现了被创建的排他队列，特性（Features）列的“Excl”即表明该队列是排他的。关键的提示信息为“cannot obtain exclusive access to locked queue”，无法获取加锁队列的访问权限，即该队列对于其他连接是无权访问的。

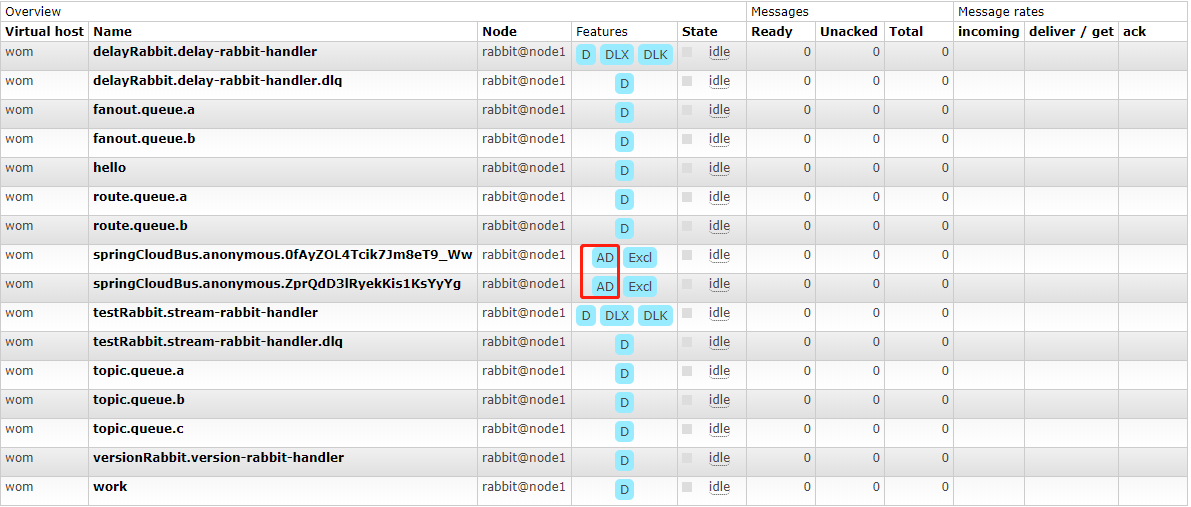




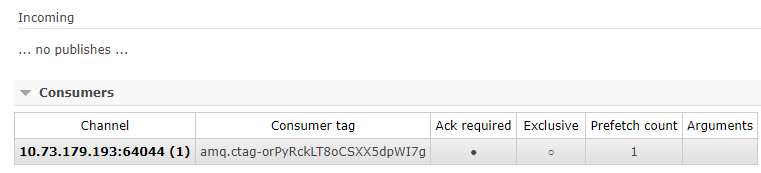
## 自动删除

若队列的autoDelete（自动删除）属性开启，当队列的最后一个消费者断开时，该队列会被自动删除。

声明一个队列，autoDelete参数设置为true，进入Web管理界面，可以看到，队列列表中出现了被创建的自动删除队列，Features列的“AD”即表明该队列是自动删除的。



点击队列名进入队列的管理界面，在“Consumers”下拉框中能看到连接该队列的所有消费者



启动一个消费者程序连接该队列，连接后断开，可以看到，该队列在消费者断开后被删除了。

## 其他参数

在声明队列方法的入参中，存在一个称为“arguments”的参数，该参数以键值对的形式，可为队列设置多种属性。以“arguments”入参声明的队列，同样须以相同的属性和值为参数进行连接。可为队列设置以下参数。

x-message-ttl：队列中消息的生存周期，单位毫秒，设置该属性后，队列对每条消息自接收开始起计时，超过设置的时间值后，自动删除该消息；

x-expires：队列闲置时间，单位毫秒，若队列在指定的时间内没有被访问，则会自动删除；

x-max-length：队列的最大长度，即队列存放消息的最大数目，若接收消息后队列的消息数超过指定的队列最大长度，则将最早的消息移出队列；

x-max-length-bytes：队列最大占用空间，单位字节，若接收消息后队列的占用空间超过指定的队列最大占用空间，则将最早的消息移出队列；

x-dead-letter-exchange、x-dead-letter-routing-key：设置这两个参数后，当队列超过指定长度或占用空间大小时，对于移出的较早的消息，将其以指定的路由键发送至指定的交换机，而不直接删除，x-dead-letter-exchange指定交换机名称，x-dead-letter-routing-key指定路由键；

x-max-priority：优先级队列，指定队列的最大优先级，可在发布消息的时候指定消息的优先级，优先级更高（数值更大的）的消息先被消费，使用见后续消息章节；

x-queue-mode：队列模式，分为默认（default）和延迟（lazy）两种，当该值设置为“lazy”时，消息将先保存到磁盘上、不放在内存中，当消费者开始消费的时候才加载到内存中。

设置参数的示例如下。

Map<String, Object> arguments = new HashMap<String, Object>();

arguments.put("x-message-ttl", 10000); // 消息生存时间，10s

arguments.put("x-expires", 20000); // 队列闲置时间，20s

arguments.put("x-max-length", 100); // 队列最大长度，100条消息

arguments.put("x-max-length-bytes", 1024); // 队列最大占用空间，1024字节

// 移出的消息发送至“amq.direct”交换机，路由键为“dead”

arguments.put("x-dead-letter-exchange", "amq.direct");

arguments.put("x-dead-letter-routing-key", "dead");

arguments.put("x-max-priority", 10); // 最大优先级为10

arguments.put("x-queue-mode", "lazy"); // 延迟模式

channel.queueDeclare(QUEUE\_NAME, true, false, false, arguments);

