# RabbitMQ

目录

[RabbitMQ 1](#_Toc21526)

[一、 认识RabbitMQ 1](#_Toc2358)

[1、 RabbitMQ简介 1](#_Toc31208)

[2、 安装ErLang开发环境 3](#_Toc1552)

[3、 安装并配置RabbitMQ 4](#_Toc23439)

[二、使用Java操作RabbitMQ 5](#_Toc28443)

[1、建立消息生产者 5](#_Toc12941)

[2、 定义消息消费者 9](#_Toc32615)

[3、 工作队列 11](#_Toc24316)

[4、 消息持久化 11](#_Toc2757)

[5、虚拟主机 11](#_Toc4351)

[二、 rabbit-0208\_虚拟主机 12](#_Toc26109)

[1、 广播模式（fanout） 12](#_Toc25261)

[2、 直连模式（direct） 14](#_Toc8595)

[3、 主题模式（topic） 16](#_Toc3634)

[三、 Spring整合RabbitMQ组件 18](#_Toc13259)

[1、定义消费者程序 19](#_Toc2636)

[2、 定义消息生产者 20](#_Toc21462)

[五、 RabbitMQ集群 22](#_Toc26437)

[1、 搭建RabbitMQ基础集群 22](#_Toc5955)

[2、 镜像队列 23](#_Toc22804)

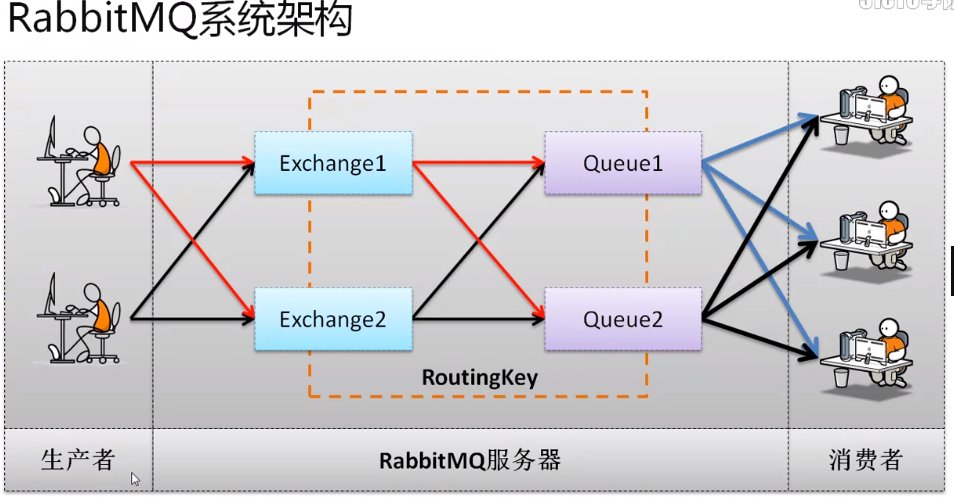
## 认识RabbitMQ

### RabbitMQ简介

从消息组件来讲主要划分为两类：

1. JMS组件：ActiveMQ(速度慢)
2. AMQP组件：性能最高，有两个主要的开源项目：
   1. RabbitMQ：使用最为广泛，速度也很快；
   2. kafka：是大数据时代产生的。

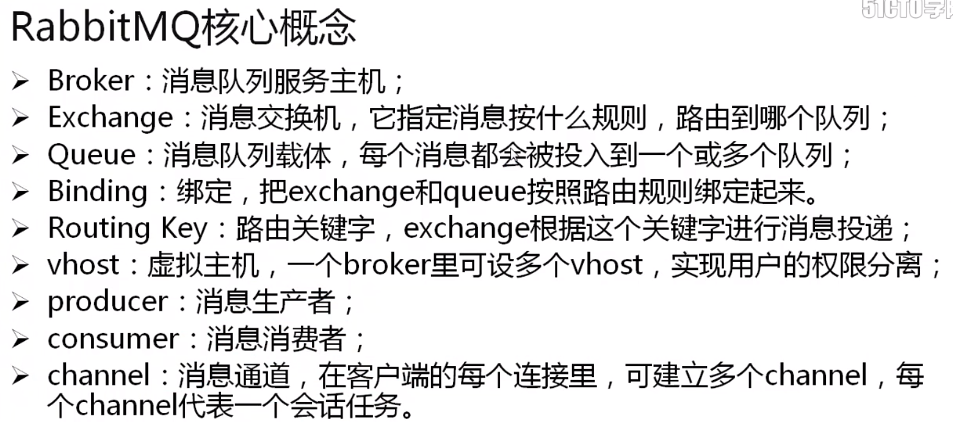




在所有的消息系统中，其基本组成一定是生产者、消费者、消息组件，其中消息组件可以起到的作用：数据缓冲队列。

但是在RabbitMQ里面这个消息组件的组成：

1. Exchange：交换空间，管理所有的队列；
2. Queue：数据队列；
3. Routingkey:如果所有的队列的routingkey都一样，那么就属于广播消息，如果不一样，则表示点对点；



在rabbitmq里面提供有一个虚拟主机的概念，所谓虚拟主机可以理解为不同用户空间，也就是说各个空间可以有自己的队列信息，有自己的操作用户。

### 安装ErLang开发环境

RabbitMQ依靠的开发语言为Erlang，所以如果想要使用RabbitMQ服务，首先一定要进行此开发环境的配置。一定要保证主机上有jdk环境。

查看系统中是否安装erlang，执行rmp -qa |grep erlang

1. 将erlang的开发包上传到系统之中，而后进行解压到“/usr/local/src”:

tar zxvf /upload/otp\_src\_19.2.tar.gz -C /usr/local/src/

1. 进入到erlang的源代码目录中对其进行编译处理：
2. 编译erlang源代码，则需要一个类库的支持：libncurses5-dev;

yum -y install libncurses5-dev

1. 建立一个erlang编译后的文件目录：

mkdir -p /usr/local/erlang

1. 进入源代码所在目录编译配置：

./configure --prefix=/usr/local/erlang/

1. 编译与安装：make && make install
2. 将erlang开发包配置到环境属性中(/etc/profile)。
3. 追加环境配置：

export ERLANG\_HOME=/usr/local/erlang

export PATH=$JAVA\_HOME/bin:$ERLANG\_HOME/bin:$PATH

1. 使环境属性立即生效：source /ect/profile
2. 测试erlang是否可用，直接输入：erl

执行io:format(“hello world”)

推出界面：halt().

安装过程中遇到问题：

configure: error: No curses library functions found

configure: error: /bin/sh '/usr/local/src/otp\_src\_19.2/erts/configure' failed for erts

解决：yum -y install ncurses-devel

### 安装并配置RabbitMQ

注意项：rabbitmq官方提供的下载包为“tar.xz”文件。

1. 将tar.xz解压缩，xz -d rabbitmq-server-generic-unix-3.6.6.tar.xz ,生成tar文件。
2. 将tar解压缩到指定的位置”usr/local/”：

tar xvf rabbitmq-server-generic-unix-3.6.6.tar -C /usr/local/

1. 更名处理:

mv rabbitmq\_server-3.6.6/ rabbitmq

1. 启动rabbitmq的服务：

/usr/local/rabbitmq/sbin/rabbitmq-server start

1. rabbitmq自带有系统监控也就是说它存在有一个WEB的客户端，但是要进行自己的操作，建议做一个新的管理员账户(yangmi/ym)。查看帮助命令：

/usr/local/rabbitmq/sbin/rabbitmqctl --help

1. 追加用户：

/usr/local/rabbitmq/sbin/rabbitmqctl add\_user yangmi ym

1. 此时此账户只是一个普通用户，随后需要将其假如到管理员中；

/usr/local/rabbitmq/sbin/rabbitmqctl set\_user\_tags yangmi administrator

查看用户列表：

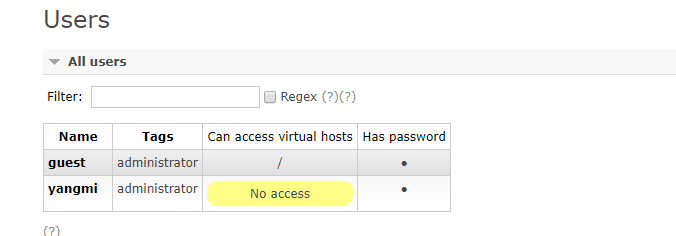
/usr/local/rabbitmq/sbin/rabbitmqctl list\_users

1. 默认情况下rabbitmq的服务是不带webserver启动的，需要自己配置启动：

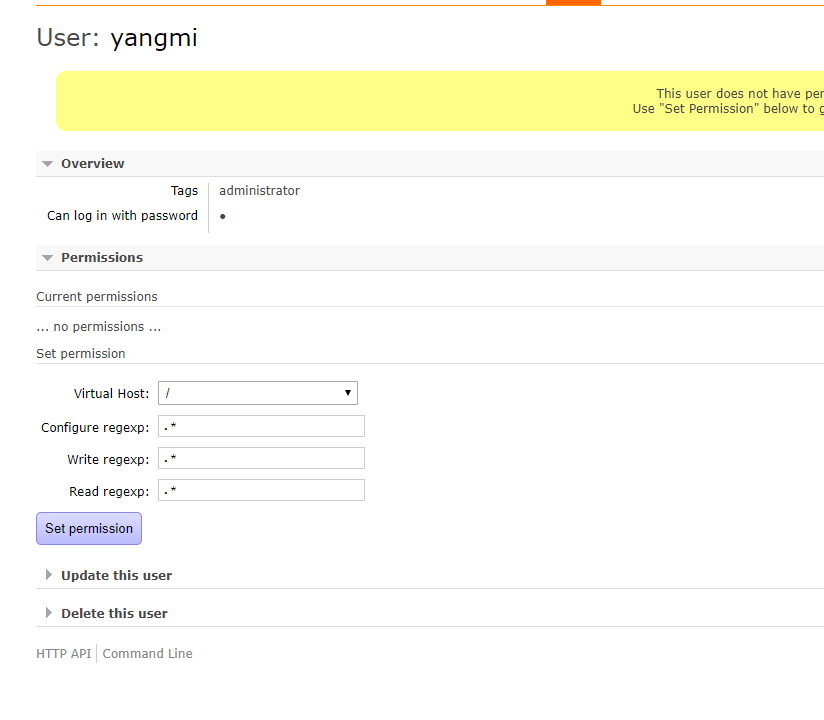
/usr/local/rabbitmq/sbin/rabbitmq-plugins enable rabbitmq\_management

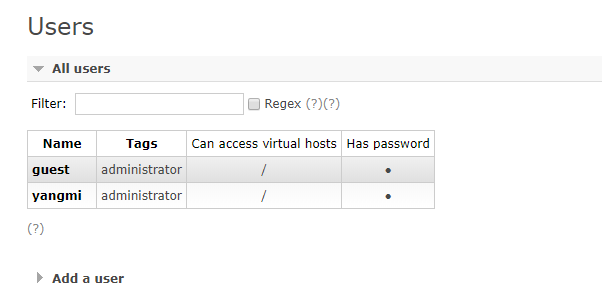
1. 可以在浏览器中输入地址访问：<http://192.168.1.217:15672/>
2. 配置虚拟主机操作权限：

修改前：



修改后：





## 二、使用Java操作RabbitMQ

### 1、建立消息生产者

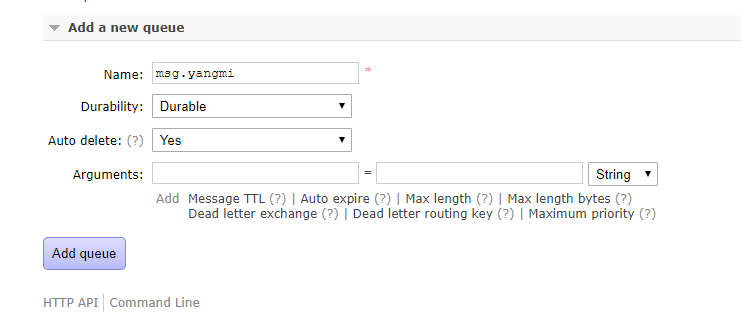
1、新建一个父（模块）项目：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>cn.ym.com</**groupId**>  
 <**artifactId**>myrabbit</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 <**packaging**>pom</**packaging**>  
 <**name**>myrabbit</**name**>  
  
 <**properties**>  
 <**jdk.version**>1.8</**jdk.version**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 </**properties**>  
 <**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.rabbitmq</**groupId**>  
 <**artifactId**>amqp-client</**artifactId**>  
 <**version**>4.1.1</**version**>  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
 <**build**>  
 <**finalName**>myrabbit</**finalName**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**configuration**>  
 <**source**>${jdk.version}</**source**>*<!-- 源代码使用的开发版本 -->* <**target**>${jdk.version}</**target**>*<!-- 需要生成的目标class文件的编译版本 -->* <**encoding**>${project.build.sourceEncoding}</**encoding**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
 <**modules**>  
 <**module**>myrabbitbase</**module**>  
 </**modules**>  
</**project**>

2、建立一个myrabbit-base的子模块：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>myrabbit</**artifactId**>  
 <**groupId**>cn.ym.com</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>myrabbit-base</**artifactId**>  
 <**name**>myrabbit-base</**name**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>com.rabbitmq</**groupId**>  
 <**artifactId**>amqp-client</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

3、 队列访问，可以在前端控制台创建，也可以通过程序自动创建：



在进行队列信息创建的时候会有如下可用选项：

1. name:队列的名称，这个名称一定是唯一的；
2. 消息保存类型：持久化保存（druid，前提是消息还没有被消费）、瞬时状态（Transient，内存中保存）;
3. 允许自动删除（消息使用后）；
4. 定义一个消息生产者程序：

**public class** MessageProducer {  
 **private static final** String ***QUEUE\_NAME*** = **"msg.yangmi"**; *// 消息队列名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 *// 设置虚拟主机  
 //*factory.setVirtualHost(**"yangmi"**);  
 *// 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 *// 定义该通道要使用的队列名称，此时的队列已经创建过了  
 // 第一个参数：队列名称（这个队列可能存在也可能不存在）  
 // 第二个参数：是否为持久保存  
 // 第三个参数：此队列是否为专用的队列信息，可以设置为false  
 // 第四个参数：是否允许自动删除* channel.queueDeclare(***QUEUE\_NAME***, **true**, **false**, **true**, **null**);  
 **long** start = System.*currentTimeMillis*();  
 **for** (**int** x = 0; x < 10; x++) {  
 String msg = **"yangmi"** + x;  
 *// 进行消息发送* channel.basicPublish(**""**, ***QUEUE\_NAME***, MessageProperties.***PERSISTENT\_TEXT\_PLAIN***, msg.getBytes());  
 }  
 System.***out***.println(**"耗时："** + (System.*currentTimeMillis*() - start));  
 channel.close();  
 connection.close();  
 }  
}

运行程序遇到问题：

Caused by: com.rabbitmq.client.ShutdownSignalException: connection error; protocol method: #method<connection.close>(reply-code=530, reply-text=NOT\_ALLOWED - access to vhost 'yangmi' refused for user 'yangmi', class-id=10, method-id=40)

at com.rabbitmq.utility.ValueOrException.getValue(ValueOrException.java:66)

at com.rabbitmq.utility.BlockingValueOrException.uninterruptibleGetValue(BlockingValueOrException.java:36)

at com.rabbitmq.client.impl.AMQChannel$BlockingRpcContinuation.getReply(AMQChannel.java:412)

at com.rabbitmq.client.impl.AMQChannel.privateRpc(AMQChannel.java:244)

at com.rabbitmq.client.impl.AMQChannel.exnWrappingRpc(AMQChannel.java:128)

... 8 more

1：检查虚拟主机“yangmi”,权限为“/”。排除问题不在这。

2：检查防火墙，也已经关闭了

3：既然是虚拟主机不允许连接，尝试不设置，居然运行成功了，如果不设置虚拟主机，则默认连接的是“/”主机。

4：之所以连接不上名称为“杨幂”的性能主机，是因为此主机不存在，哎。

### 定义消息消费者

消费者额处理模式基本流程与生产者形式类似，因为消费者一定也要通过特定的队列进行处理操作。

**public class** MessageConsumer {  
 **private static final** String ***QUEUE\_NAME*** = **"msg.yangmi"**; *// 消息队列名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 *//factory.setVirtualHost("yangmi");  
 // 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 channel.queueDeclare(***QUEUE\_NAME***, **true**, **false**, **true**, **null**);  
 *// 在RabbitMQ里面所有的消费者的信息是通过一个回调方法完成的* Consumer consumer = **new** DefaultConsumer(channel) {  
 *// 需要覆写指定的方法实现消息消费处理* @Override  
 **public void** handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte**[] body) **throws** IOException {  
 String message = **new** String(body);  
 System.***out***.println(**"消息消费者："** + message);  
 }  
 };  
 *// 消息消费* channel.basicConsume(***QUEUE\_NAME***, consumer);  
 }  
}

在rabbitmq中还有一个比较高级的应答机制，所谓的应答机制指的是如果配置了该机制，那么当消费者成功取走消息之后，告诉服务器已经取走内容，服务器就可以把内容输出掉。这种机制的好处在于整体消息的可靠性高，但是缺点在于性能将受到影响。

**public class** MessageConsumer {  
 **private static final** String ***QUEUE\_NAME*** = **"msg.yangmi"**; *// 消息队列名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 *//factory.setVirtualHost("yangmi");  
 // 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 channel.queueDeclare(***QUEUE\_NAME***, **true**, **false**, **true**, **null**);  
 *// 在RabbitMQ里面所有的消费者的信息是通过一个回调方法完成的* Consumer consumer = **new** DefaultConsumer(channel) {  
 *// 需要覆写指定的方法实现消息消费处理* @Override  
 **public void** handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte**[] body) **throws** IOException {  
 String message = **new** String(body);  
 System.***out***.println(**"消息消费者："** + message);  
 *//通过消息标签实现应答处理* channel.basicAck(envelope.getDeliveryTag(),**false**);  
 }  
 };  
 *// 消息消费* channel.basicConsume(***QUEUE\_NAME***, consumer);  
 }  
}

### 工作队列

如果一个生产者对应多个消费者，则表示一个工作队列，而工作队列的最大特征在于：若干个消费者一起完成工作。

因为使用的是队列消息，所以队列中的所有消费者会自动进行负载均衡设计。

### 消息持久化

所谓消息持久化指的是即使rabbitmq服务(主机)停止后，其为消费的消息 也可以在重启之后继续进行消费处理。

1、观察临时消息队列，在rabbitmq里面队列可以有用户在使用的时候创建：

**private static final** String ***QUEUE\_NAME*** = **"msg.yangmi.tmp"**; *// 消息队列名称*

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

channel.basicPublish(**""**, ***QUEUE\_NAME***, **null**, msg.getBytes());

在这个时候如果服务停止了，并且有未消费的消息存在，则在重新启动之后这些消息将被清空。

1. 建立持久化消息：

**private static final** String ***QUEUE\_NAME*** = **"msg.yangmi"**; *// 消息队列名称*

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

channel.basicPublish(**""**, ***QUEUE\_NAME***, MessageProperties.***PERSISTENT\_TEXT\_PLAIN***, msg.getBytes());

如果不编写“ MessageProperties.PERSISTENT\_TEXT\_PLAIN”则消息无法被持久化。

### 5、虚拟主机

虚拟主机可以理解为rabbitmq中的重大特点，实际上在之前就已经存在有虚拟主机了，因为每一个rabbitmq如果没有配置则会默认存在一个寻主机的信息，当然用户也可以建立属于自己的虚拟主机，并且每一个虚拟主机里面有自己的队列信息。

1. 虚拟主机的配置最简单的做法是直接砸控制台上通过界面创建；

创建完成后要设置访问用户及操作权限。

1. 命令行的模式完成创建：rabbitmqctl

/usr/local/rabbitmq/sbin/rabbitmqctl add\_vhost bq

/usr/local/rabbitmq/sbin/rabbitmqctl set\_permissions -p bq yangmi ".\*" ".\*" ".\*"

1. 程序中设置虚拟主机：

factory.setVirtualHost(**"yangmi"**);

使用虚拟主机的最大好处在于可以区分不同用户的操作空间。

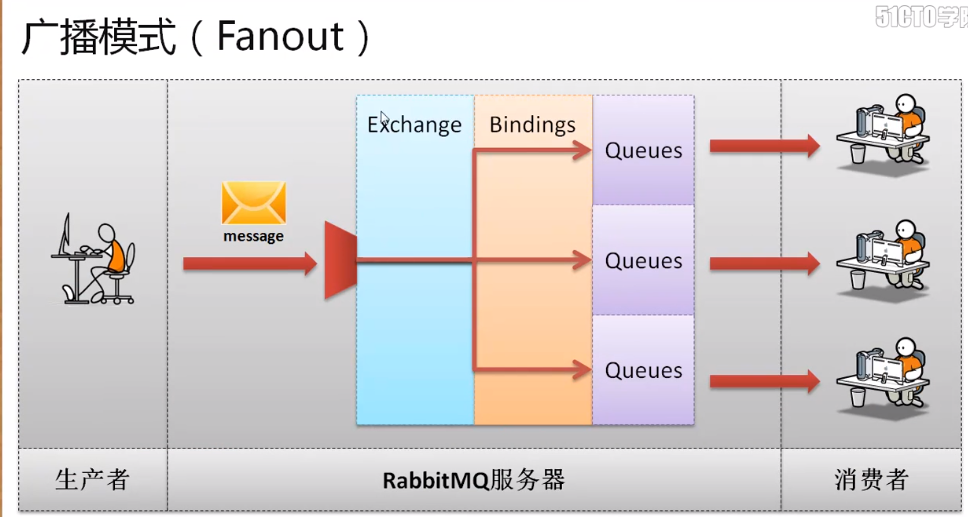
## rabbit-0208\_虚拟主机

exchange主要作用是在于一方面接收生产者发送的消息，另外一方面负责向消费者队列进行消息的推送。

在exchange之中有如下几种类型：topic、direct、fanout。

### 广播模式（fanout）

广播指的是一条消息将被所有的消费者进行处理，而要想实现广播，必须利用exchange处理完成。



1、定义生产者实现广播模式

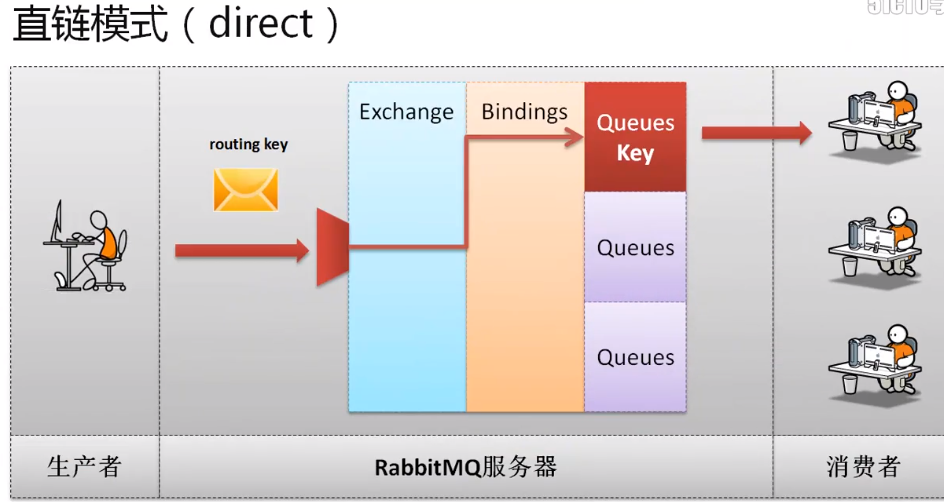
**public class** MessageProducer {  
 **private static final** String ***EXCHANGE\_NAME*** = **"yangmi.msg.fanout"**; *// 定义Exchange的名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 *// 设置虚拟主机* factory.setVirtualHost(**"yangmi"**);  
 *// 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 *// 定义EXCHANGE的声明* channel.exchangeDeclare(***EXCHANGE\_NAME***, **"fanout"**);  
 **long** start = System.*currentTimeMillis*();  
 **for** (**int** x = 0; x < 1000; x++) {  
 String msg = **"yangmi"** + x;  
 *// 进行消息发送* channel.basicPublish(***EXCHANGE\_NAME***, **""**, MessageProperties.***PERSISTENT\_TEXT\_PLAIN***, msg.getBytes());  
 }  
 System.***out***.println(**"耗时："** + (System.*currentTimeMillis*() - start));  
 channel.close();  
 connection.close();  
 }  
}

1. 生产者通过exchange发送消息，而后消费者通过exchange取出消息

**public class** MessageProducer {  
 **private static final** String ***EXCHANGE\_NAME*** = **"yangmi.msg.fanout"**; *// 定义Exchange的名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 *// 设置虚拟主机* factory.setVirtualHost(**"yangmi"**);  
 *// 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 *// 定义EXCHANGE的声明* channel.exchangeDeclare(***EXCHANGE\_NAME***, **"fanout"**);  
 **long** start = System.*currentTimeMillis*();  
 **for** (**int** x = 0; x < 1000; x++) {  
 String msg = **"yangmi"** + x;  
 *// 进行消息发送* channel.basicPublish(***EXCHANGE\_NAME***, **""**, MessageProperties.***PERSISTENT\_TEXT\_PLAIN***, msg.getBytes());  
 }  
 System.***out***.println(**"耗时："** + (System.*currentTimeMillis*() - start));  
 channel.close();  
 connection.close();  
 }  
}

### 直连模式（direct）

直连模式的特点就是routingkey的使用，routingkey直白一点就好比每一个人的电话号码一样，如果现在该消息就要求指定一个具备指定routingkey的操作者进行处理，那么只需要两个的routingkey匹配即可。



1. 定义生产者

**public class** MessageProducer {  
 **private static final** String ***EXCHANGE\_NAME*** = **"yangmi.msg.direct"**; *// 定义Exchange的名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 *// 设置虚拟主机* factory.setVirtualHost(**"yangmi"**);  
 *// 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 *// 定义EXCHANGE的声明* channel.exchangeDeclare(***EXCHANGE\_NAME***, **"direct"**);  
 **long** start = System.*currentTimeMillis*();  
 **for** (**int** x = 0; x < 1000; x++) {  
 String msg = **"yangmi"** + x;  
 *// 进行消息发送* channel.basicPublish(***EXCHANGE\_NAME***, **"yangmi-key"**, MessageProperties.***PERSISTENT\_TEXT\_PLAIN***, msg.getBytes());  
 }  
 System.***out***.println(**"耗时："** + (System.*currentTimeMillis*() - start));  
 channel.close();  
 connection.close();  
 }  
}

1. 定义消费者，只有指定的routingkey队列才能取得消息：

**public class** MessageConsumer {  
 **private static final** String ***EXCHANGE\_NAME*** = **"yangmi.msg.direct"**; *// 定义Exchange的名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 factory.setVirtualHost(**"yangmi"**);  
 *// 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 *// 定义EXCHANGE的声明* channel.exchangeDeclare(***EXCHANGE\_NAME***, **"direct"**);  
 *// 通过通道获取一个队列名称* String queueName = channel.queueDeclare().getQueue();  
 channel.queueBind(queueName, ***EXCHANGE\_NAME***, **"yangmi-key"**);  
 *// 在RabbitMQ里面所有的消费者的信息是通过一个回调方法完成的* Consumer consumer = **new** DefaultConsumer(channel) {  
 *// 需要覆写指定的方法实现消息消费处理* @Override  
 **public void** handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte**[] body) **throws** IOException {  
 String message = **new** String(body);  
 System.***out***.println(**"消息消费者A："** + message);  
 }  
 };  
 *// 消息消费* channel.basicConsume(queueName, consumer);  
 }  
}

### 主题模式（topic）

所谓的主题模式也就相当于一种广播模式，这个exchange之中实际上只需要控制好exchange名字、routingkey的信息，就实现了连接处理模式。

1. 定义生产者

**public class** MessageProducer {  
 **private static final** String ***EXCHANGE\_NAME*** = **"yangmi.msg.topic"**; *// 定义Exchange的名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 *// 设置虚拟主机* factory.setVirtualHost(**"yangmi"**);  
 *// 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 *// 定义EXCHANGE的声明* channel.exchangeDeclare(***EXCHANGE\_NAME***, **"topic"**);  
 **long** start = System.*currentTimeMillis*();  
 **for** (**int** x = 0; x < 1000; x++) {  
 String msg = **"yangmi"** + x;  
 *// 进行消息发送* **if** (x % 2 == 0) {  
 channel.basicPublish(***EXCHANGE\_NAME***, **"yangmi-key-1"**, MessageProperties.***PERSISTENT\_TEXT\_PLAIN***, msg.getBytes());  
 }**else**{  
 channel.basicPublish(***EXCHANGE\_NAME***, **"yangmi-key-2"**, MessageProperties.***PERSISTENT\_TEXT\_PLAIN***, msg.getBytes());  
 }  
 }  
 System.***out***.println(**"耗时："** + (System.*currentTimeMillis*() - start));  
 channel.close();  
 connection.close();  
 }  
}

1. 定义消费者

**public class** MessageConsumer {  
 **private static final** String ***EXCHANGE\_NAME*** = **"yangmi.msg.topic"**; *// 定义Exchange的名称* **private static final** String ***HOST*** = **"192.168.1.217"**; *// 消息服务的主机* **private static final** Integer ***PORT*** = 5672; *// 消息服务的端口号* **public static void** main(String[] args) **throws** Exception {  
 *// 建立一个连接工厂，所有的连接信息在此配置* ConnectionFactory factory = **new** ConnectionFactory();  
 factory.setHost(***HOST***);  
 factory.setPort(***PORT***);  
 factory.setUsername(**"yangmi"**);  
 factory.setPassword(**"ym"**);  
 factory.setVirtualHost(**"yangmi"**);  
 *// 定义一个新的RabbitMQ连接* Connection connection = factory.newConnection();  
 *// 创建一个通讯的通道* Channel channel = connection.createChannel();  
 *// 定义EXCHANGE的声明* channel.exchangeDeclare(***EXCHANGE\_NAME***, **"topic"**);  
 *// 通过通道获取一个队列名称* String queueName = channel.queueDeclare().getQueue();  
 channel.queueBind(queueName, ***EXCHANGE\_NAME***, **"yangmi-key-1"**);  
 *// 在RabbitMQ里面所有的消费者的信息是通过一个回调方法完成的* Consumer consumer = **new** DefaultConsumer(channel) {  
 *// 需要覆写指定的方法实现消息消费处理* @Override  
 **public void** handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte**[] body) **throws** IOException {  
 String message = **new** String(body);  
 System.***out***.println(**"消息消费者A-mldn-key-1："** + message);  
 }  
 };  
 *// 消息消费* channel.basicConsume(queueName, consumer);  
 }  
}

主题模式类似于广播模式与直连模式的整合操作，所有的消费者都可以接收到对应的主题信息。

## Spring整合RabbitMQ组件

rabbitmq与spring整合需要的开发包：

<**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 <**scope**>test</**scope**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.rabbitmq</**groupId**>  
 <**artifactId**>amqp-client</**artifactId**>  
 <**version**>4.1.1</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.amqp</**groupId**>  
 <**artifactId**>spring-rabbit</**artifactId**>  
 <**version**>1.7.3.RELEASE</**version**>  
</**dependency**>

如果想要进行spring的配置（结合maven来讲）必须考虑多个profile的情况，所以需要建立一个rabbitmq.properties配置文件。

*# RabbitMQ的连接主机地址***mq.rabbit.host**=**192.168.1.217***# RabbitMQ的连接端口号***mq.rabbit.port**=**5672***# RabbitMQ的虚拟主机名称***mq.rabbit.vhost**=**yangmi***# RabbitMQ的exchange名称***mq.rabbit.exchange**=**yangmi.msg.spring***# 用户名***mq.rabbit.username**=**yangmi***# 密码***mq.rabbit.password**=**y**

### 1、定义消费者程序

1、spring整合所有消息组件的处理模式都采用了同样的操作接口：MessaeListener，也就说要进行消息处理，那么一定需要一个消息处理类：

**public class** MessageConsumer **implements** MessageListener {  
  
 @Override  
 **public void** onMessage(Message message) {  
 System.***out***.println(**"【\*\*\* 消息消费者 \*\*\*】"** + message);  
 }  
}

2、如果想要进行消费者的使用，则一定要通过spring配置文件整合，建立spring-consumer.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:rabbit="http://www.springframework.org/schema/rabbit"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.3.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd  
 http://www.springframework.org/schema/rabbit http://www.springframework.org/schema/rabbit/spring-rabbit-1.7.xsd"**>  
  
 *<!-- 定义rabbitmq配置的相关属性文件信息 -->* <**context:property-placeholder location="classpath:rabbitmq.properties"**/>  
 *<!-- 如果要想进行RabbiMQ的操作管理，则首先一定要准备出一个连接工厂类 -->* <**rabbit:connection-factory id="connectionFactory" host="${mq.rabbit.host}" port="${mq.rabbit.port}"  
 username="${mq.rabbit.username}" password="${mq.rabbit.password}"  
 virtual-host="${mq.rabbit.vhost}"**/>  
 *<!-- 所有的连接工厂要求被RabbitMQ所管理 -->* <**rabbit:admin connection-factory="connectionFactory"**/>  
 *<!-- 创建一个队列信息 -->* <**rabbit:queue id="myQueue" durable="true" auto-delete="true" exclusive="false" name="yangmi.queue.title"**/>  
 *<!-- 下面实现一个直连的操作模式 -->* <**rabbit:direct-exchange id="mqDirect" name="${mq.rabbit.exchange}" durable="true" auto-delete="true"**>  
 <**rabbit:bindings**>  
 *<!-- exchange绑定到指定的队列之中 -->* <**rabbit:binding queue="myQueue" key="yangmi-key"**></**rabbit:binding**>  
 </**rabbit:bindings**>  
 </**rabbit:direct-exchange**>  
 *<!-- 定义具体的消费处理类 -->* <**bean id="messageConsumer" class="cn.ym.com.consumer.MessageConsumer"**/>  
 *<!-- 启动消费监听程序 -->* <**rabbit:listener-container connection-factory="connectionFactory"**>  
 <**rabbit:listener ref="messageConsumer" queues="myQueue"**/>  
 </**rabbit:listener-container**>  
</**beans**>

1. 测试程序：

@ContextConfiguration(locations = {**"classpath:spring-consumer.xml"**})  
@RunWith(SpringJUnit4ClassRunner.**class**)  
**public class** TestConsumer {  
  
 @Test  
 **public void** testConsumer(){  
 **try** {  
 *// 启动消费容器* Thread.*sleep*(Long.***MAX\_VALUE***);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

### 定义消息生产者

如果要进行生产者定义，在Spring中会利用一个消息的处理模板进行消息的发送操作。

1. 建立一个消息发送的业务接口：

**public interface** IMessageService {  
 **public void** send(String msg) ;  
}

2、通过spring-consumer.xml文件复制一份spring-producer.xml配置文件：

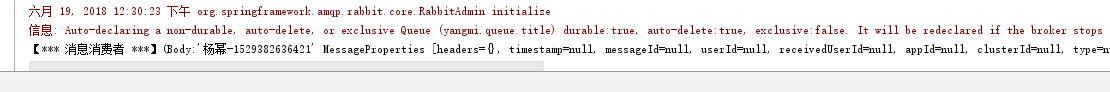
*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:rabbit="http://www.springframework.org/schema/rabbit"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.3.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd  
 http://www.springframework.org/schema/rabbit http://www.springframework.org/schema/rabbit/spring-rabbit-1.7.xsd"**>  
 <**context:component-scan base-package="cn.ym.com"**/>  
 *<!-- 定义rabbitmq配置的相关属性文件信息 -->* <**context:property-placeholder location="classpath:rabbitmq.properties"**/>  
 *<!-- 如果要想进行RabbiMQ的操作管理，则首先一定要准备出一个连接工厂类 -->* <**rabbit:connection-factory id="connectionFactory" host="${mq.rabbit.host}" port="${mq.rabbit.port}"  
 username="${mq.rabbit.username}" password="${mq.rabbit.password}"  
 virtual-host="${mq.rabbit.vhost}"**/>  
 *<!-- 所有的连接工厂要求被RabbitMQ所管理 -->* <**rabbit:admin connection-factory="connectionFactory"**/>  
 *<!-- 创建一个队列信息 -->* <**rabbit:queue id="myQueue" durable="true" auto-delete="true" exclusive="false" name="yangmi.queue.title"**/>  
 *<!-- 下面实现一个直连的操作模式 -->* <**rabbit:direct-exchange id="mqDirect" name="${mq.rabbit.exchange}" durable="true" auto-delete="true"**>  
 <**rabbit:bindings**>  
 *<!-- exchange绑定到指定的队列之中 -->* <**rabbit:binding queue="myQueue" key="yangmi-key"**></**rabbit:binding**>  
 </**rabbit:bindings**>  
 </**rabbit:direct-exchange**>  
 *<!-- 所有整合的消息系统都会有一个模版 -->* <**rabbit:template id="amqpTemplate" exchange="${mq.rabbit.exchange}" connection-factory="connectionFactory"**/>  
</**beans**>

3、进行消息发送处理子类：

@Service  
**public class** MessageServiceImpl **implements** IMessageService {  
  
 @Autowired  
 **private** AmqpTemplate **amqpTemplate**;  
  
 @Override  
 **public void** send(String msg) {  
 *//指定routingkey* **this**.**amqpTemplate**.convertAndSend(**"yangmi-key"**, msg);  
 }  
}

4、测试消息发送：

@ContextConfiguration(locations = {**"classpath:spring-producer.xml"**})  
@RunWith(SpringJUnit4ClassRunner.**class**)  
**public class** TestProducer {  
  
 @Autowired  
 **private** IMessageService **messageService**;  
  
 @Test  
 **public void** testSend() {  
 **this**.**messageService**.send(**"杨幂-"** + System.*currentTimeMillis*());  
 }  
}



## RabbitMQ集群

rabbitmq集群设计分为两类：普通集群、镜像队列，其中镜像队列为重点。rabbitmq的集群本身不带有HA机制以及负载均衡机制。如果要使用复杂的rabbitmq集群设计，那么必须要求在结合负载均衡组件（HAProxy、LVS）以及keepalived（HA机制）。

### 搭建RabbitMQ基础集群

1. 【rabbitmq-server-\*】修改对应主机host地址（也可以用IP代替）

192.168.1.216 rabbitmq-server-a

192.168.1.215 rabbitmq-server-b

192.168.1.214 rabbitmq-server-c

2、【rabbitmq-server-\*】如果要配置基础集群必须保证有一个前提，要求erlang.cookie信息同步，该配置文件的路径为：/root/.erlang.cookie。

3、【rabbitmq-server-\*】启动所有主机的rabbitmq服务：

/usr/local/rabbitmq/sbin/rabbitmq-server start

4、【rabbitmq-server-a】查看当前的集群状态：

/usr/local/rabbitmq/sbin/rabbitmqctl cluster\_status

此时显然没有其他节点的集群

5、【rabbitmq-server-b、c】如果要想将“rabbitmq-server--b”、“rabbitmq-server--c”配置为“rabbitmq-server-a”的从主机，那么要停掉两个集群的APP服务；

/usr/local/rabbitmq/sbin/rabbitmqctl stop\_app

此时rabbitmq没有停止。

6、【rabbitmq-server-b、c】分别追加“rabbitmq-server-a”主机项：

/usr/local/rabbitmq/sbin/rabbitmqctl join\_cluster rabbit@rabbitmq-server-a

7、【rabbitmq-server-b、c】启动APP服务：

/usr/local/rabbitmq/sbin/rabbitmqctl start\_app

8、【rabbitmq-server-a】查看集群状态：

/usr/local/rabbitmq/sbin/rabbitmqctl cluster\_status

9、【rabbitmq-server-a】创建用户

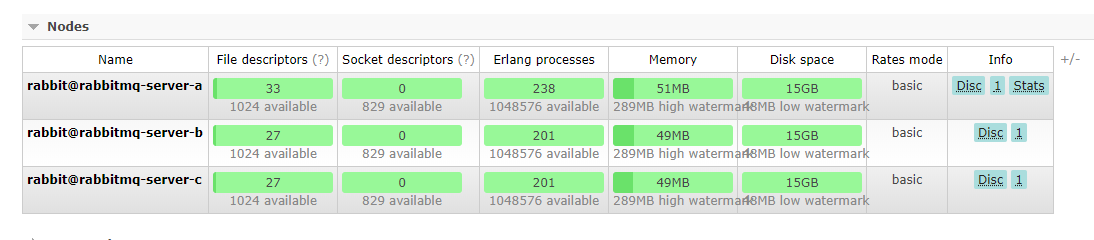
/usr/local/rabbitmq/sbin/rabbitmqctl add\_user yangmi ym

/usr/local/rabbitmq/sbin/rabbitmqctl set\_user\_tags yangmi administrator（追加到管理员组）

/usr/local/rabbitmq/sbin/rabbitmqctl set\_permissions -p / yangmi ".\*" ".\*" ".\*"（设置权限）

9、【rabbitmq-server-a】前端查看集群状态

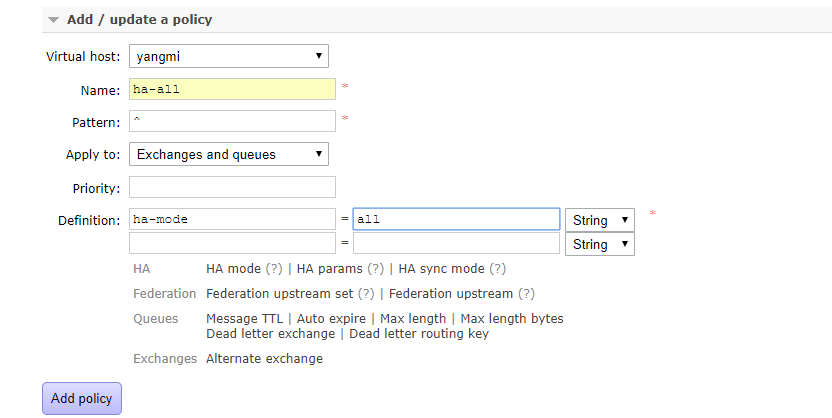
<http://192.168.1.216:15672/>



### 镜像队列

所谓镜像队列指的是将某一主机中的队列信息直接复制到其他主机中，这样就可以保证在实际运行中每一台rabbitmq具备的数据都是一样的，这样就实现不同主机的读写操作。

1. 控制台配置：



2、命令行配置：

/usr/local/rabbitmq/sbin/rabbitmqctl set\_policy -p yangmi ha-all "^" '{"ha-mode":"all"}'

这行命令在vhost名称为yangmi创建了一个策略，策略名称为ha-all, 策略模式为 all 即复制到所有节点，包含新增节点,策略正则表达式为 “^” 表示所有匹配所有队列名称。

所谓的镜像队列只是进行数据的副本而已。