RabbitMQ消息组件

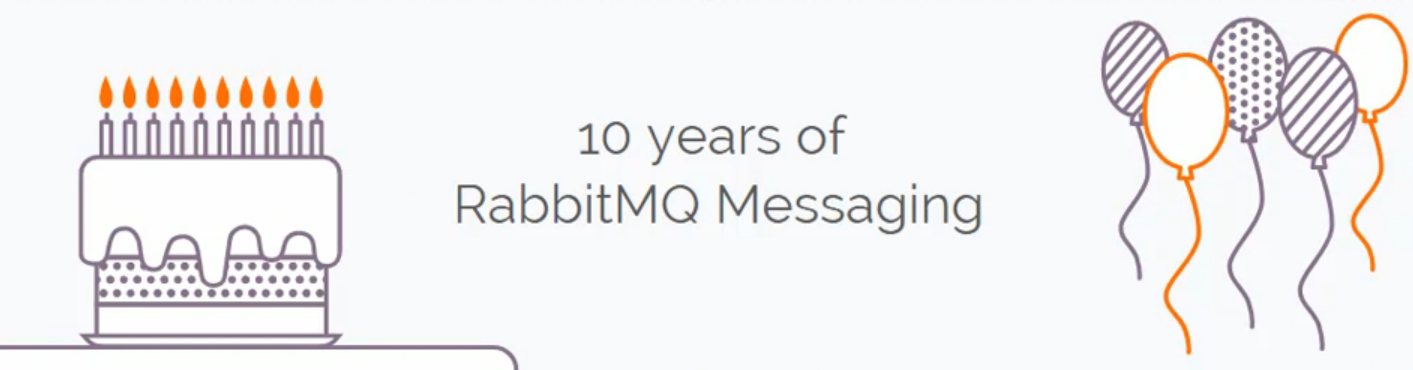
# RabbitMQ简介

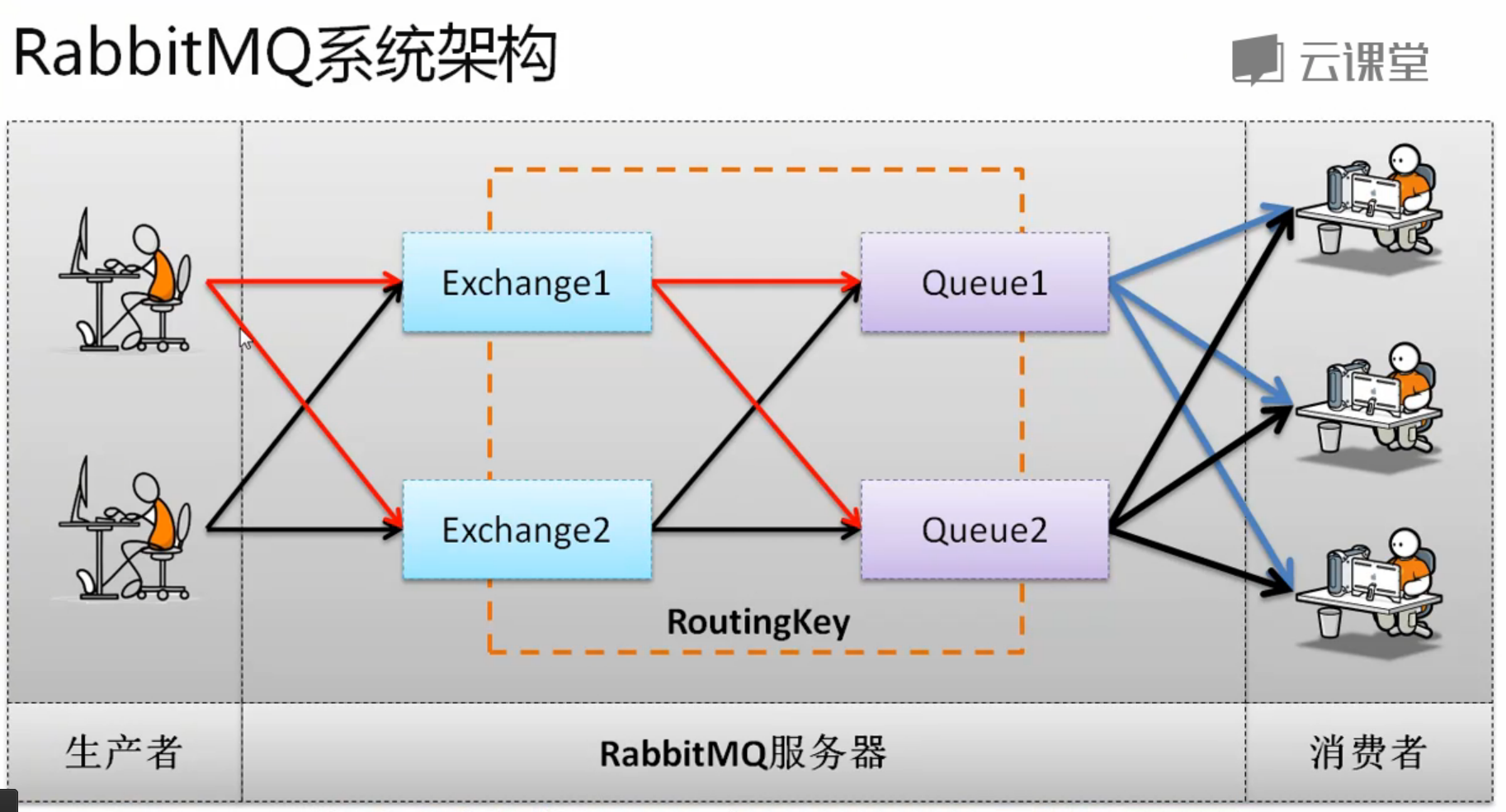
* RabbitMQ是一个油erlang开发的AMQP(Advanced Message Queue)的开源实现；
* RabbitMQ是由RabbitMQ Technologies Ltd开发并且提供商业支持的。该公司在2010年4月被SpringSource(VMWare的一个部门)收购。在2013年5月被并入Pivotal。
* RabbitMQ的官网是<http://www.rabbitmq.com>。

## 具体内容

如果从消息组件来讲主要划分为两类：

* JMS组件：ActiveMQ（慢）；
* AMQP组件（协议）：性能是最高的，而AMQP有两个主要的开源
  + RabbitMQ：使用最为广泛、速度也很快；
  + Kafka：是在大数据时代产生，明日之星。





在所有的消息系统之中，其基本组成一定就是生产者、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：

* Exchange：交换空间，管理所有的队列；
* Queue：数据队列；
* RoutingKey：如果你现在所有队列的RoutingKey都一样，那么久属于广播消息，而如果不一样，则表示点对点消息。

RabbitMQ核心概念

* Broker：消息队列服务主机；
* Exchange：消息交换机，它制定消息按什么规则，路由到哪个队列；
* Queue：消息队列载体，每个消息都会被投入到一个或多个队列；
* Binding：绑定，把exchange和queue按照路由规则绑定起来；
* Routing Key：路由关键字，exchange根据这个关键字进行消息投递；
* Vhost：虚拟主机，一个broker里可设置多个vhost，实现用户的权限分离；
* Producer：消息生产者；
* Consumer：消息消费者；
* Channel：消息通道，在客户端的每个连接里，可建立多个channel，每个channel代表一个会话任务。

在RabbitMQ里面提供有一个虚拟主机的概念，所谓的虚拟主机可以理解为不同的用户空间，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

## 具体内容

RabbitMQ依靠的开发语言为ErLang，所以如果要想使用RabbitMQ服务，首先一定要在系统之中进行此开发环境的配置。

1. 安装erlang：
   1. 下载erlang/otp开发包（otp\_src\_20.1.tar.gz）：<http://www.erlang.org/download/otp_src_20.1.tar.gz>
   2. 解压（otp\_src\_20.1.tar.gz）：tar xzvf /otp\_src\_20.1.tar.gz –C /usr/local/src/
2. 进入到erlang的源代码目录之中对其进行编译处理：
   1. 一定要保证你的主机上已经配置了JDK；
   2. 如果要想编译erlang源代码，则需要一个类库的支持：

apt-get install build-essential

apt-get install g++

apt-get install libncurses5-dev

apt-get install libssl-dev

apt-get install m4

apt-get install unixodbc unixodbc-dev

apt-get install freeglut3-dev libwxgtk3.0-dev

apt-get install xsltproc

apt-get install fop

apt-get install tk8.5

apt-get install make

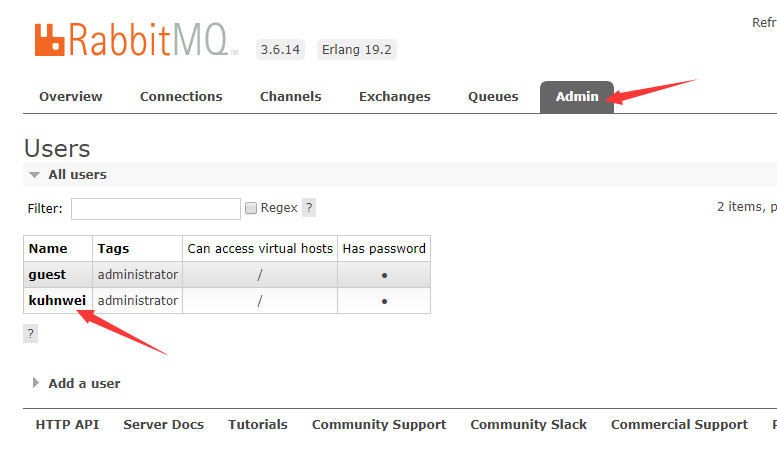
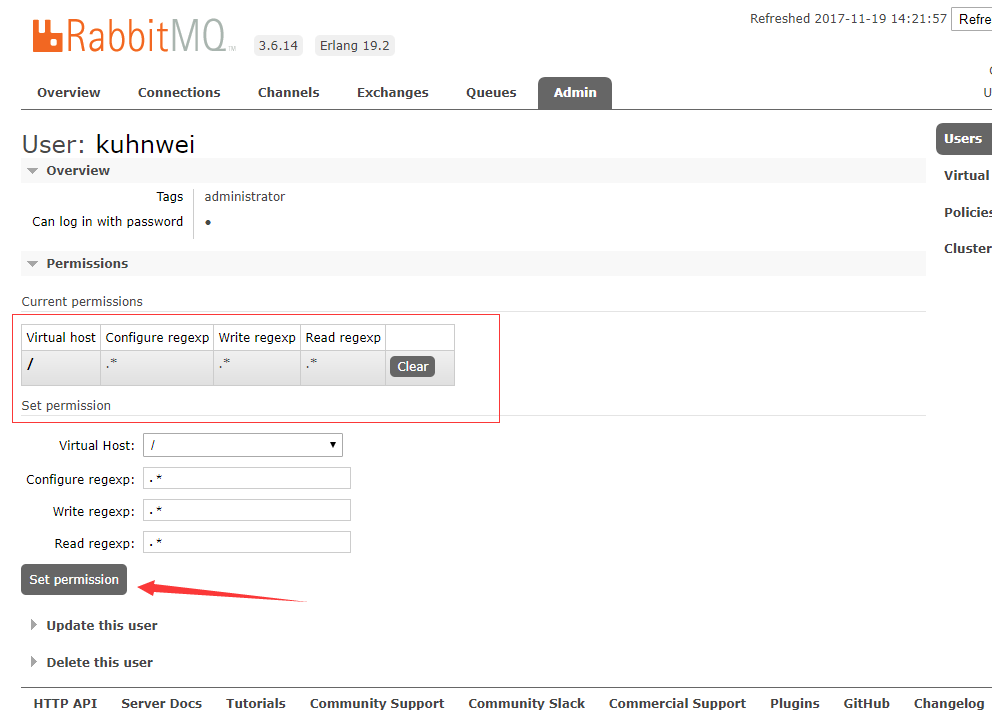
* 1. 建立一个erlang编译后的文件目录：mkdir –p /usr/local/erlang；
  2. 进入到源代码所在目录：cd /usr/local/src/otp\_src\_20.1；
  3. 进行编译配置：./configure --prefix=/usr/local/erlang；
  4. 进行编译与安装：make&&make install；
  5. 直接安装erlang：apt-get install erlang

1. 考虑到随后可能的使用，建议将erlang开发包配置到环境属性之中：
   1. 打开环境配置文件：vim /etc/profile；
   2. 追加新的环境配置：exprt ERLANG\_HOME=/usr/local/erlang；exprt=PATH=$PATH:$ERLANG\_HOME/bin:；
   3. 使配置立即生效：source /etc/profile；
2. 测试erlang是否可以使用：
   1. 直接输入：erl；
   2. io:format(“hello world”).；
   3. 如果要想退出编译器，直接输入“halt().”

# 安装并配置RabbitMQ

## 具体内容

RabbitMQ的配置过程比较容易，但是这里面有一个小小的注意项，一般而言，在Linux之中所有的压缩文件都会以“.tar”的格式，可是RabbitMQ如果你通过官方网站下载的你会发现可以使用的是“xz”文件。

1. 下载rabbitmq开发包：<https://dl.bintray.com/rabbitmq/binaries/rabbitmq-server-generic-unix-3.6.14.tar.xz>；而后进行解压缩
   1. tar.xz解压缩：xz –d rabbitmq-server-generic-unix-3.6.14.tar.xz
   2. tar解压缩：tar xvf rabbitmq-server-generic-unix-3.6.14.tar –C /usr/local/
2. 做一个更名处理：mv rabbitmq\_server-3.6.14 rabbitmq；
3. 启动rabbitmq的服务：/usr/local/rabbitmq/sbin/rabbitmq-server start
4. 后台启动rabbitmq的服务：./rabbitmq-server -detached
5. RabbitMQ本身自带有系统监控也就是说它存在有一个WEB的客户端，但是如果你要想进行自己的操作，强烈建议做一个新的管理员账户（kuhnwei/650901）;
   1. /usr/local/rabbitmq/sbin/rabbitmqctl add\_user kuhnwei 650901
6. 此时kuhnwei的账户只是一个普通用户，随后需要将其加入到管理员之中；
   1. /usr/local/rabbitmq/sbin/rabbitmqctl set\_user\_tags kuhnwei administrator
7. 默认情况下rabbitmq的服务器是不带有webserver启动的，需要自己配置启动，执行如下命令：/usr/local/rabbitmq/sbin/rabbitmq-plugins enable rabbitmq\_management
8. 删除掉rabbitmq进程：
   1. ps –ef | grep rabbit
   2. kill <rabbit进程编号>
   3. 重新启动:rabbit-server start
9. 当启动了rabbitmq之后就表示启动了一个webserver，所以可以在浏览器中输入地址访问：192.168.127.129:15672  
     
     
   

# 使用Java操作RabbitMQ

## 具体内容

本次的开发将给予Maven建立RabbitMQ的相关程序，所以首先要建立一父项目：rabbitdemo，随后修改pom.xml配置文件，追加有相应的配置项：

|  |
| --- |
| <?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>com.kuhnwei</groupId>  <artifactId>rabbitdemo</artifactId>  <version>1.0-SNAPSHOT</version>  <modules>  <module>rabbit-base</module>  </modules>  <packaging>pom</packaging>  <properties>  <jdk.version>1.8</jdk.version>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>com.rabbitmq</groupId>  <artifactId>amqp-client</artifactId>  <version>4.1.1</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <finalName>rabbitdemo</finalName>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <!-- 源代码使用的开发版本 -->  <source>${jdk.version}</source>  <!-- 需要生成的目标class文件的编译版本-->  <target>${jdk.version}</target>  <encoding>${project.build.sourceEncoding}</encoding>  </configuration>  </plugin>  </plugins>  </build> </project> |

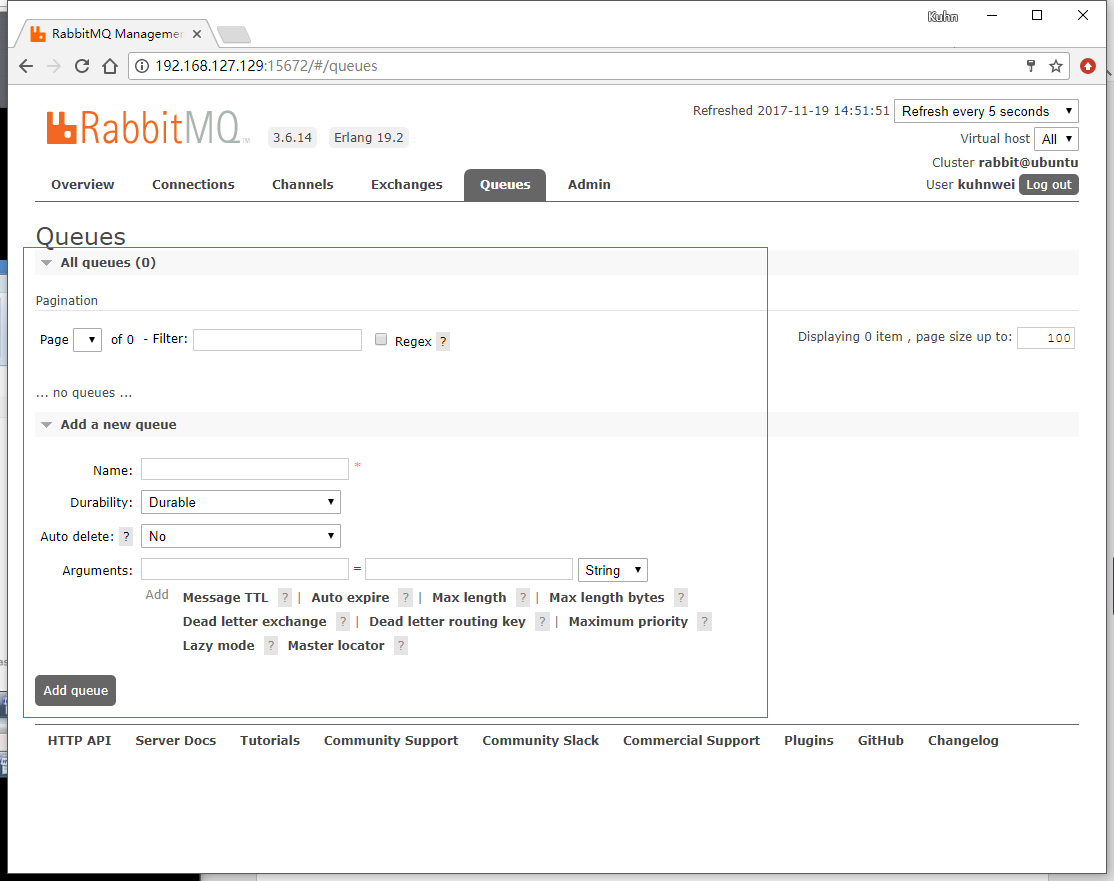
在此项目的基础上要求搭建生产者、消费者等相关程序。

### 建立消息生产者

1. 首先为了方便起见，建立有一个rabbit-base的子模块；

|  |
| --- |
| <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>rabbitdemo</artifactId>  <groupId>com.kuhnwei</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>rabbit-base</artifactId>  <packaging>jar</packaging>  <name>rabbit-base</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>  <dependencies>  <dependency>  <groupId>com.rabbitmq</groupId>  <artifactId>amqp-client</artifactId>  </dependency>  </dependencies> </project> |

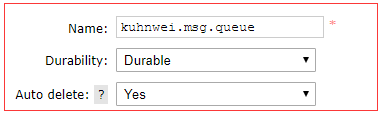
1. 随后现在要进行一个队列访问，但是这个队列信息可以提前设计好，也可以通过程序动态设计。



在进行队列信息创建的时候会有如下可用选项：

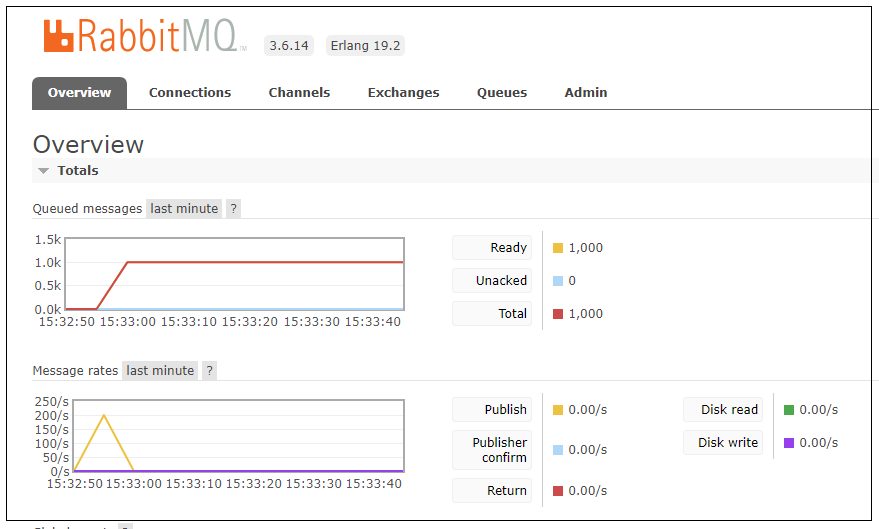
* + Name: 队列的名称；
  + 消息保存类型（Durability）：持久化保持（Druid）、瞬时状态（Transient）；
  + 允许自动删除（Auto delete）。

本次建立一个“kuhnwei.msg.queue”消息，该消息为持久消息，并且允许自动删除；

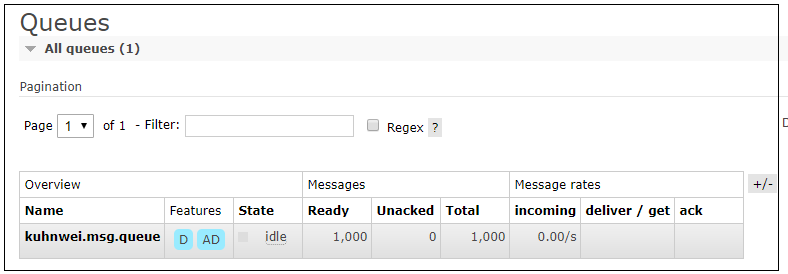


1. 定义一个消息的生产者程序：

|  |
| --- |
| package com.kuhnwei.examples.producer; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection; import com.rabbitmq.client.ConnectionFactory; import java.io.IOException; import java.util.concurrent.TimeoutException; */\*\*  \* 消息生产者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:07  \*/* public class MessageProducer {  */\*\*  \* 消息队列名称  \*/* private static final String *QUEUE\_NAME* = "kuhnwei.msg.queue";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  public static void main(String[] args) throws IOException, TimeoutException {  // 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("kuhnwei");  factory.setPassword("650901");  //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();  /\*  定义该通道使用的队列名称，此时的队列已经创建过了  第一个参数queue：队列名称（这个参数可能存在也可能不存在）  第二个参数durable：是否为持久保持  第三个参数exclusive：是否为专用的队列信息  第四个参数：是否允许自动删除  第五个参数：  \*/  channel.queueDeclare(*QUEUE\_NAME*, true, false, true, null);  long start = System.*currentTimeMillis*();  for (int x = 0; x < 1000; x ++){  /\*  exchange:  routingKey:  props:  body:  \*/  String msg = "rabbitmq - " + x;  // 进行消息发送  channel.basicPublish("", *QUEUE\_NAME*, null, msg.getBytes());  }  long end = System.*currentTimeMillis*();  System.*out*.println("本次操作所花费的时间：" + (end - start));  channel.close();  connection.close();  } } |



此时由于发送了1000条消息，并且该消息没有被消费，所以会有相应的未消费记录。



### 定义消息消费者

消费者的处理基本流程与生产者的形式类似的，因为消费者一定也要通过特定的队列进行处理操作。

范例：定义消费者

|  |
| --- |
| package com.kuhnwei.examples.consumer;  import com.rabbitmq.client.\*; import java.io.IOException;  */\*\*  \* 消息消费者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:39  \*/* public class MessageConsumer {  */\*\*  \* 消息队列名称  \*/* private static final String *QUEUE\_NAME* = "kuhnwei.msg.queue";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  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("kuhnwei");  factory.setPassword("650901");  //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函数完成的  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了消息之后，告诉服务器我已经把内容取走了，而后服务器就可以把内容删除掉。这种机制的好处在于整体消息消费的可靠性要高，但是缺点是性能将受到影响。

范例：实现消息应答

|  |
| --- |
| package com.kuhnwei.examples.consumer;  import com.rabbitmq.client.\*; import java.io.IOException;  */\*\*  \* 消息消费者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:39  \*/* public class MessageConsumer {  */\*\*  \* 消息队列名称  \*/* private static final String *QUEUE\_NAME* = "kuhnwei.msg.queue";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  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("kuhnwei");  factory.setPassword("650901");  //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函数完成的  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);  // 表示本消息通过一个消息的标签实现了应答处理  channel.basicAck(envelope.getDeliveryTag(), false);  }  };  // 消息消费  channel.basicConsume(*QUEUE\_NAME*, consumer);  } } |

此时会向服务器进行消息的接受确认，整个的过程除了性能稍微降低一点之外，其它的没有任何特点。

### 工作队列

现在的程序只定义了一个生产者和一个消费者，如果说现在一个生产者对应有多个消费则呢？那么就表示一个工作队列，而工作队列最大的特征在于：若干个消费者一起完成工作。

当启动了多个消费者之后，这些消费者会在一起共同完成所发出的消息消费处理，那么这样的处理可以保证消息处理的速度更加的快速。但是千万要记住，此时由于使用的是普通队列消息。

所以队列消息之中的所有消费者会自动进行负载均衡设计。

### 消息持久化

在之前所创建的消息队列形式实际上都会发现有一个持久化的选项，所谓的持久化本质在于：即便RabbitMQ服务停机之后其未消费的消息也可以在重启之后进行消费处理。

1. 观察临时消息队列，在RabbitMQ里面队列可以由用户在使用的时候创建；

|  |
| --- |
| private static final String *QUEUE\_NAME* = "kuhnwei.msg.queue.tmp"; |
| channel.queueDeclare(*QUEUE\_NAME*, false, false, true, null); |

这个时候如果服务器关闭，并且有未消费的消息存在，则在重新启动之后这些消息将会被清空。

1. 建立持久化消息：

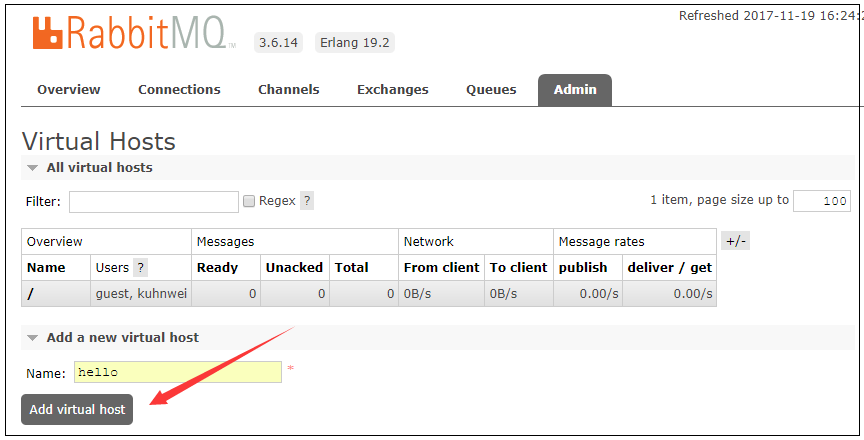
|  |
| --- |
| channel.queueDeclare(*QUEUE\_NAME*, true, false, true, null); |
| channel.basicPublish("", *QUEUE\_NAME*, MessageProperties.*PERSISTENT\_TEXT\_PLAIN*, msg.getBytes()); |

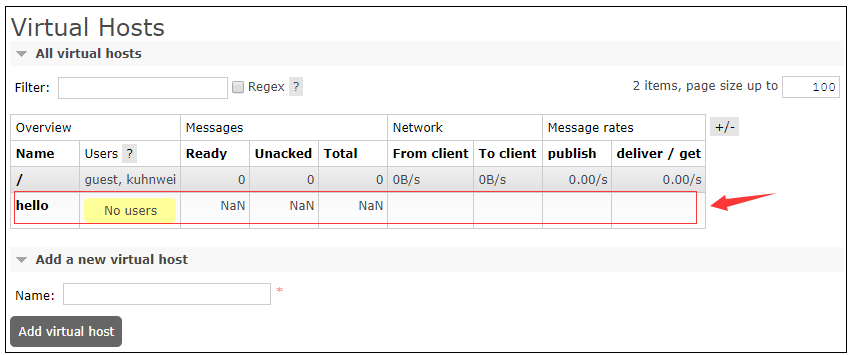
如果不编写“MessageProperties.*PERSISTENT\_TEXT\_PLAIN* “则消息无法被持久化。

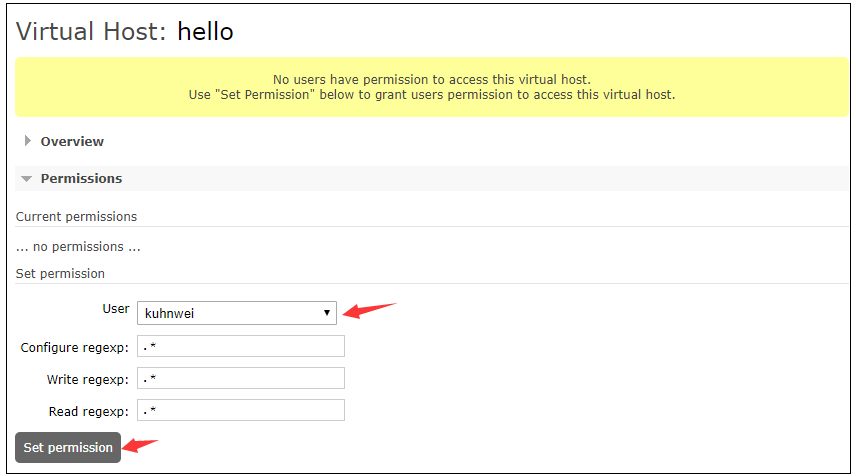
### 虚拟主机

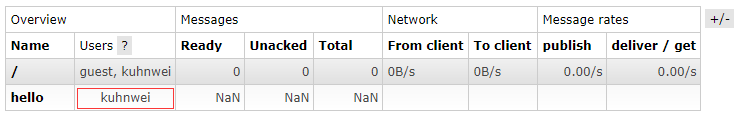
虚拟主机可以理解为RabbitMQ中的重大特点，实际上在之前就已经存在有虚拟主机了，因为每一个rabbitmq如果没有配置则会默认存在有一个虚拟主机的信息，当然，用户也可以建立属于自己的虚拟主机，并且每一个虚拟主机里有自己的队列信息。

1. 虚拟主机的配置最简单的做法是直接在控制台上通过界面创建；









如果你不想同故宫界面进行创建，那么也可以通过命令行的模式进行完成：rabbitmqctl;

1. 那么随后如果要想在项目之中去使用虚拟主机，则要在连接处进行配置：

|  |
| --- |
| // 设置虚拟主机 factory.setVirtualHost("hello"); |

使用虚拟主机的最大好处可以区分不同用户的操作空间。

## 发布订阅模式

## 具体内容

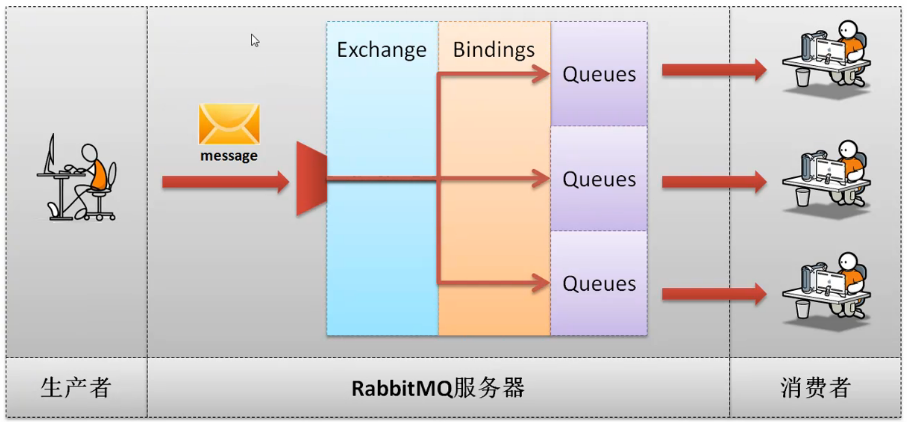
在之前所讲解的RabbitMQ操作里面实际上都有如下的几个局限：

* 在整个的处理之中并没有考虑到rabbitmq中的exchange、routingkey概念；
* 在整个的处理里面实际上只是编写了一个简单的队列名称；
* 在消息系统之中除了有队列消息之外一定会存在有主题消息，同一条消息可以被所有的消费者同时处理。

要想实现以上的功能，就必须清楚的认识到exchange的作用：exchange主要作用是在于一方面它接收生产者发送的消息，而另外一方面负责向队列进行消息的推送。

在Exchange之中有如下的几种类型：topic、director、fanout；

### 广播模式（fanout）



所谓的广播指的是一条消息将被所有的消费者进行处理，而要想实现广播的操作，就必须利用Exchange处理完成。

范例：定义生产者实现广播模式

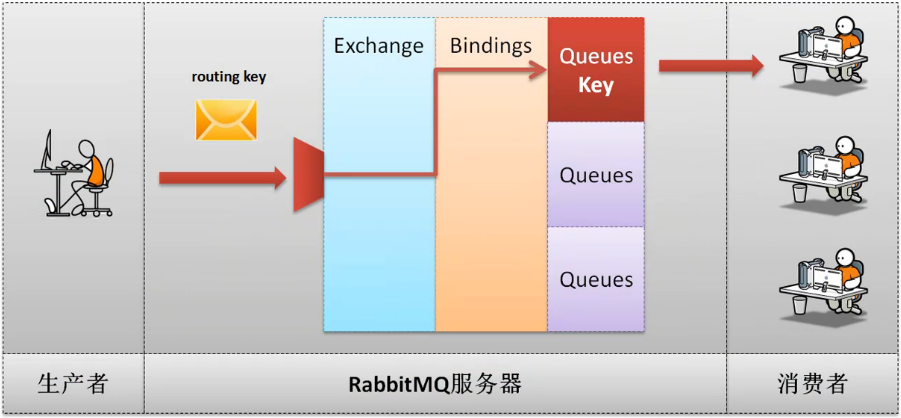
|  |
| --- |
| package com.kuhnwei.examples.producer; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection; import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.MessageProperties; import java.io.IOException; import java.util.concurrent.TimeoutException; */\*\*  \* 消息生产者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:07  \*/* public class MessageProducer {  */\*\*  \* Exchange名称  \*/* private static final String *EXCHANGE\_NAME* = "kuhnwei.msg.fanout";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  public static void main(String[] args) throws IOException, TimeoutException {  // 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("kuhnwei");  factory.setPassword("650901");  // 设置虚拟主机  factory.setVirtualHost("hello");  //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 < 10; x ++){  String msg = "rabbitmq - " + x;  // 进行消息发送  channel.basicPublish(*EXCHANGE\_NAME*, "", MessageProperties.*PERSISTENT\_TEXT\_PLAIN*, msg.getBytes());  }  long end = System.*currentTimeMillis*();  System.*out*.println("本次操作所花费的时间：" + (end - start));  channel.close();  connection.close();  } } |

1. 生产者同故宫exchange发送了消息，而后消费者要通过exchange取出消息。

|  |
| --- |
| package com.kuhnwei.examples.consumer; import com.rabbitmq.client.\*; import java.io.IOException; */\*\*  \* 消息消费者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:39  \*/* public class MessageConsumer {  */\*\*  \* Exchange名称  \*/* private static final String *EXCHANGE\_NAME* = "kuhnwei.msg.fanout";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  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("kuhnwei");  factory.setPassword("650901");  // 设置虚拟主机  factory.setVirtualHost("hello");  //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");  // 通过通道获取一个队列名称  String queueName = channel.queueDeclare().getQueue();  // 进行绑定处理  channel.queueBind(queueName, *EXCHANGE\_NAME*, "");  // 在RabbitMQ里面所有的消费者的信息是通过一个回调函数完成的  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(queueName, consumer);  } } |

现在的形式是通过了一个exchange整合了要使用的队列以及消费者。现在表示为广播模式，也就是说一个主题信息可以同时被多个消费者同时消费处理。

### 直连模式（direct）



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

范例：定义生产者

|  |
| --- |
| package com.kuhnwei.examples.producer;  import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection; import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.MessageProperties;  import java.io.IOException; import java.util.concurrent.TimeoutException;  */\*\*  \* 消息生产者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:07  \*/* public class MessageProducer {  */\*\*  \* Exchange名称  \*/* private static final String *EXCHANGE\_NAME* = "kuhnwei.msg.direct";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  public static void main(String[] args) throws IOException, TimeoutException {  // 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("kuhnwei");  factory.setPassword("650901");  // 设置虚拟主机  factory.setVirtualHost("hello");  //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 < 10; x ++){  String msg = "rabbitmq - " + x;  // 进行消息发送  channel.basicPublish(*EXCHANGE\_NAME*, "msg-key", MessageProperties.*PERSISTENT\_TEXT\_PLAIN*, msg.getBytes());  }  long end = System.*currentTimeMillis*();  System.*out*.println("本次操作所花费的时间：" + (end - start));  channel.close();  connection.close();  } } |

范例：定义消费者

|  |
| --- |
| package com.kuhnwei.examples.consumer;  import com.rabbitmq.client.\*; import java.io.IOException;  */\*\*  \* 消息消费者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:39  \*/* public class MessageConsumer {  */\*\*  \* Exchange名称  \*/* private static final String *EXCHANGE\_NAME* = "kuhnwei.msg.direct";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  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("kuhnwei");  factory.setPassword("650901");  // 设置虚拟主机  factory.setVirtualHost("hello");  //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();  // 进行绑定处理  channel.queueBind(queueName, *EXCHANGE\_NAME*, "msg-key");  // 在RabbitMQ里面所有的消费者的信息是通过一个回调函数完成的  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(queueName, consumer);  } } |

可以将rountingkey比喻为一个唯一标记，这样就可以将消息准确的送到消费者手中了。

### 主题模式（topic）

所谓的主题模式也就相当于是一种广播模式，你会发现在整个exchange之中实际上你只需要控制好exchange名字、以及roundingkey的信息就相当于实现了所有所谓的连接处理了。

1. 定义一个消息生产者

|  |
| --- |
| package com.kuhnwei.examples.producer; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection; import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.MessageProperties; import java.io.IOException; import java.util.concurrent.TimeoutException; */\*\*  \* 消息生产者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:07  \*/* public class MessageProducer {  */\*\*  \* Exchange名称  \*/* private static final String *EXCHANGE\_NAME* = "kuhnwei.msg.topic";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  public static void main(String[] args) throws IOException, TimeoutException {  // 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("kuhnwei");  factory.setPassword("650901");  // 设置虚拟主机  factory.setVirtualHost("hello");  //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 < 10; x ++){  String msg = "rabbitmq - " + x;  // 进行消息发送  if (x % 2 == 0) {  channel.basicPublish(*EXCHANGE\_NAME*, "msg-key-1", MessageProperties.*PERSISTENT\_TEXT\_PLAIN*, msg.getBytes());  } else {  channel.basicPublish(*EXCHANGE\_NAME*, "msg-key-2", MessageProperties.*PERSISTENT\_TEXT\_PLAIN*, msg.getBytes());  }  }  long end = System.*currentTimeMillis*();  System.*out*.println("本次操作所花费的时间：" + (end - start));  channel.close();  connection.close();  } } |

1. 消费者之中只需要对其做一个简单的控制即可

|  |
| --- |
| package com.kuhnwei.examples.consumer; import com.rabbitmq.client.\*; import java.io.IOException; */\*\*  \* 消息消费者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:39  \*/* public class MessageConsumer {  */\*\*  \* Exchange名称  \*/* private static final String *EXCHANGE\_NAME* = "kuhnwei.msg.topic";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  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("kuhnwei");  factory.setPassword("650901");  // 设置虚拟主机  factory.setVirtualHost("hello");  //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();  // 进行绑定处理  channel.queueBind(queueName, *EXCHANGE\_NAME*, "msg-key-1");  // 在RabbitMQ里面所有的消费者的信息是通过一个回调函数完成的  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 msg-key-1]" + message);  }  };  // 消息消费  channel.basicConsume(queueName, consumer);  } } |
| package com.kuhnwei.examples.consumer; import com.rabbitmq.client.\*; import java.io.IOException; */\*\*  \* 消息消费者  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 15:39  \*/* public class MessageConsumer {  */\*\*  \* Exchange名称  \*/* private static final String *EXCHANGE\_NAME* = "kuhnwei.msg.topic";  */\*\*  \* 消息服务地址  \*/* private static final String *HOST* = "192.168.127.129";  */\*\*  \* 消息服务端口号  \*/* private static final Integer *PORT* = 5672;  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("kuhnwei");  factory.setPassword("650901");  // 设置虚拟主机  factory.setVirtualHost("hello");  //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();  // 进行绑定处理  channel.queueBind(queueName, *EXCHANGE\_NAME*, "msg-key-2");  // 在RabbitMQ里面所有的消费者的信息是通过一个回调函数完成的  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 msg-key-2]" + 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主题信息，但是如果要想进行正确的处理，则一定需要有一个合适的rountingkey完成操作。

# Spring整合RabbitMQ组件

## 具体内容

所有的组件如果要想在开发中使用，必定需要与Spring整合，如果现在要想将rabbitmq与spring整合处理，则首先要考虑好要将整合包配置到项目之中。

|  |
| --- |
| <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> |

如果要想进行spring的配置（结合maven来讲）必须要考虑profile的情况所以需要建立一个rabbitmq.properties配置文件。

范例：在src/main/resources建立rabbitmq.properties配置文件

|  |
| --- |
| # RabbitMQ的连接主机地址 mq.rabbit.host=192.168.127.129 # RabbitMQ的连接端口号 mq.rabbit.port=5672 # RabbitMQ的虚拟主机名称 mq.rabbit.vhost=hello # RabbitMQ的exchange名称 mq.rabbit.exchange=kuhnwei.msg.spring # RabbitMQ的用户名 mq.rabbit.username=kuhnwei # RabbitMQ的用户密码 mq.rabbit.password=650901 |

随后生产者和消费者的程序类都要求读取此配置项。

### 建立消费者

1. Spring整合所有消息组件的处理模式都采用了一个同样的操作接口：MessageListener，也就是说现在如果要想进行消息的处理，则首先一定要有一个消息的处理类：

|  |
| --- |
| package com.kuhnwei.examples.ssm.listener; import org.springframework.amqp.core.Message; import org.springframework.amqp.core.MessageListener; */\*\*  \* 消息消费处理类  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 22:34  \*/* public class MessageConsumer implements MessageListener {  @Override  public void onMessage(Message message) {  System.*out*.println(">> [消费者]" + message);  } } |

1. 如果要想进行消费者的使用，则一定需要同故宫Spring配置文件整合，建立“src/main/resources/spring/spring-consumer.xml”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.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/rabbit  http://www.springframework.org/schema/rabbit/spring-rabbit.xsd">  <!-- 定义rabbitmq配置的相关属性信息 -->  <context:property-placeholder location="classpath:rabbitmq.properties"/>  <!-- 如果要想进行RabbitMQ的操作处理，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="${mq.rabbit.vhost}"/>  <!-- 所有的连接工厂要求被RabbitMQ所管理 -->  <rabbit:admin connection-factory="connectionFactory"/>  <!-- 创建一个队列信息 -->  <rabbit:queue id="myQueue" durable="true" auto-delete="true" exclusive="false" name="kuhnwei.queue.title"/>  <!-- 下面实现一个直连的操作模式 -->  <rabbit:direct-exchange id="mq-direct" name="${mq.rabbit.exchange}" durable="true" auto-delete="true">  <rabbit:bindings>  <!-- 绑定到指定的队列之中 -->  <rabbit:binding queue="myQueue" key="msg-key"/>  </rabbit:bindings>  </rabbit:direct-exchange>  <!-- 定义具体的消费者处理类 -->  <bean id="messageConsumer" class="com.kuhnwei.examples.ssm.listener.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. 编写一个测试程序类，进行消费者代码启动。

|  |
| --- |
| package com.kuhnwei.examples.ssm.rabbit; import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.test.context.ContextConfiguration; import org.springframework.test.context.junit4.SpringJUnit4ClassRunner; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 22:51  \*/* @ContextConfiguration(locations = {"classpath:spring/spring-consumer.xml"}) @RunWith(SpringJUnit4ClassRunner.class) public class TestConsumer {  @Test  public void testConsumer() {  try {  Thread.*sleep*(Long.*MAX\_VALUE*);  } catch (InterruptedException e) {  e.printStackTrace();  }  } } |

那么此时消费者已经成功运行。

定义生产者

如果要想进行生产者定义在Spring之中都会利用一个消息的处理模板进行消息的发送操作。

1. 建立一个发送消息的业务接口

|  |
| --- |
| package com.kuhnwei.examples.ssm.service; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/19 23:56  \*/* public interface IMessageService {  void send(String msg); } |

1. 实现发送消息的业务类

|  |
| --- |
| package com.kuhnwei.examples.ssm.service.impl; import com.kuhnwei.examples.ssm.service.IMessageService; import org.springframework.amqp.core.AmqpTemplate; import org.springframework.stereotype.Service; import javax.annotation.Resource; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/20 0:02  \*/* @Service public class MessageServiceImpl implements IMessageService{  @Resource  private AmqpTemplate amqpTemplate;  @Override  public void send(String msg) {  this.amqpTemplate.convertAndSend("msg-key", msg);  } } |

1. 通过spring-consumer.xml文件复制一份spring-producer.xml配置文件，作为消息生产者配置。

|  |
| --- |
| <?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.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/rabbit  http://www.springframework.org/schema/rabbit/spring-rabbit.xsd">  <!-- 定义rabbitmq配置的相关属性信息 -->  <context:property-placeholder location="classpath:rabbitmq.properties" ignore-unresolvable="true"/>  <!-- 如果要想进行RabbitMQ的操作处理，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="${mq.rabbit.vhost}"/>  <!-- 所有的连接工厂要求被RabbitMQ所管理 -->  <rabbit:admin connection-factory="connectionFactory"/>  <!-- 创建一个队列信息 -->  <rabbit:queue id="myQueue" durable="true" auto-delete="true" exclusive="false" name="kuhnwei.queue.title"/>  <!-- 下面实现一个直连的操作模式 -->  <rabbit:direct-exchange id="mq-direct" name="${mq.rabbit.exchange}" durable="true" auto-delete="true">  <rabbit:bindings>  <!-- 绑定到指定的队列之中 -->  <rabbit:binding queue="myQueue" key="msg-key"/>  </rabbit:bindings>  </rabbit:direct-exchange>  <!-- 所有整合的消息系统都会有一个模板 -->  <rabbit:template id="amqpTemplate" exchange="${mq.rabbit.exchange}" connection-factory="connectionFactory"/> </beans> |

1. 编写测试程序，测试消息发送：

|  |
| --- |
| package com.kuhnwei.examples.ssm.rabbit; import com.kuhnwei.examples.ssm.service.IMessageService; import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.test.context.ContextConfiguration; import org.springframework.test.context.junit4.SpringJUnit4ClassRunner; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/20 0:06  \*/* @ContextConfiguration(locations = {"classpath:spring/spring-\*.xml"}) @RunWith(SpringJUnit4ClassRunner.class) public class TestProducer {  @Autowired  private IMessageService messageService;  @Test  public void testSend() {  this.messageService.send("msg - " + System.*currentTimeMillis*());  } } |

那么以后在实际的项目之中就可以采用此种模式整合所有的消息服务组件。

# 搭建RabbitMQ集群

## 具体内容

RabbitMQ的集群搭建异常简单，但是这个集群的设计分为两类：普通集群、镜像队列，其中镜像队列必须要求会配置。而且RabbitMQ的集群本身不带有所谓的HA机制以及负载均衡机制。你如果要想在实际的更加复杂的运行之中去使用RabbitMQ集群服务，那么必须要求再结合负载均衡组件（HAProxy、LVS）以及Keepalived（机制）。

那么本次如果要想使用RabbitMQ的集群操作，预计一共采用三台主机。