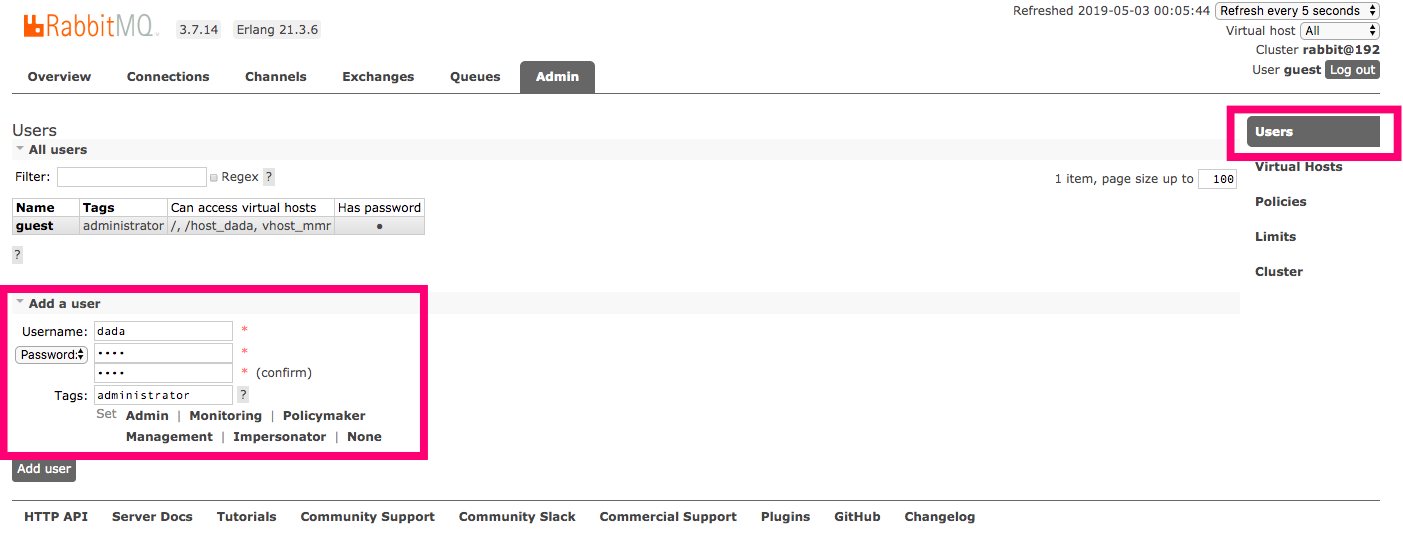
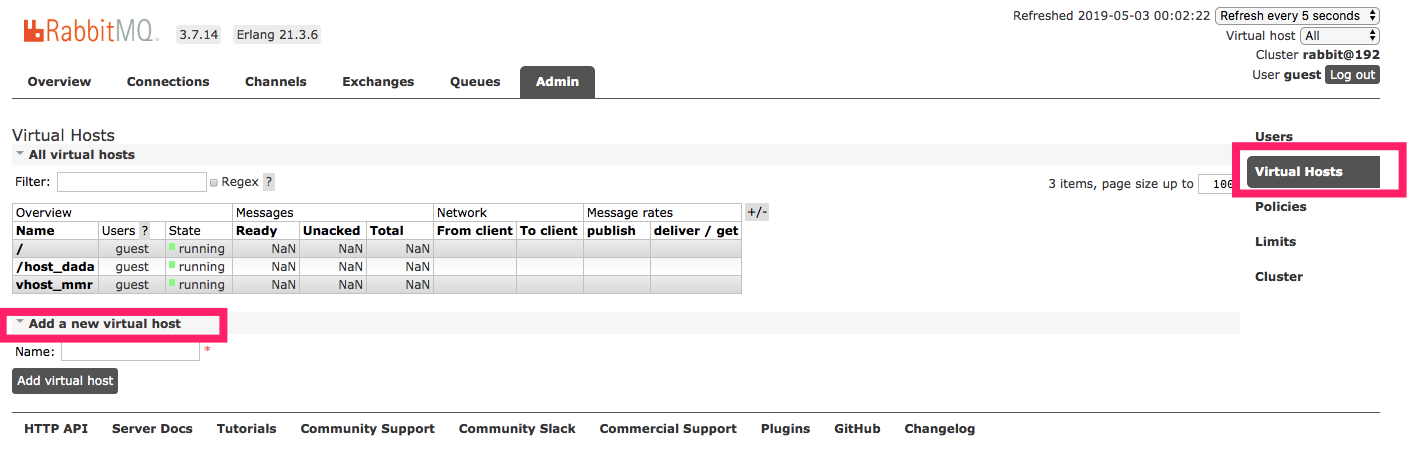
# 1、控制台介绍

## 1.1 添加用户:

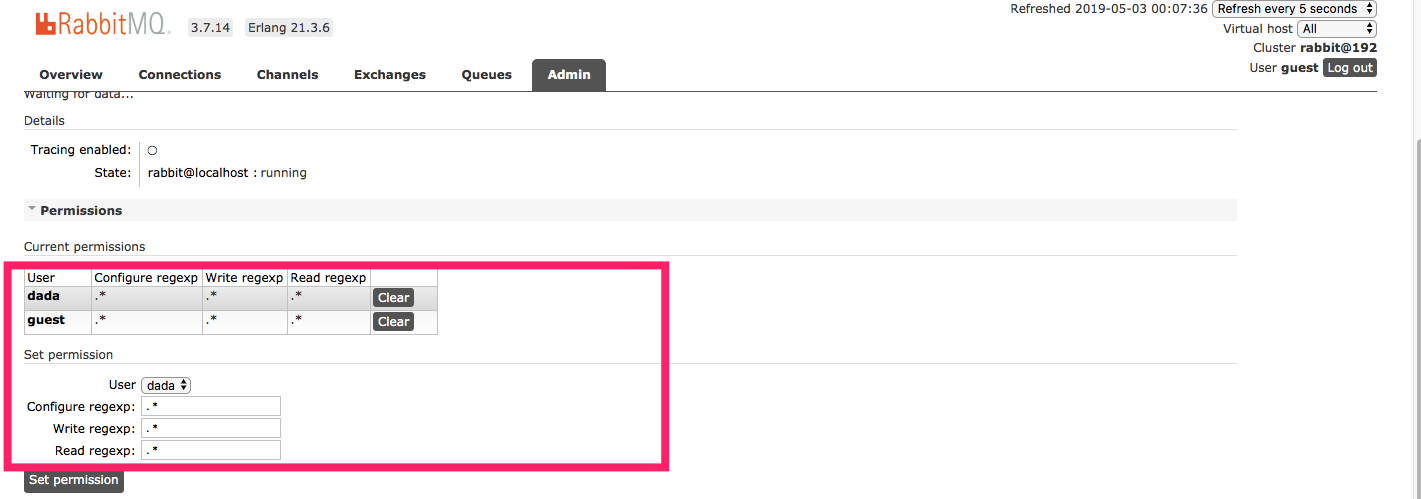


## 1.2 virtual hosts:管理

（1）irtual hosts就相当于mysql的DB添加一个数据库：



（2）设置权限



# 2、简单队列

IMG_256

P：消息的生产者

红色：消息队列

C：消费者

三个对象： 生产者 队列(RabbitMq) 消费者

## 2.1 maven依赖

|  |
| --- |
| ***<?*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.dada***</*groupId*>  <*artifactId*>***RabbitMQ***</*artifactId*>  <*version*>***1.0-SNAPSHOT***</*version*>   <*dependencies*>   <*dependency*>  <*groupId*>***com.rabbitmq***</*groupId*>  <*artifactId*>***amqp-client***</*artifactId*>  <*version*>***5.5.2***</*version*>  </*dependency*>   <*dependency*>  <*groupId*>***org.slf4j***</*groupId*>  <*artifactId*>***slf4j-api***</*artifactId*>  <*version*>***1.7.25***</*version*>  </*dependency*>   <*dependency*>  <*groupId*>***log4j***</*groupId*>  <*artifactId*>***log4j***</*artifactId*>  <*version*>***1.2.17***</*version*>  </*dependency*>   <*dependency*>  <*groupId*>***junit***</*groupId*>  <*artifactId*>***junit***</*artifactId*>  <*version*>***4.12***</*version*>  </*dependency*>  </*dependencies*>  </*project*>*** |

## 2.2 获取MQ的连接

|  |
| --- |
| */\*\*  \* 连接工厂  \*/* **public class** ConnectionUtil ***{*** */\*\*  \* 获取Mq的连接  \** ***@return*** *\** ***@throws*** *IOException  \** ***@throws*** *TimeoutException  \*/* **public static** Connection getConnection***()* throws** IOException, TimeoutException ***{*** *// 定义一个连接工厂* ConnectionFactory connectionFactory = **new** ConnectionFactory***()***;   *// 设置服务地址* connectionFactory.setHost***(*"127.0.0.1"*)***;   *// 设置端口号* connectionFactory.setPort***(***5672***)***;   *// 设置数据库 Vhost* connectionFactory.setVirtualHost***(*"/vhostnew"*)***;   *// 设置用户名密码* connectionFactory.setUsername***(*"dada"*)***;  connectionFactory.setPassword***(*"dada"*)***;   *// 获取连接* Connection connection = connectionFactory.newConnection***()***;   **return** connection;  ***} }*** |

## 2.3 生产者

|  |
| --- |
| */\*\*  \* 生产者  \*/* **public class** Send ***{* private static final** String ***Queue\_name*** = **"test\_simple\_queque"**;   **public static void** main***(***String***[]*** args***)* throws** Exception ***{*** *// 获取一个连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 从连接中获取一个通道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 发送的消息* String msg= **"hello Simple"**;   *// 发送* channel.basicPublish***(*""**,***Queue\_name***,**null**,msg.getBytes***())***;   System.***out***.println***(*"----send msg:"** + msg***)***;   *//关闭连接* channel.close***()***;  connection.close***()***;   ***} }*** |

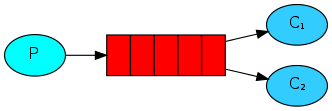
## 2.4 消费者

|  |
| --- |
| */\*\*  \* 消费者消费消息  \*/* **public class** Recv ***{* private static final** String ***Queue\_name*** = **"test\_simple\_queque"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 创建频道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 声明监听对象* DefaultConsumer consumer = **new** DefaultConsumer***(***channel***) {*** */\*\*  \* 一旦有消息进入队列 就会触发这个方法  \** ***@param consumerTag*** *\** ***@param envelope*** *\** ***@param properties*** *\** ***@param body*** *数据  \** ***@throws*** *IOException  \*/* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body***)***;  System.***out***.println***(***msg***)***;  ***}  }***;   *// 监听队列* channel.basicConsume***(Queue\_name***,consumer***)***;    ***} }*** |

## 2.5 简单队列的不足

耦合性高：生产者一一对应消费者（如果有多个消费者消费队列中的消息，这时候就不行了）队列名变更，这时候得同时变更

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



一个生产者对应多个消费者

为什么会出现工作队列?

Simple队列是一一对应的，而且我们实际开发，生产者发送消息是毫不费力的，而且一般消费者是跟业务相结合的。消费者接收到消息之后就需要处理，可能需要花费时间，这时候队列就会积压很多消息。

## 3.1 默认采用轮询

### **3.1.1生产者：**

|  |
| --- |
| **package** com.dada.work;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.Channel; **import** com.rabbitmq.client.Connection;  **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  */\*\*  \* 工作队列  \*/* **public class** Send ***{* private static final** String ***Queue\_name*** = **"test\_work\_queque"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException, InterruptedException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 获取channel* Channel channel = connection.createChannel***()***;   *// 声明一个队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 发送50条消息* **for *(*int** i=0;i<50 ; i++***){*** String msg= **" hello"**+i;  System.***out***.println***(*"send======="**+msg***)***;  channel.basicPublish***(*""**,***Queue\_name***,**null**,msg.getBytes***())***;  Thread.*sleep****(***i\*20***)***;  ***}*** channel.close***()***;  connection.close***()***;  ***} }*** |

### **3.1.2消费者1**

|  |
| --- |
| **package** com.dada.work;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.\*;  **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  **public class** Recv1 ***{* private static final** String ***Queue\_name*** = **"test\_work\_queque"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** Connection connection = ConnectionUtil.*getConnection****()***;  Channel channel = connection.createChannel***()***;  channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[1] Recv msg :::: z"**+msg***)***;  **try *{*** Thread.*sleep****(***20000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** System.***out***.println***(*"`1 [down]"*)***;  ***}  }  }***;   **boolean** autoAck= **true**;  channel.basicConsume***(Queue\_name***,autoAck,consumer***)***;  ***} }*** |

### **3.1.3 消费者2**

|  |
| --- |
| **package** com.dada.work;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.\*;  **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  **public class** Recv2 ***{* private static final** String ***Queue\_name*** = **"test\_work\_queque"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** Connection connection = ConnectionUtil.*getConnection****()***;  Channel channel = connection.createChannel***()***;  channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[2] Recv msg :::: z"**+msg***)***;  **try *{*** Thread.*sleep****(***10000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** System.***out***.println***(*"`2 [down]"*)***;  ***}  }  }***;   **boolean** autoAck= **true**;  channel.basicConsume***(Queue\_name***,autoAck,consumer***)***;  ***} }*** |

## 3.2 公平分发

队列一次只发给消费者一个消息，当前消费者处理完毕之后会发rabbitMq一个响应之后rabbitmq才会给分发下一个

### **3.2.1 消费者**

|  |
| --- |
| **package** com.dada.workfair;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.\*;  **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  **public class** Recv1 ***{* private static final** String ***Queue\_name*** = **"test\_work\_quequefair"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** Connection connection = ConnectionUtil.*getConnection****()***;  Channel channel = connection.createChannel***()***;  channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 设置一次只接收一个队列的消息* channel.basicQos***(***1***)***;   *// 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[1] Recv msg :::: z"**+msg***)***;  **try *{*** Thread.*sleep****(***20000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** *// 手动应答rabiitMQ 队列*  channel.basicAck**(**envelope.getDeliveryTag**()**,**false)**;  System.***out***.println***(*"`1 [down]"*)***;  ***}  }  }***;   *// 自动应答fasle* **boolean** autoAck= **false**;  channel.basicConsume***(Queue\_name***,autoAck,consumer***)***;  ***} }*** |

### **3.2.2 生产者**

|  |
| --- |
| **package** com.dada.workfair;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.Channel; **import** com.rabbitmq.client.Connection; **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  */\*\*  \* 工作队列  \*/* **public class** Send ***{* private static final** String ***Queue\_name*** = **"test\_work\_quequefair"**;   **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException, InterruptedException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 获取channel* Channel channel = connection.createChannel***()***;   *// 声明一个队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   */\*\*  \* 每个消费者发送确认消息之前 消息队列不发送下一个消息到消费者，一次只处理一个消息  \* 限制发送给同一个消费者，不超过一个消息  \*/* **int** prefetchCount=1;  channel.basicQos***(***prefetchCount***)***;   *// 发送50条消息* **for *(*int** i=0;i<50 ; i++***){*** String msg= **" hello"**+i;  System.***out***.println***(*"send======="**+msg***)***;  channel.basicPublish***(*""**,***Queue\_name***,**null**,msg.getBytes***())***;  Thread.*sleep****(***i\*20***)***;  ***}*** channel.close***()***;  connection.close***()***;  ***} }*** |

## 3.3 消息应达与消息持久化

### **3.3.1 消息应达**

**boolean** autoAck= **true;**  
channel.basicConsume***(Queue\_name***,autoAck,consumer***)***;

**（1）boolean autoAck= true;自动确认模式，一旦rabbitMq将消息分发给消费者就会从内存中删除，这种情况下：如果杀死正在执行的消费者，就会丢失正在处理的消息。**

**（2）boolean autoAck= fasle;手动模式，如果有一个消费者挂掉了，就会交付给其他消费者，rabbbitMQ支持消息应答，告诉rabbitMQ这个消息已经处理完成可以删除了，然后rabbitMQ就会删除内存中的消息。**

**（3）消息应答默认是打开的false**

**（4）如果rabbitMq挂了，我们的消息仍然会丢失！！！！**

### **3.3.2 持久化**

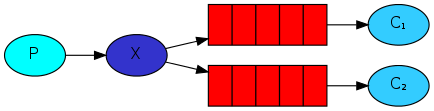
channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;

Queue.DeclareOk queueDeclare***(***String queue, **boolean** durable, **boolean** exclusive, **boolean** autoDelete,  
 Map***<***String, Object***>*** arguments***)* throws** IOException;

**（1）Durable：持久化**

**（2）已经定义的队列不可更改**

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



**（1）解读：**

**1，一个生产者，多个消费者**

**2，每一个消费者都有自己的一个队列**

**3，消费者没有直接把消息发送到队列，而是发送到了交换机(转发器上面 Exchange)**

**4，每个队列都要绑定到交换机上**

**5，生产者发送消息，经过交换机，到达队列，就能实现一个消息被多个消费者消费**

**（2）例子：**

**注册-> 邮件->短信**

**（3）交换机:**



## 4.1 生产者:

|  |
| --- |
| **public class** Send ***{* private static final** String ***exchangeName*** = **"test\_exchange\_fanout"**;   **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** Connection connection = ConnectionUtil.*getConnection****()***;   Channel channel = connection.createChannel***()***;   *// 声明交换机* channel.exchangeDeclare***(exchangeName***,**"fanout"*)***;*// 分发   // 发送消息* String msg = **"hello ps"**;   channel.basicPublish***(exchangeName***,**""**,**null**,msg.getBytes***())***;   System.***out***.println***(*"sned "**+msg ***)***;   channel.close***()***;   connection.close***()***;   ***} }*** |

**消息哪里去了？丢失了!!!!!!因为交换机没有存储的能力,在rabbitmq中只有队列有存储能力，因为这个时候还没有队列绑定到交换机所以数据会丢失。**

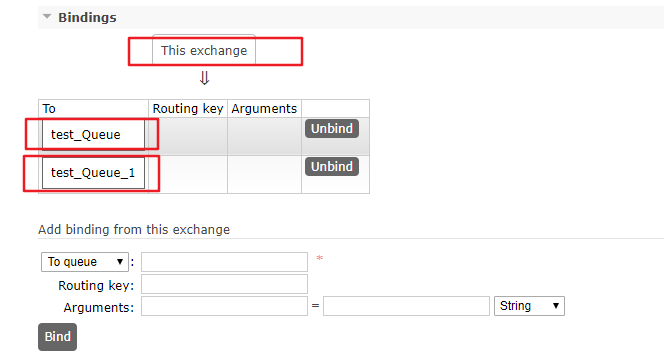
## 4.2 消费者1

|  |
| --- |
| **package** com.dada.pb;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.\*;  **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  **public class** Recv1 ***{* private static final** String ***QUEUE\_NAME***=**"test\_Queue"**;   **private static final** String ***exchangeName*** = **"test\_exchange\_fanout"**;   **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** Connection connection = ConnectionUtil.*getConnection****()***;   Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(QUEUE\_NAME***,**false**,**false**,**false**,**null*)***;   *// 绑定到交换机或者转发器* channel.queueBind***(QUEUE\_NAME***,***exchangeName***,**""*)***;   channel.basicQos***(***1***)***;   *// 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[2] Recv msg :::: z"**+msg***)***;  **try *{*** Thread.*sleep****(***10000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** *// 手动应答rabiitMQ 队列* channel.basicAck***(***envelope.getDeliveryTag***()***,**false*)***;  System.***out***.println***(*"`2 [down]"*)***;  ***}  }  }***;   **boolean** autoAck= **false**;  channel.basicConsume***(QUEUE\_NAME***,autoAck,consumer***)***;  ***} }*** |

## 4.3 消费者2

|  |
| --- |
| **package** com.dada.pb;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.\*;  **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  **public class** Recv2 ***{* private static final** String ***QUEUE\_NAME***=**"test\_Queue\_1"**;   **private static final** String ***exchangeName*** = **"test\_exchange\_fanout"**;   **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** Connection connection = ConnectionUtil.*getConnection****()***;   Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(QUEUE\_NAME***,**false**,**false**,**false**,**null*)***;   *// 绑定到交换机或者转发器* channel.queueBind***(QUEUE\_NAME***,***exchangeName***,**""*)***;   channel.basicQos***(***1***)***;   *// 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[1] Recv msg :::: z"**+msg***)***;  **try *{*** Thread.*sleep****(***10000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** *// 手动应答rabiitMQ 队列* channel.basicAck***(***envelope.getDeliveryTag***()***,**false*)***;  System.***out***.println***(*"`1 [down]"*)***;  ***}  }  }***;   **boolean** autoAck= **false**;  channel.basicConsume***(QUEUE\_NAME***,autoAck,consumer***)***;  ***} }*** |

## 4.4 管理界面

\

# 5、Exchange（交换机 转发器）

一方面接收生产者的消息，另一方面是向队列推送消息

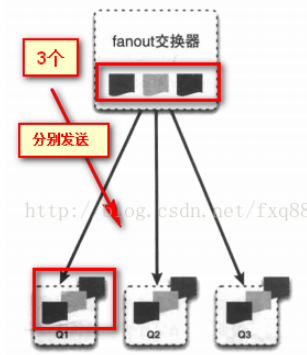
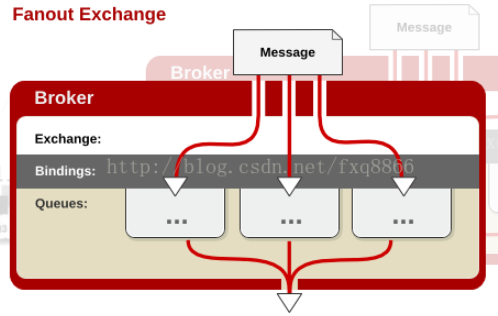
## 5.1 匿名转发

匿名转发””

|  |
| --- |
| *// 发送* channel.basicPublish***(*""**,***Queue\_name***,**null**,msg.getBytes***())***; |

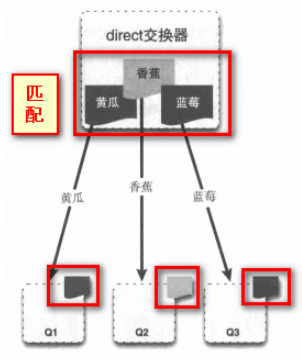
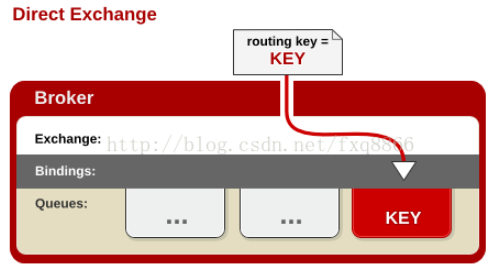
## 5.2 Fanout

不处理路由键只要与它绑定的队列都能收到消息

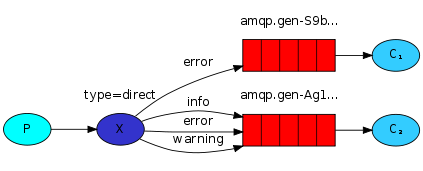


## 5.3 Direct

Direct(处理路由键)



### 5.3.1 路由模式



### **5.3.2 生产者**

|  |
| --- |
| **public class** Send ***{* private static final** String ***EXCHANGE\_NAME***=**"TESTECXCHANGE\_DIRECT"**;   **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 创建频道* Channel channel = connection.createChannel***()***;   *// 声明Exchangge* channel.exchangeDeclare***(EXCHANGE\_NAME***,**"direct"*)***;   *// 消息* String msg=**"hello direct"**;   *// 路由key // String routerkey="error";* String routerkey=**"info"**;   *// 发送消息* channel.basicPublish***(EXCHANGE\_NAME***,routerkey,**null**,msg.getBytes***())***;   *// 关闭资源* channel.close***()***;  connection.close***()***;  ***} }*** |

### **5.3.3 消费者1**

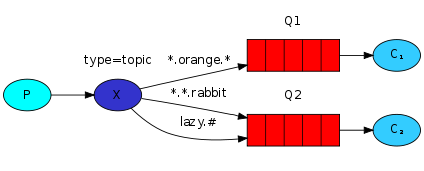
|  |
| --- |
| **public class** Recv1 ***{* private static final** String ***EXCHANGE\_NAME***=**"TESTECXCHANGE\_DIRECT"**;   **private static final** String ***Queue\_name*** = **"test\_queque\_direct1"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 创建频道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 设置手动提交* channel.basicQos***(***1***)***;   *//绑定队列* channel.queueBind***(Queue\_name***,***EXCHANGE\_NAME***,**"error"*)***;  *// 如需多个绑定 // channel.queueBind(Queue\_name,EXCHANGE\_NAME,"error"); // channel.queueBind(Queue\_name,EXCHANGE\_NAME,"error");   // 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[1] Recv msg :::: "**+msg***)***;  **try *{*** Thread.*sleep****(***10000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** *// 手动应答rabiitMQ 队列* channel.basicAck***(***envelope.getDeliveryTag***()***,**false*)***;  System.***out***.println***(*"`【1】 down"*)***;  ***}  }  }***;   **boolean** autoAck= **false**;  channel.basicConsume***(Queue\_name***,autoAck,consumer***)***;  ***} }*** |

### **5.3.4 消费者2**

|  |
| --- |
| **public class** Recv2 ***{* private static final** String ***EXCHANGE\_NAME***=**"TESTECXCHANGE\_DIRECT"**;   **private static final** String ***Queue\_name*** = **"test\_queque\_direct1"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 创建频道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 设置手动提交* channel.basicQos***(***1***)***;   *//绑定队列* channel.queueBind***(Queue\_name***,***EXCHANGE\_NAME***,**"error"*)***;  *//绑定队列* channel.queueBind***(Queue\_name***,***EXCHANGE\_NAME***,**"info"*)***;  *//绑定队列* channel.queueBind***(Queue\_name***,***EXCHANGE\_NAME***,**"warning"*)***;  *// 如需多个绑定 // channel.queueBind(Queue\_name,EXCHANGE\_NAME,"error"); // channel.queueBind(Queue\_name,EXCHANGE\_NAME,"error");   // 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[2] Recv msg :::: "**+msg***)***;  **try *{*** Thread.*sleep****(***10000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** *// 手动应答rabiitMQ 队列* channel.basicAck***(***envelope.getDeliveryTag***()***,**false*)***;  System.***out***.println***(*"`【2】 down"*)***;  ***}  }  }***;   **boolean** autoAck= **false**;  channel.basicConsume***(Queue\_name***,autoAck,consumer***)***;  ***} }*** |

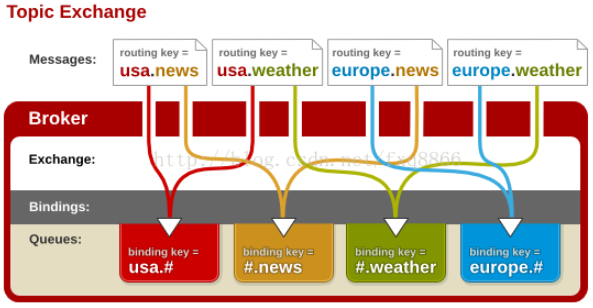
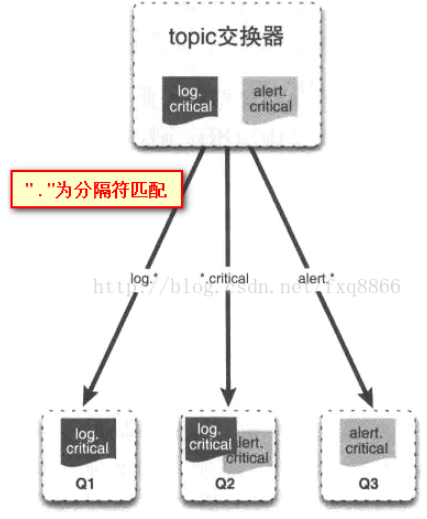
## 5.4 [Topics](https://www.rabbitmq.com/tutorials/tutorial-five-python.html)

### **5.4.1 模型:**



### **5.4.2 Topic Exchange**

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



### **5.4.3 生产者**

例子：商品:发布、修改、删除、查询。。。。。

|  |
| --- |
| **public class** Send ***{* private static final** String ***EXCHANGE\_NAME***=**"TESTECXCHANGE\_Topic"**;   **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 创建频道* Channel channel = connection.createChannel***()***;   *// 声明Exchangge* channel.exchangeDeclare***(EXCHANGE\_NAME***,**"topic"*)***;   *// 消息* String msg=**"商品。。。。。。。。"**;   *// 路由key* String routerkey=**"goods.delete"**;   *// 发送消息* channel.basicPublish***(EXCHANGE\_NAME***,routerkey,**null**,msg.getBytes***())***;   *//打印* System.***out***.println***(*"msg："**+routerkey***)***;   *// 关闭资源* channel.close***()***;  connection.close***()***;  ***} }*** |

### **5.4.4 消费者1**

例子：商品:发布、修改、删除、查询。。。。

|  |
| --- |
| **public class** Recv1 ***{* private static final** String ***EXCHANGE\_NAME***=**"TESTECXCHANGE\_Topic"**;   **private static final** String ***Queue\_name*** = **"test\_queque\_topic1"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 创建频道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 设置手动提交* channel.basicQos***(***1***)***;   *//绑定队列* channel.queueBind***(Queue\_name***,***EXCHANGE\_NAME***,**"goods.add"*)***;  *// 如需多个绑定 // channel.queueBind(Queue\_name,EXCHANGE\_NAME,"error"); // channel.queueBind(Queue\_name,EXCHANGE\_NAME,"error");   // 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[1] Recv msg :::: "**+msg***)***;  **try *{*** Thread.*sleep****(***10000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** *// 手动应答rabiitMQ 队列* channel.basicAck***(***envelope.getDeliveryTag***()***,**false*)***;  System.***out***.println***(*"`【1】 down"*)***;  ***}  }  }***;   **boolean** autoAck= **false**;  channel.basicConsume***(Queue\_name***,autoAck,consumer***)***;  ***} }*** |

### **5.4.5 消费者2**

例子：商品:发布、修改、删除、查询。。。。。

|  |
| --- |
| **public class** Recv2 ***{* private static final** String ***EXCHANGE\_NAME***=**"TESTECXCHANGE\_Topic"**;   **private static final** String ***Queue\_name*** = **"test\_queque\_topic2"**;    **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 创建频道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 设置手动提交* channel.basicQos***(***1***)***;   *//绑定队列* channel.queueBind***(Queue\_name***,***EXCHANGE\_NAME***,**"goods.#"*)***;    *// 定义一个消费者* Consumer consumer = **new** DefaultConsumer***(***channel***) {*** *// 消息到达触发这个方法* @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** String msg = **new** String***(***body,**"utf-8"*)***;  System.***out***.println***(*"[2] Recv msg :::: "**+msg***)***;  **try *{*** Thread.*sleep****(***10000***)***;  ***}* catch *(***InterruptedException e***) {*** e.printStackTrace***()***;  ***}*finally *{*** *// 手动应答rabiitMQ 队列* channel.basicAck***(***envelope.getDeliveryTag***()***,**false*)***;  System.***out***.println***(*"`【2】 down"*)***;  ***}  }  }***;   **boolean** autoAck= **false**;  channel.basicConsume***(Queue\_name***,autoAck,consumer***)***;  ***} }*** |

# RabbitMQ的消息确认机制 (事务+confirm)

生产者消息的确认：

1. 在rabbitMQ中，我们可以通过持久化数据来解决RabbitMQ服务器异常的数据丢失问题
2. 生产者将消息发送出去之后消息到底有没有到达rabbitmq服务器，默认情况是不知道的
   1. AMQP协议实现了事务机制
   2. Confirm

## 6.1 事务机制

TxSelect txCommit txRollback

TxSelect：用户当前将channel设置成transation模式

txCommit ：用户提交事务

txRollback ：用于回滚事务

### **6.1.1 生产者**

|  |
| --- |
| **package** com.dada.tx;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.Channel; **import** com.rabbitmq.client.Connection;  **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  **public class** TxSend ***{* private static final** String ***QUEUE\_NAME*** = **"TXQUUE"**;   **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取一个连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 从连接中获取一个通道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(QUEUE\_NAME***, **false**, **false**, **false**, **null*)***;   String msg = **"Hello TX"**;   *// 开启事务模式* channel.txSelect***()***;  **try *{*** channel.basicPublish***(*""**, ***QUEUE\_NAME***, **null**, msg.getBytes***())***;  *// 测试异常回滚 // int x = 1/0;* channel.txCommit***()***;  ***}*catch *(***Exception e***){*** channel.txRollback***()***;  System.***out***.println***(*"QUEUE Rollback"*)***;  ***}*** channel.close***()***;  connection.close***()***;  ***} }*** |

### **6.1.2 消费者**

|  |
| --- |
| **package** com.dada.tx;  **import** com.dada.util.ConnectionUtil; **import** com.rabbitmq.client.\*; **import** java.io.IOException; **import** java.util.concurrent.TimeoutException;  **public class** TxRecv ***{* private static final** String ***QUEUE\_NAME*** = **"TXQUUE"**;   **public static void** main***(***String***[]*** args***)* throws** IOException, TimeoutException ***{*** *// 获取一个连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 从连接中获取一个通道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(QUEUE\_NAME***, **false**, **false**, **false**, **null*)***;   *// 消费者* channel.basicConsume***(QUEUE\_NAME***, **true**, **new** DefaultConsumer***(***channel***) {*** @Override  **public void** handleDelivery***(***String consumerTag, Envelope envelope, AMQP.BasicProperties properties, **byte*[]*** body***)* throws** IOException ***{*** System.***out***.println***(*new** String***(***body***))***;  ***}  })***;     ***} }*** |

## 6.2 Confirm模式

1. 生产者端confirm模式的实现原理

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

       confirm模式最大的好处在于他是异步的，一旦发布一条消息，生产者应用程序就可以在等信道返回确认的同时继续发送下一条消息，当消息最终得到确认之后，生产者应用便可以通过回调方法来处理该确认消息，如果RabbitMQ因为自身内部错误导致消息丢失，就会发送一条nack消息，生产者应用程序同样可以在回调方法中处理该nack消息；

1. 开启Confirm模式 Channel.confirmSelect()

### ****6.2.1编程模式****

(1)客户端实现生产者confirm有三种编程方式：

1.普通confirm模式：每发送一条消息后，调用waitForConfirms()方法，等待服务器端confirm。实际上是一种串行confirm了。

2.批量confirm模式：每发送一批消息后，调用waitForConfirms()方法，等待服务器端confirm。

3.异步confirm模式：提供一个回调方法，服务端confirm了一条或者多条消息后Client端会回调这个方法。

#### ****6.2.1.1 单条****

|  |
| --- |
| **public class** Send1 ***{* private static final** String ***Queue\_name*** = **"test\_queue\_confirm\_1"**;   **public static void** main***(***String***[]*** args***)* throws** Exception ***{*** *// 获取一个连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 从连接中获取一个通道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 设置为confirm模式* channel.confirmSelect***()***;   *// 发送的消息* String msg= **"hello confirm"**;   *// 发送* channel.basicPublish***(*""**,***Queue\_name***,**null**,msg.getBytes***())***;   **if *(***!channel.waitForConfirms***()){*** System.***out***.println***(*"send fail"*)***;  ***}*else *{*** System.***out***.println***(*"----send msg:"** + msg***)***;  ***}*** *//关闭连接* channel.close***()***;  connection.close***()***;   ***} }*** |

#### ****6.2.1.2 批量****

|  |
| --- |
| **public class** Send2 ***{* private static final** String ***Queue\_name*** = **"test\_queue\_confirm\_1"**;   **public static void** main***(***String***[]*** args***)* throws** Exception ***{*** *// 获取一个连接* Connection connection = ConnectionUtil.*getConnection****()***;   *// 从连接中获取一个通道* Channel channel = connection.createChannel***()***;   *// 声明队列* channel.queueDeclare***(Queue\_name***,**false**,**false**,**false**,**null*)***;   *// 设置为confirm模式* channel.confirmSelect***()***;   *// 发送的消息* String msg= **"hello confirm"**;   **for *(*int** i = 0; i < 50; i++***) {*** *// 批量发送* channel.basicPublish***(*""**,***Queue\_name***,**null**,msg.getBytes***())***;  ***}*** *// 发完之后确认* **if *(***!channel.waitForConfirms***()){*** System.***out***.println***(*"send fail"*)***;  ***}*else *{*** System.***out***.println***(*"----send msg:"** + msg***)***;  ***}*** *//关闭连接* channel.close***()***;  connection.close***()***;   ***} }*** |

#### ****6.2.1.3 异步****

Channel对象提供的ConfirmListener()回调方法只包含deliveryTag（当前Chanel发出的消息序号），我们需要自己为每一个Channel维护一个unconfirm的消息序号集合，每publish一条数据，集合中元素加1，每回调一次handleAck方法，unconfirm集合删掉相应的一条（multiple=false）或多条（multiple=true）记录。从程序运行效率上看，这个unconfirm集合最好采用有序集合SortedSet存储结构。实际上，SDK中的waitForConfirms()方法也是通过SortedSet维护消息序号的。