

RabbitMQ 文档

一 简介

MQ全称为Message Queue, 消息队列 (MQ) 是一种应用程序对应用程序的通信方法。应用程序通过读写出入队列的消息 (针对应用程序的数据) 来通信, 而无需专用连接来链接它们。消息传递指的是程序之间通过在消息中发送数据进行通信, 而不是通过直接调用彼此来通信, 直接调用通常是用于诸如远程过程调用的技术。排队指的是应用程序通过 队列来通信。队列的使用除去了接收和发送应用程序同时执行的要求。其中较为成熟的MQ产品有IBM WEBSHERE MQ等等。

RabbitMQ是一个在AMQP基础上完成的, 可复用的企业消息系统。他遵循Mozilla Public License开源协议。

AMQP, 即Advanced Message Queuing Protocol,一个提供统一消息服务的应用层标准高级消息队列协议,是应用层协议的一个开放标准,为面向消息的中间件设计。基于此协议的客户端与消息中间件可传递消息, 并不受客户端/中间件不同产品, 不同的开发语言等条件的限制。Erlang中的实现有RabbitMQ等。

二 安装 RabbitMQ

此环境使用 centos7

1. 下载 erlang solution

```
wget https://packages.erlang-solutions.com/erlang-solutions-1.0-1.noarch.rpm
```

2. 安装

```
rpm -Uvh erlang-solutions-1.0-1.noarch.rpm  
yum install epel-release  
yum install erlang
```

3. 下载 rabbit

```
wget http://www.rabbitmq.com/releases/rabbitmq-server/v3.6.6/rabbitmq-server-3.6.6-1.el7.noarch.rpm
```

4. 安装 rabbit

```
yum install rabbitmq-server-3.6.6-1.el7.noarch.rpm
```

5. 开启 允许远程访问(非必须)

```
vi /etc/rabbitmq/rabbitmq.config  
###添加一下内容  
[{{rabbit, [{{loopback_users, []}}}}].
```

6. 开启 web 端管理访问(非必须,如果要开启,需要先开启允许远程访问)

```
rabbitmq-plugins enable rabbitmq_management
```

7. 安装消息延迟插件

```
cd /usr/lib/rabbitmq/lib/rabbitmq_server-3.6.6/plugins  
  
wget https://dl.bintray.com/rabbitmq/community-  
plugins/rabbitmq_delayed_message_exchange-0.0.1.ez  
  
rabbitmq-plugins enable rabbitmq_delayed_message_exchange
```

8. 放行端口

```
firewall-cmd --add-port=15672/tcp --permanent  
  
firewall-cmd --add-port=5672/tcp --permanent
```

9. 启动

```
service rabbitmq-server start  
###查看状态  
service rabbitmq-server status  
###重启  
service rabbitmq-server restart  
#停止  
service rabbitmq-server stop
```

三 添加用户

3.1 账号级别

1. 超级管理员administrator, 可以登录控制台, 查看所有信息, 可以对用户和策略进行操作
2. 监控者monitoring, 可以登录控制台, 可以查看节点的相关信息, 比如进程数, 内存磁盘使用情况
3. 策略制定者policymaker, 可以登录控制台, 制定策略, 但是无法查看节点信息
4. 普通管理员 management 仅能登录控制台
5. 其他, 无法登录控制台, 一般指的是提供者 and 消费者

3.2 添加账号

3.2.1 命令模式

```
rabbitmqctl add_user luke luke #添加账号 luke 密码是 luke
```

```
rabbitmqctl set_user_tags luke administrator #设置 luke 为administrator级别
```

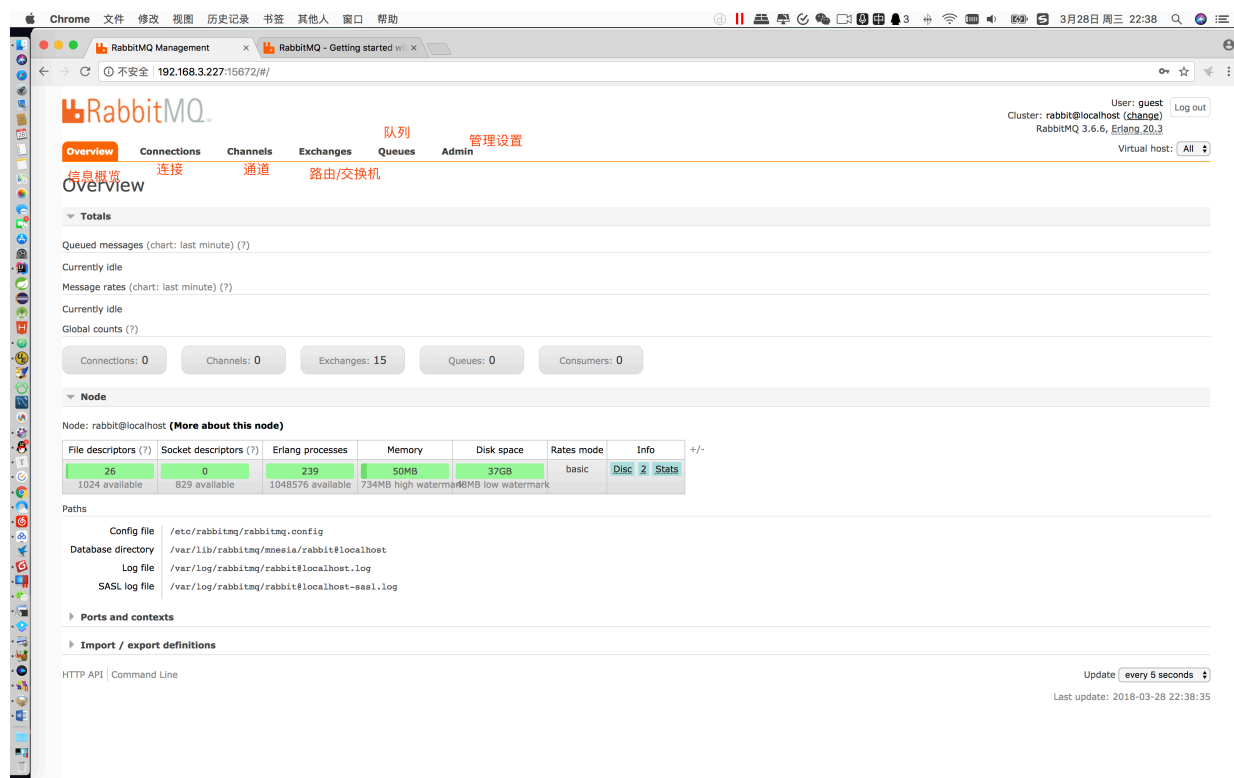
3.2.2 web 方式

此方式需要开启 web 访问

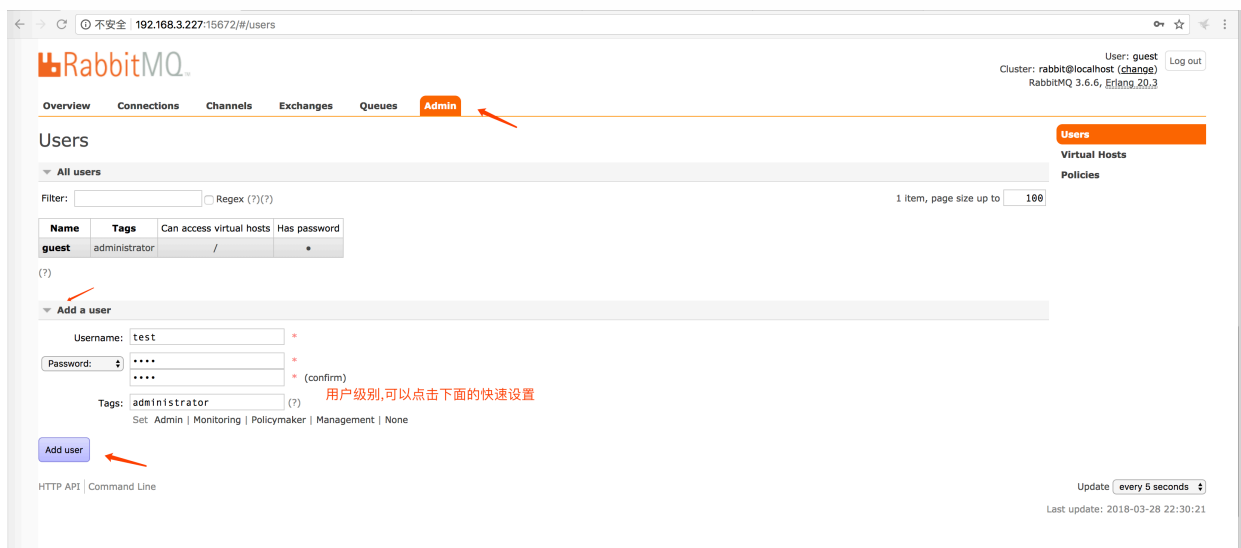
3.2.2.1 访问web

<http://192.168.3.227:15672/> 此处 ip 是本人 ip, 实际中请以实际 ip 为准

使用 guest guest 登录 guest 具有最高权限

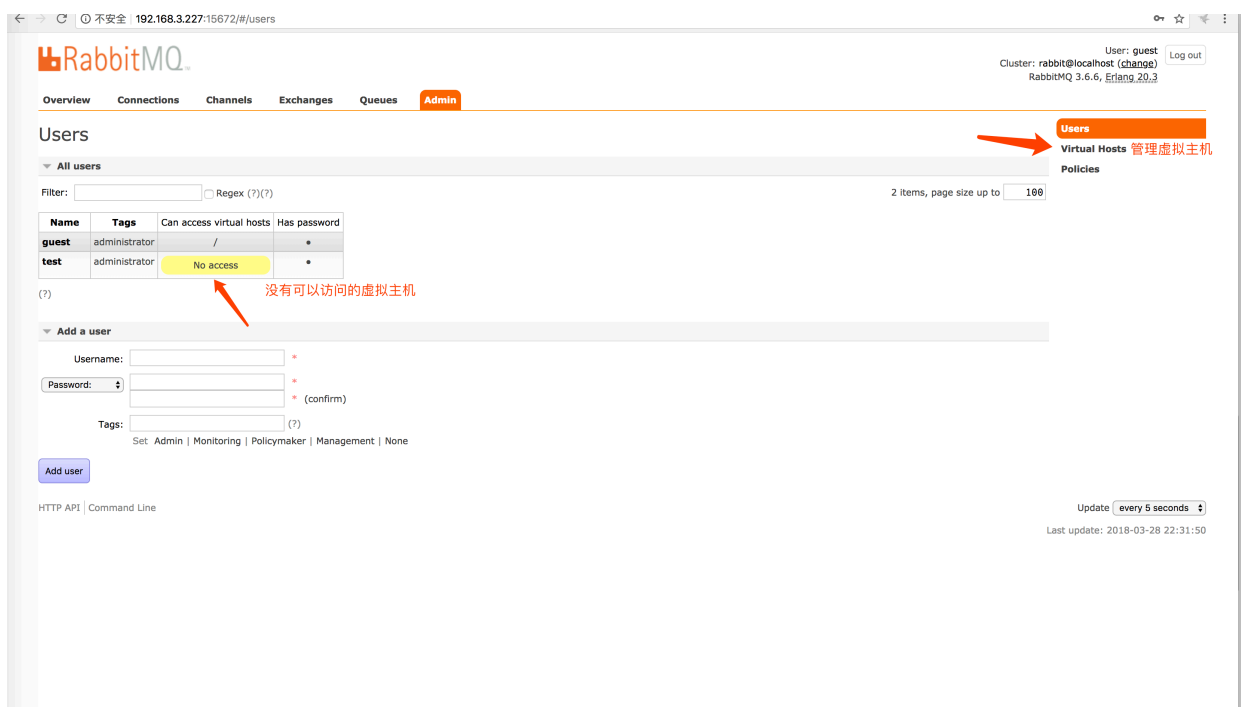


3.2.2.2 添加用户

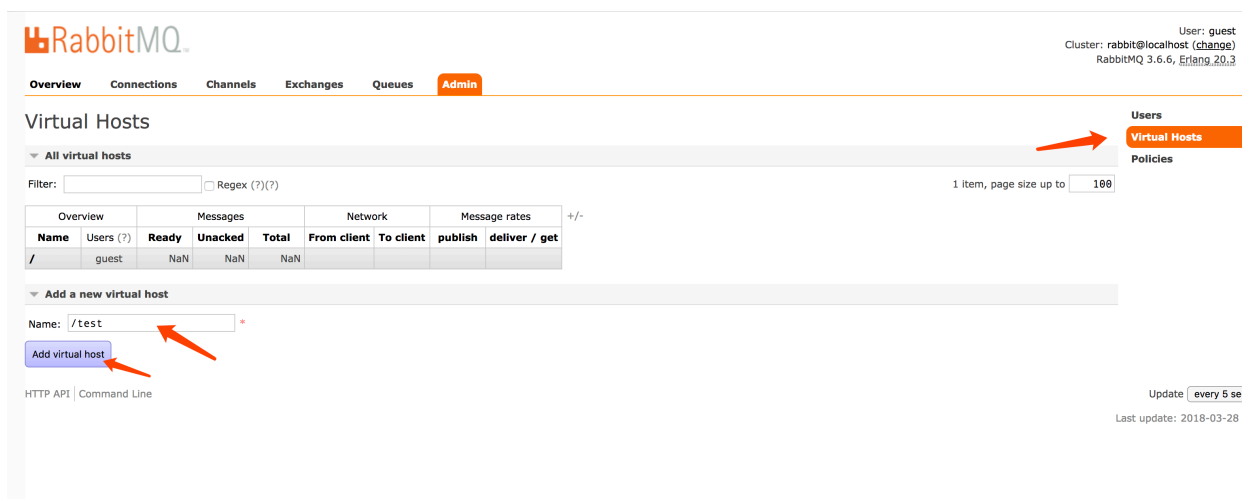


3.2.2.3 分配可以访问的虚拟主机

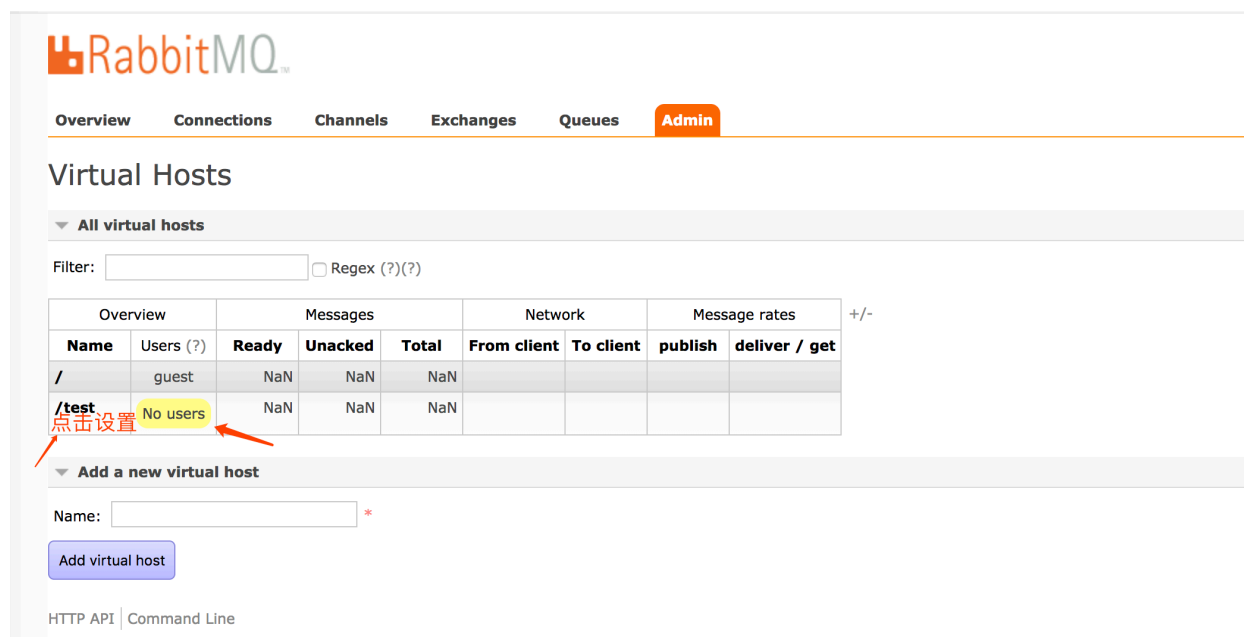
默认情况下没有任何可以访问的,我们可以添加一个主机(相当于添加一个数据库),然后分配权限



3.2.2.4 创建虚拟主机



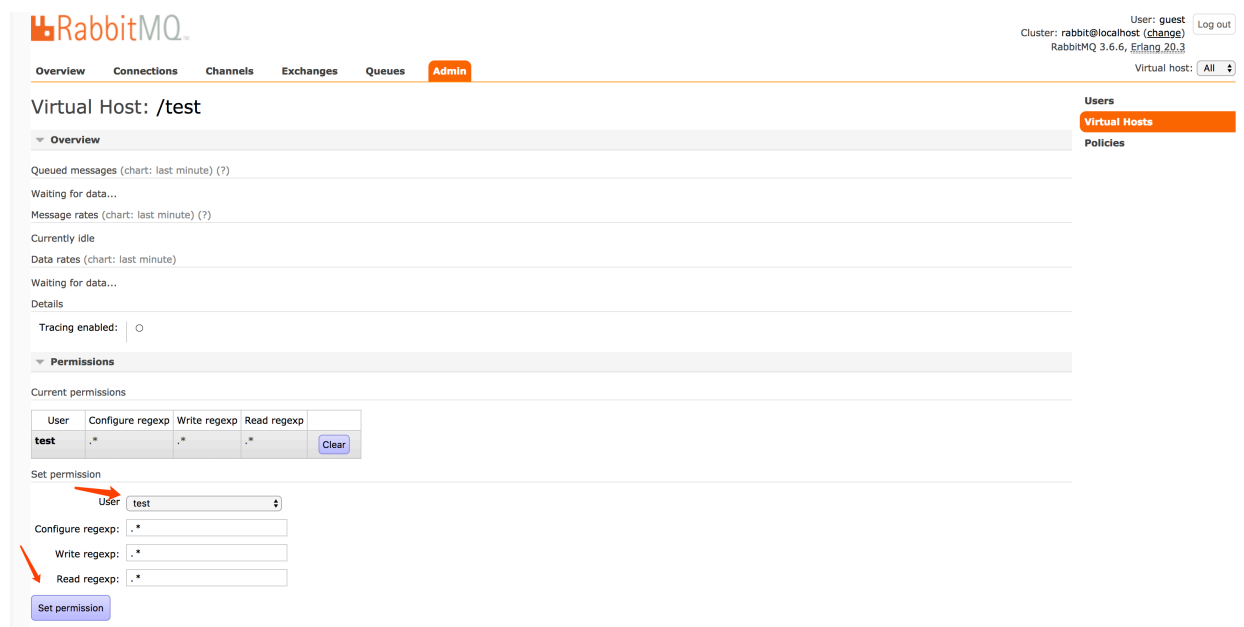
3.2.2.5 给虚拟主机分配权限



The screenshot shows the RabbitMQ Admin interface. The 'Admin' tab is selected. Under 'Virtual Hosts', the 'All virtual hosts' section is expanded. A table lists virtual hosts. The first row is for the '/' host, showing it has 'guest' users. The second row is for the '/test' host, showing 'No users'. A red arrow points to the '/test' row with the text '点击设置' (Click settings). Below the table, there is a form to 'Add a new virtual host' with a 'Name' field and an 'Add virtual host' button. At the bottom, there are links for 'HTTP API' and 'Command Line'.

Overview		Messages			Network		Message rates	
Name	Users (?)	Ready	Unacked	Total	From client	To client	publish	deliver / get
/	guest	NaN	NaN	NaN				
/test	No users	NaN	NaN	NaN				

3.2.2.6 给指定用户分配权限



The screenshot shows the RabbitMQ Admin interface for the '/test' virtual host. The 'Admin' tab is selected. The 'Permissions' section is expanded. It shows 'Current permissions' for the 'test' user, with 'Configure regexp', 'Write regexp', and 'Read regexp' all set to '.*'. Below this, there is a 'Set permission' section where the 'User' is set to 'test' and the same three regexps are set to '.*'. A red arrow points to the 'Set permission' button. On the right side, there are links for 'Users', 'Virtual Hosts', and 'Policies'.

User	Configure regexp	Write regexp	Read regexp
test	.*	.*	.*

四 消息

<http://www.rabbitmq.com/getstarted.html>

消息测试都在一个项目中,不同包下做测试

4.1消息模式种类

1 "Hello World!"

The simplest thing that does something



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

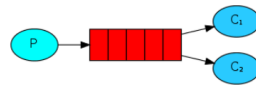
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

2 Work queues

Distributing tasks among workers (the [competing consumers pattern](#))



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

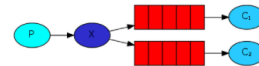
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

3 Publish/Subscribe

Sending messages to many consumers at once



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

4 Routing

Receiving messages selectively



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

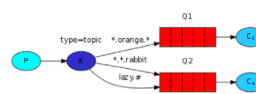
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

5 Topics

Receiving messages based on a pattern (topics)



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

6 RPC

[Request/reply pattern](#)
example



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

[Spring AMQP](#)

4.2 pom&log4j.properties

```

<dependencies>
  <!-- https://mvnrepository.com/artifact/com.rabbitmq/amqp-client -->
  <!-- https://mvnrepository.com/artifact/com.rabbitmq/amqp-client -->
  <dependency>
    <groupId>com.rabbitmq</groupId>
  
```

```
        <artifactId>amqp-client</artifactId>
        <version>4.5.0</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12 -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.25</version>
    </dependency>

    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.3.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework.amqp/spring-
rabbit
整合 spring 时使用,amqp 只对 rabbitmq 做了支持
-->
    <dependency>
        <groupId>org.springframework.amqp</groupId>
        <artifactId>spring-rabbit</artifactId>
        <version>1.7.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
</dependencies>
```

log4j.properties

```
log4j.rootLogger=DEBUG,A1
log4j.logger.com.taotao = DEBUG
log4j.logger.org.mybatis = DEBUG

log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-
[%p] %m%n
```

4.3 工具类ConnectionUtil

```
public class ConnectionUtil {

    public static Connection getConnection() throws Exception {
        //定义连接工厂
        ConnectionFactory factory = new ConnectionFactory();
        //设置服务地址
        factory.setHost("192.168.3.227");
        //端口
        factory.setPort(5672);
        //设置账号信息，用户名、密码、vhost
        factory.setVirtualHost("/test");
        factory.setUsername("test");
        factory.setPassword("test");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        return connection;
    }

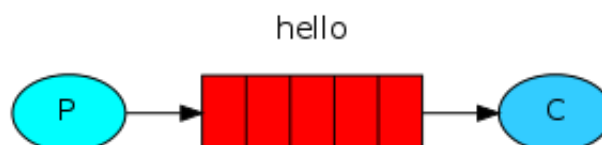
}
```

4.2 简单模式

<http://www.rabbitmq.com/tutorials/tutorial-one-python.html>

简单模式就是我们的生产者将消息发到队列,消费者从队列中取消息

一条消息对应一个消费者



4.2.1 生产者

```
public class Send {  
  
    private final static String QUEUE_NAME = "testhello";  
  
    public static void main(String[] argv) throws Exception {  
        // 获取到连接以及mq通道  
        Connection connection = ConnectionUtil.getConnection();//相当于数据库中的创建连接  
  
        // 从连接中创建通道  
        Channel channel = connection.createChannel();//相当于数据库中的 statement  
  
        // 声明（创建）队列,如果存在就不创建,不存在就创建  
        //参数1 队列名,  
        //参数2 durable: 是否持久化, 队列的声明默认是存放在内存中的, 如果rabbitmq重启会丢失, 如果想重启之后还存在就要使队列持久化, 保存到Erlang自带的Mnesia数据库中, 当rabbitmq重启之后会读取该数据库  
        //exclusive: 是否排外的, 有两个作用, 一: 当连接关闭时connection.close()该队列是否会自动删除; 二: 该队列是否是私有的private, 如果不是排外的, 可以使用两个消费者都访问同一个队列, 没有任何问题, 如果是排外的, 会对当前队列加锁, 其他通道channel是不能访问的, 如果强制访问会报异常: com.rabbitmq.client.ShutdownSignalException: channel error; protocol method: #method<channel.close>(reply-code=405, reply-text=RESOURCE_LOCKED - cannot obtain exclusive access to locked queue 'queue_name' in vhost '/', class-id=50, method-id=20)一般等于true的话用于一个队列只能有一个消费者来消费的场景  
        //autoDelete: 是否自动删除, 当最后一个消费者断开连接之后队列是否自动被删除, 可以通过RabbitMQ Management, 查看某个队列的消费者数量, 当consumers = 0时队列就会自动删除  
        //arguments: 参数  
  
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);//  
  
        // 消息内容  
        String message = "Hello World!";  
        //参数1 交换机,此处无 参数2 发送到哪个队列 ,参数3 属性 参数4 内容  
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes());//将消息发动到数据库  
  
        System.out.println(" 发送数据 '" + message + "'");  
  
        //关闭通道和连接  
        channel.close();  
        connection.close();  
    }  
}
```

4.2.2 消费者

```

/**
 * Created by jackiechan on 2018/4/5/下午4:01
 */
public class Recver {
    private final static String QUEUE = "testhello";//队列的名字

    public static void main(String[] args) throws Exception{
        Connection connection = ConnexionUtil.getConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare(QUEUE, false, false, false, null);

        QueueingConsumer consumer = new QueueingConsumer(channel);//定义一个消费者,QueueingConsumer已经过时,建议使用DefaultConsumer子类
        //接收消息 ,参数2 是自动确认
        channel.basicConsume(QUEUE, true, consumer);

        while (true) {
            //获取消息
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();//如果没有消息会等待,有的话就获取执行然后销毁,是一次性的
            String message = new String(delivery.getBody());
            System.out.println(message);
        }
    }
}

```

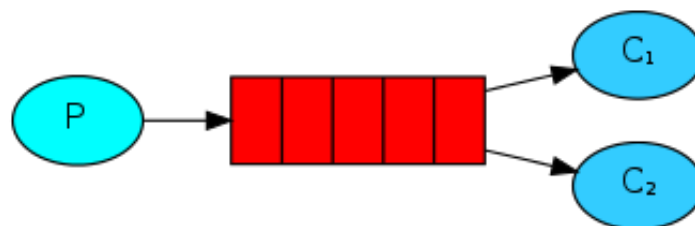
4.2.3 测试

运行 send, 和 recv 测试可以收发消息

4.3 work 模式

一条消息可以被多个消费者尝试接收,但是最终只能有一个消费者能获取

<http://www.rabbitmq.com/tutorials/tutorial-two-python.html>



4.3.1 发送者

```

public class Send {

```

```

private final static String QUEUE_NAME = "testwork";

public static void main(String[] argv) throws Exception {
    // 获取到连接以及mq通道
    Connection connection = ConnectionUtil.getConnection();
    Channel channel = connection.createChannel();

    // 声明队列
    channel.queueDeclare(QUEUE_NAME, false, false, false, null);

    for (int i = 0; i < 100; i++) { // 循环发送消息, 但是每条消息的时间间隔越来越长
        // 消息内容
        String message = "" + i;
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
        System.out.println(" 发送消息 '" + message + "'");

        Thread.sleep(i * 10); // 休眠
    }

    channel.close();
    connection.close();
}
}

```

4.3.2 消费者1

```

/**
 * Created by jackiechan on 2018/4/5/下午4:11
 */
public class Recver1 {
    private final static String QUEUE = "testwork"; // 队列的名字

    public static void main(String[] args) throws Exception {

        Connection connection = ConnexionUtil.getConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare(QUEUE, false, false, false, null);
        // 同一时刻服务器只会发一条消息给消费者, 只有当前消费者将消息处理完成后才会获取到
        下一条消息
        // 注释掉后可以获取多条消息, 但是会一条一条处理
        channel.basicQos(1); // 告诉服务器, 在我们没有确认当前消息完成之前, 不要给我发新的
        消息

        DefaultConsumer consumer = new DefaultConsumer(channel) {

            @Override

```

```

        public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
            //当我们收到消息的时候调用
            System.out.println("消费者1 收到的内容是:"+new String(body));
            //确认
            try {
                Thread.sleep(10); //模拟耗时
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            channel.basicAck(envelope.getDeliveryTag(), false); //参数2, false 为
确认收到消息, true 为拒接收到消息
        }
    };
    //注册消费者, 参数2 手动确认, 代表我们收到消息后需要手动告诉服务器, 我收到消息
了
    channel.basicConsume(QUEUE, false, consumer);
}
}

```

4.3.3 消费者2

```

/**
 * Created by jackiechan on 2018/4/5/下午4:11
 */
public class Recver2 {
    private final static String QUEUE = "testwork"; //队列的名字

    public static void main(String[] args) throws Exception {

        Connection connection = ConnexionUtil.getConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare(QUEUE, false, false, false, null);
        // 同一时刻服务器只会发一条消息给消费者, 只有当前消费者将消息处理完成后才会获取到
下一条消息
        //注释掉后可以获取多条消息, 但是会一条一条处理
        channel.basicQos(1); //告诉服务器, 在我们没有确认当前消息完成之前, 不要给我发新的
消息
        DefaultConsumer consumer = new DefaultConsumer(channel){

            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                //当我们收到消息的时候调用

```

```
System.out.println("消费者2 收到的内容是:"+new String(body));
//确认
try {
    Thread.sleep(10); //模拟耗时
} catch (InterruptedException e) {
    e.printStackTrace();
}
channel.basicAck(envelope.getDeliveryTag(), false); //参数2, false 为
确认收到消息, true 为拒接收到消息
};
//注册消费者, 参数2 手动确认, 代表我们收到消息后需要手动告诉服务器, 我收到消息
了
channel.basicConsume(QUEUE, false, consumer);
}
```

4.3.4 测试

启动消费者1, 消费者2, 发送者

在channel.basicQos(1);代码注释掉的情况下, 我们发现两个消费者获取到的消息数量是一致的, 会轮流从队列取消息

channel.basicQos(1);代码打开后, 发现消费者1可以获取到更多数据, 因为消费者的处理时间端, 处理快, 所以可以获取到更多的消息

4.4 消息的确认模式

当我们发送消息后, 服务端如何知道消息已经被消费

模式1: 自动模式, 不管消费者获取到消息后是否是成功处理消息, 服务端都认为是成功的

模式2: 手动模式, 消费者获取到消息后, 服务器会将消息标记为不可用, 等待消费者反馈, 如果不反馈, 则一直标记为不可用

4.5 订阅模式

<http://www.rabbitmq.com/tutorials/tutorial-three-python.html>

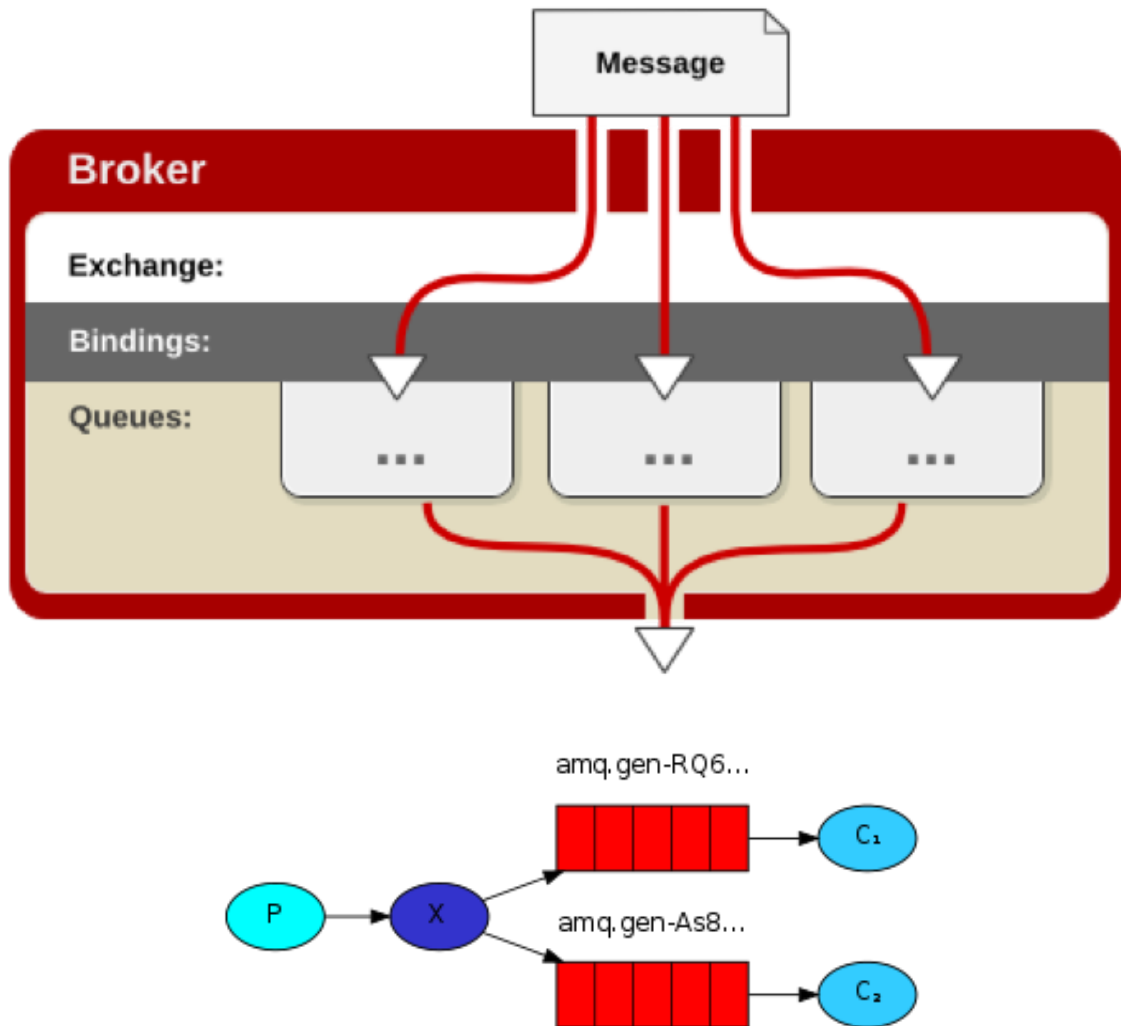
一条消息可以被多个消费者同时获取

生产者将消息发送到交换机

消费者将自己对应的队列注册到交换机

当发送消息后 所有注册的队列的消费者都可以收到消息

Fanout Exchange



4.5.1 生产者

```
public class Send {

    private final static String EXCHANGE_NAME = "testexchange";

    public static void main(String[] argv) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明exchange
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

        // 消息内容
        String message = "Hello World!";
        //将消息发送到交换机,如果此时没有队列绑定,则消息会丢失,因为交换机没有存储消息的能力

        channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
        System.out.println(" 发送消息 '" + message + "'");
    }
}
```

```

        channel.close();
        connection.close();
    }
}

```

4.5.2 消费者1

```

/**
 * Created by jackiechan on 2018/4/5/下午4:33
 */
public class Recver1 {
    private final static String EXCHANGE_NAME = "testexchange";//定义交换机的名字

    public static void main(String[] args) throws Exception{

        Connection connection = ConnexionUtil.getConnection();

        Channel channel = connection.createChannel();

        channel.queueDeclare("testpubqueue1", false, false, false,null);
        //绑定队列到交换机
        channel.queueBind("testpubqueue1", EXCHANGE_NAME, "");

        channel.basicQos(1);
        DefaultConsumer consumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者11111111:"+new String(body));
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };
        channel.basicConsume("testpubqueue1", false, consumer);
    }
}

```

4.5.3 消费者2

```

/**
 * Created by jackiechan on 2018/4/5/下午4:33
 */
public class Recver2 {
    private final static String EXCHANGE_NAME = "testexchange";//定义交换机的名字

    public static void main(String[] args) throws Exception{

```

```
Connection connection = ConnexionUtil.getConnection();

Channel channel = connection.createChannel();

channel.queueDeclare("testpubqueue2", false, false, false, null);

//绑定队列到交换机
channel.queueBind("testpubqueue2", EXCHANGE_NAME, "");

channel.basicQos(1);

DefaultConsumer consumer=new DefaultConsumer(channel){
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
        System.out.println("消费者2222222:"+new String(body));
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};
channel.basicConsume("testpubqueue2", false, consumer);
}
```

4.5.4 测试

启动消费者1,2 生产者测试

4.6 路由模式

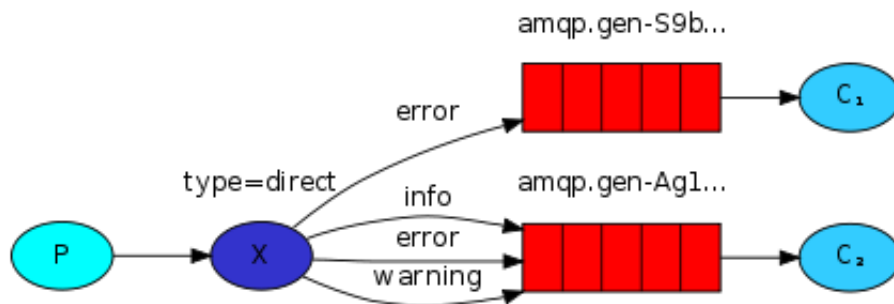
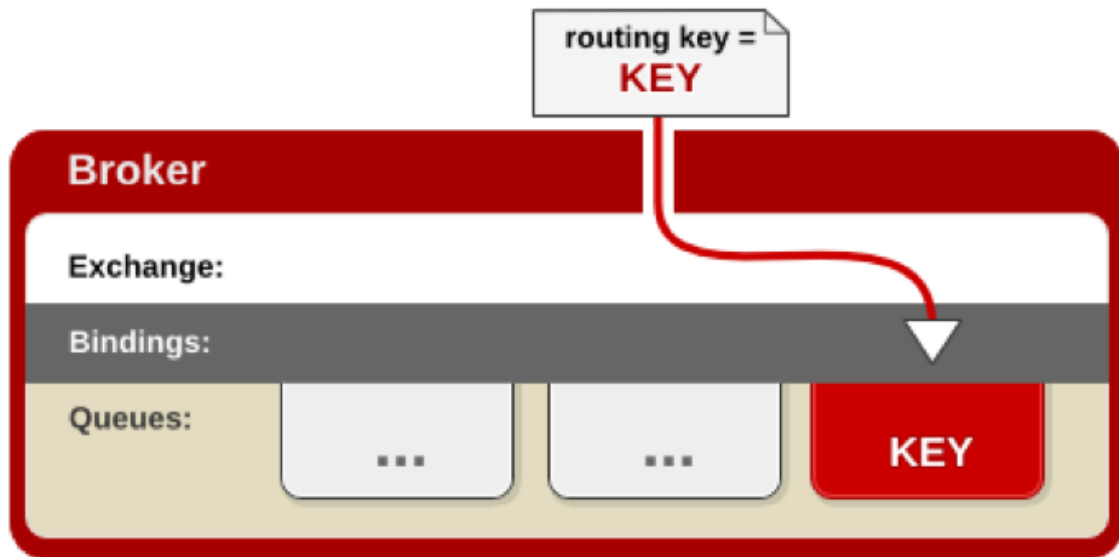
<http://www.rabbitmq.com/tutorials/tutorial-four-python.html>

生产者将消息发送到了 type 为 direct 模式的交换机

消费者的队列在将自己绑定到路由的时候会给自己绑定一个 key

只有消费者发送对应 key 格式的消息时候 队列才会收到消息

Direct Exchange



4.6.1 生产者

```
/**
 * Created by jackiechan on 2018/4/5/下午4:43
 */
public class Sender {
    private final static String EXCHANGE_NAME = "testroute";

    public static void main(String[] args) throws Exception{
        Connection connection = ConnexionUtil.getConnection();
        Channel channel = connection.createChannel();
        channel.exchangeDeclare(EXCHANGE_NAME, "direct");//定义路由格式的交换机
        //发送 key 为 key3的消息,测试的时候根据接收者的具体 key 做测试
        channel.basicPublish(EXCHANGE_NAME, "key3", null, "路由消息".getBytes());
        channel.close();
        connection.close();
    }
}
```

4.6.2 消费者1

```

/**
 * Created by jackiechan on 2018/4/5/下午4:33
 */
public class Recver1 {
    private final static String EXCHANGE_NAME = "testroute";//定义交换机的名字

    public static void main(String[] args) throws Exception{
        Connection connection = ConnexionUtil.getConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare("testroutequeue1", false, false, false,null);
        //绑定队列到交换机
        //参数3 标记,绑定到交换机的时候会指定一个标记,只有和它一样的标记的消息才会被当前
        //消费者收到
        // 绑定队列到交换机,绑定自己的关键字 key 为key,注意在绑定到指定路由(交换机)的时
        //候,该路由必须存在,也就是我们必须先由发送者创建一个路由才可以
        channel.queueBind("testroutequeue1", EXCHANGE_NAME, "key1");
        //如果要接收多个标记,只需要再执行一次即可
        channel.queueBind("testroutequeue1", EXCHANGE_NAME, "key2");
        channel.basicQos(1);
        DefaultConsumer consumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
            AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者11111111:"+new String(body));
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };
        channel.basicConsume("testroutequeue1", false, consumer);
    }
}

```

4.6.3 消费者2

```

/**
 * Created by jackiechan on 2018/4/5/下午4:33
 */
public class Recver2 {
    private final static String EXCHANGE_NAME = "testroute";//定义交换机的名字

    public static void main(String[] args) throws Exception{
        Connection connection = ConnexionUtil.getConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare("testroutequeue1", false, false, false,null);
        //绑定队列到交换机
    }
}

```

//参数3 标记,绑定到交换机的时候会指定一个标记,只有和它一样的标记的消息才会被当前消费者收到

// 绑定队列到交换机,绑定自己的关键字 key 为key,注意在绑定到指定路由(交换机)的时候,该路由必须存在,也就是我们必须先由发送者创建一个路由才可以

```
channel.queueBind("testroutequeue2", EXCHANGE_NAME, "key1");
//如果要接收多个标记,只需要再执行一次即可
channel.queueBind("testroutequeue2", EXCHANGE_NAME, "key2");
channel.basicQos(1);
DefaultConsumer consumer=new DefaultConsumer(channel){
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
        System.out.println("消费者22222:"+new String(body));
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};
channel.basicConsume("testroutequeue2", false, consumer);
}
```

4.6.4 测试

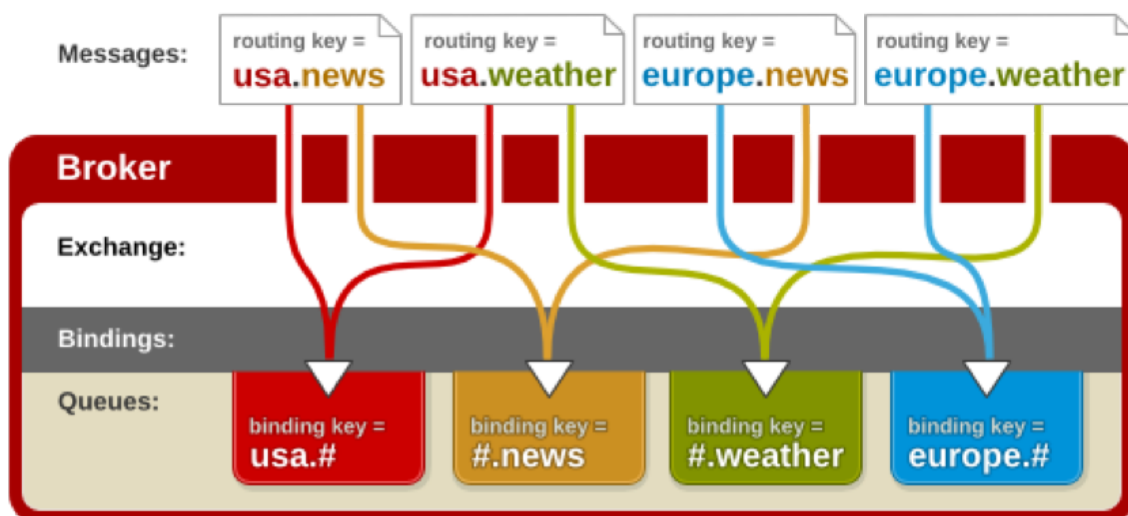
启动发送者创建路由,启动消费者1,消费者2

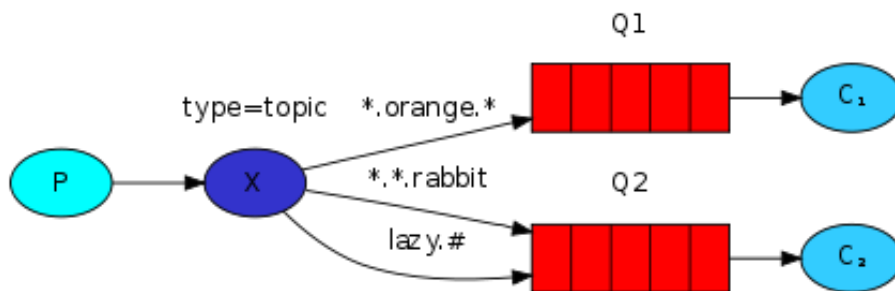
通过修改发送者代码中的 key 来多次执行测试,发现可以分别收到不同消息,如果监听了相同的 key 可以一起收到消息

4.7 通配符模式

<http://www.rabbitmq.com/tutorials/tutorial-five-python.html>

Topic Exchange





- *(star) can substitute for exactly one word.
- # (hash) can substitute for zero or more words.
- 将路由键和某模式进行匹配。此时队列需要绑定到一个模式上。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”

4.7.1 生产者

```

/**
 * Created by jackiechan on 2018/4/5/下午4:54
 */
public class Sender {
    private final static String EXCHANGE_NAME = "testtopic";

    public static void main(String[] args) throws Exception{
        Connection connection = ConnexionUtil.getConnection();
        Channel channel = connection.createChannel();
        // 声明exchange,声明为 topic 也就是通配符类型
        channel.exchangeDeclare(EXCHANGE_NAME, "topic");
        //发送 abc.1.2数据,凡是能匹配到这个关键词的都会收到
        channel.basicPublish(EXCHANGE_NAME, "abc.1.2", null, "topic 模式消息
111".getBytes());
        channel.close();
        connection.close();
    }
}

```

4.7.2消费者1

```

/**
 * Created by jackiechan on 2018/4/5/下午4:33
 */
public class Recver1 {
    private final static String EXCHANGE_NAME = "testtopic";//定义交换机的名字

    public static void main(String[] args) throws Exception{

```

```

Connection connection = ConnexionUtil.getConnection();
Channel channel = connection.createChannel();
channel.queueDeclare("testtopicqueue1", false, false, false, null);
//绑定队列到交换机
//参数3 标记,绑定到交换机的时候会指定一个标记,只有和它一样的标记的消息才会被当前
消费者收到

channel.queueBind("testtopicqueue1", EXCHANGE_NAME, "key.*");
//如果要接收多个标记,只需要再执行一次即可
channel.queueBind("testtopicqueue1", EXCHANGE_NAME, "abc.#");
channel.basicQos(1);
DefaultConsumer consumer=new DefaultConsumer(channel){
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
        System.out.println("消费者11111111:"+new String(body));
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};
channel.basicConsume("testtopicqueue1", false, consumer);
}
}

```

4.7.3 消费者2

```

/**
 * Created by jackiechan on 2018/4/5/下午4:33
 */
public class Recver2 {
    private final static String EXCHANGE_NAME = "testtopic";//定义交换机的名字

    public static void main(String[] args) throws Exception{
        Connection connection = ConnexionUtil.getConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare("testtopicqueue2", false, false, false, null);
        //绑定队列到交换机
        //参数3 标记,绑定到交换机的时候会指定一个标记,只有和它一样的标记的消息才会被当前
消费者收到

        channel.queueBind("testtopicqueue2", EXCHANGE_NAME, "key.#");
        //如果要接收多个标记,只需要再执行一次即可
        channel.queueBind("testtopicqueue2", EXCHANGE_NAME, "abc.#");
        channel.basicQos(1);
        DefaultConsumer consumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者22222:"+new String(body));
            }
        };
        channel.basicConsume("testtopicqueue2", false, consumer);
    }
}

```

```

        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};
channel.basicConsume("testtopicqueue2", false, consumer);
}
}

```

4.7.4 测试

通过发送不同的 key 的数据测试发现消费者可能会在不同情况下分别收到数据

五 整合 spring

spring 对 amqp 做了支持,但是当前只实现了 rabbitmq

5.1 spring 自动模式

5.1.1 spring 配置文件

```

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:rabbit="http://www.springframework.org/schema/rabbit"
    xsi:schemaLocation="http://www.springframework.org/schema/rabbit
        http://www.springframework.org/schema/rabbit/spring-rabbit-1.7.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd">

    <!-- 定义RabbitMQ的连接工厂 -->
    <rabbit:connection-factory id="connectionFactory"
        host="192.168.3.227" port="5672" username="test" password="test"
        virtual-host="/test" />

    <!-- 定义Rabbit模板, 指定连接工厂以及定义exchange
        如果要消息发送到队列而不是交换机,则声明queue="" 而不是exchange=""
    -->
    <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
        exchange="fanoutExchange" />
    <!-- <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
        exchange="fanoutExchange" routing-key="foo.bar" /> -->

    <!-- MQ的管理, 包括队列、交换器等 -->
    <rabbit:admin connection-factory="connectionFactory" />

    <!-- 定义队列, 自动声明 -->
    <rabbit:queue name="myQueue" auto-declare="true"/>

    <!-- 定义交换器, 自动声明 -->

```

```
<rabbit:fanout-exchange name="fanoutExchange" auto-declare="true" >
    <rabbit:bindings>
        <!--将下列队列绑定到当前交换机-->
        <rabbit:binding queue="myQueue"/>
    </rabbit:bindings>
</rabbit:fanout-exchange>
<!--
通配符模式
-->
<!--

<rabbit:topic-exchange name="myExchange">
    <rabbit:bindings>
        <rabbit:binding queue="myQueue" pattern="foo.*" />
    </rabbit:bindings>
</rabbit:topic-exchange>

-->

<!--路由设置 将队列绑定，属于direct类型
<rabbit:direct-exchange id="directExchange"
                        name="directExchange" durable="true" auto-
delete="false">
    <rabbit:bindings>
        <rabbit:binding queue="myQueue"
key="${rabbitmq.system.out.log.error.mail}" />
    </rabbit:bindings>
</rabbit:direct-exchange>

-->

<!-- 队列监听
acknowledged = "manual" 属性为手动应答
-->
<rabbit:listener-container connection-factory="connectionFactory">
    <!--指定对应队列myQueue的监听为 foo 中的 listen 方法-->
    <rabbit:listener ref="foo" method="listen" queue-names="myQueue" />
</rabbit:listener-container>

<bean id="foo" class="com.qianfeng.rabbitmq.spring.Foo" />

</beans>
```

5.1.2 接收者

```
/**
 * 消费者类,任意类都可以
 */
public class Foo {

    //具体执行业务的方法
    public void listen(String foo) {
        System.out.println("消费者: " + foo);
    }
}
```

5.1.3 测试类

```
public class TestMain {
    public static void main(final String... args) throws Exception {
        AbstractApplicationContext ctx = new ClassPathXmlApplicationContext(
            "classpath:spring/rabbitmq-context.xml");
        //RabbitMQ模板
        RabbitTemplate template = ctx.getBean(RabbitTemplate.class);
        //发送消息
        template.convertAndSend("Hello, world!");
        Thread.sleep(1000); // 休眠1秒
        ctx.destroy(); // 容器销毁
    }
}
```

5.1.4 启动测试

5.2 spring 手动模式

5.2.1 配置文件

主要介绍template 和 listener 的不同,其他同上

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:rabbit="http://www.springframework.org/schema/rabbit"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/rabbit
http://www.springframework.org/schema/rabbit/spring-rabbit-1.7.xsd
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">

<context:component-scan base-package="com.qianfeng.rabbitmq"/>
    <bean id="jsonMessageConverter"
class="org.springframework.amqp.support.converter.Jackson2JsonMessageConverter" />

    <!-- 定义RabbitMQ的连接工厂
publisher-confirms 发布确认,防止消息丢失-->
    <rabbit:connection-factory id="connectionFactory"
                                host="192.168.3.227" port="5672" username="test"
password="test"
                                virtual-host="/test" publisher-confirms="true" />

    <rabbit:admin connection-factory="connectionFactory" />

    <!-- 给模板指定转换器 --><!-- mandatory必须设置true,return callback才生效 -->
    <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
                    confirm-callback="confirmCallbackListener"
                    return-callback="returnCallbackListener"
                    mandatory="true"
    />

    <rabbit:queue name="CONFIRM_TEST" />

    <rabbit:direct-exchange name="DIRECT_EX" id="DIRECT_EX" >
        <rabbit:bindings>
            <rabbit:binding queue="CONFIRM_TEST" />
        </rabbit:bindings>
    </rabbit:direct-exchange>

    <!-- 配置consumer, 监听的类和queue的对应关系 -->
    <rabbit:listener-container
        connection-factory="connectionFactory" acknowledge="manual" >
        <rabbit:listener queues="CONFIRM_TEST" ref="receiveConfirmTestListener" />
    </rabbit:listener-container>
</beans>

```

5.2.2 消费者

```

/**
 * Created by jackiechan on 2018/3/29/上午12:51
 */
@Service("receiveConfirmTestListener")
public class ReceiveConfirmTestListener implements ChannelAwareMessageListener {

```

```

@Override
public void onMessage(Message message, Channel channel) throws Exception {
    try{
        System.err.println("消费者收到消息--
:"+message.getMessageProperties()+":"+new String(message.getBody()));
        channel.basicAck(message.getMessageProperties().getDeliveryTag(),
false); //手动应答
    }catch(Exception e){
        e.printStackTrace();//TODO 业务处理
        channel.basicNack(message.getMessageProperties().getDeliveryTag(),
false,false);
    }
}
}

```

5.2.3 确认后回调

```

/**
 * Created by jackiechan on 2018/3/29/上午12:51
 */
@Service("confirmCallbackListener")
public class ConfirmCallbackListener implements RabbitTemplate.ConfirmCallback {
    @Override
    public void confirm(CorrelationData correlationData, boolean ack, String
cause) {
        System.err.println("确认--
:correlationData:"+correlationData+",ack:"+ack+",cause:"+cause);
    }
}

```

5.2.3 失败回滚

```

/**
 * Created by jackiechan on 2018/3/29/上午12:51
 */
@Service("returnCallbackListener")
public class ReturnCallbackListener implements RabbitTemplate.ReturnCallback {
    @Override
    public void returnedMessage(Message message, int replyCode, String replyText,
String exchange, String routingKey) {
        System.err.println("失败--message:"+new
String(message.getBody()+"",replyCode:"+replyCode+",replyText:"+replyText+",exchan
ge:"+exchange+",routingKey:"+routingKey);
    }
}

```

5.2.3 生产者

```
@Service("publishService")
public class PublishService {
    @Autowired
    private AmqpTemplate amqpTemplate;

    public void send(String exchange, String routingKey, Object message) {
        amqpTemplate.convertAndSend(exchange, routingKey, message);
    }
}
```

5.2.4 消费者

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath:spring/application-context.xml"})

public class TestConfirm {
    @Autowired
    private PublishService publishService;

    private static String exChange = "DIRECT_EX";

    @Test
    public void test1() throws InterruptedException{
        String message = "currentTime:"+System.currentTimeMillis();
        System.out.println("test1---message:"+message);
        //exchange,queue 都正确,confirm被回调, ack=true
        publishService.send(exChange,"CONFIRM_TEST",message);
        Thread.sleep(1000);
    }

    @Test
    public void test2() throws InterruptedException{
        String message = "currentTime:"+System.currentTimeMillis();
        System.out.println("test2---message:"+message);
        //exchange 错误,queue 正确,confirm被回调, ack=false
        publishService.send(exChange+"NO","CONFIRM_TEST",message);
        Thread.sleep(1000);
    }

    @Test
    public void test3() throws InterruptedException{
        String message = "currentTime:"+System.currentTimeMillis();
        System.out.println("test3---message:"+message);
        //exchange 正确,queue 错误 ,confirm被回调, ack=true; return被回调
        replyText:NO_ROUTE
    }
}
```

```

        publishService.send(exChange, "", message);
    //        Thread.sleep(1000);
    }

    @Test
    public void test4() throws InterruptedException{
        String message = "currentTime:"+System.currentTimeMillis();
        System.out.println("test4---message:"+message);
        //exchange 错误,queue 错误,confirm被回调, ack=false
        publishService.send(exChange+"NO", "CONFIRM_TEST", message);
        Thread.sleep(1000);
    }
}

```

5.2.5 启动测试

六 消息持久化

6.1 生产者

```

/**
 * Created by jackiechan on 2018/3/30/下午10:36
 */
public class Send {
    private static String EXCHANGE_NAME="test_dur";

    public static void main(String[] args) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();
        boolean durable=true;
        channel.exchangeDeclare(EXCHANGE_NAME, "direct", durable, false, null);
        String message="hello world";
        //发送消息, 并设置为持久化

        channel.basicPublish(EXCHANGE_NAME, "blue",
        MessageProperties.PERSISTENT_TEXT_PLAIN, message.getBytes());
        System.out.println(" 发送消息'" + message + "'");

        channel.close();
        connection.close();
    }
}

```

6.2 消费者

```
/**
 * Created by jackiechan on 2018/3/30/下午10:36
 */
public class Recv {
    private static String EXCHANGE_NAME="test_dur";
    private static String queueName="test_dur_queue";

    public static void main(String[] args) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();
        //声明持久化交换机
        boolean durable=true;
        channel.exchangeDeclare(EXCHANGE_NAME, "direct", durable, false, null);
        //声明持久化队列
        channel.queueDeclare(queueName, durable, false, false, null);
        channel.queueBind(queueName, EXCHANGE_NAME, "blue");
        System.out.println(" [*] Waiting for messages. To exit press CTRL+C");
        QueueingConsumer consumer = new QueueingConsumer(channel);
        channel.basicConsume(queueName, true, consumer);
        while (true)
        {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            String routingKey = delivery.getEnvelope().getRoutingKey();
            System.out.println(" [x] Received routingKey = " + routingKey +
",msg = " + message + ".");
        }
    }
}
```

6.3测试

生产者发送消息后重启 rabbitmq 服务端 启动消费者,收到消息