天翼云

分布式消息 RabbitMQ

用户指南

1 产品介绍

什么是分布式消息服务 RabbitMQ？

分布式消息服务 RabbitMQ 完全兼容 RabbitMQ 开源社区，支持消息路由、事务消息、优先级队列、延迟队列、死信队列、镜像队列等功能的消息云服务。用户可开箱即用，无需部署免运维，从而实现快速上云。

分布式消息服务 RabbitMQ 的优势

分布式消息服务 RabbitMQ 完全兼容开源社区版本，为用户提供一款高可靠、功能丰富、高可用的消息队列。业务无需改动即可快速迁移上云，为您节省维护和使用成本。

优势	概况词	简介文案
功能丰富	功能丰富	支持优先级队列、延迟队列、死信队列、镜像队列等多种丰富的功能。
高可用性	高可用	支持生产消费自动负载均衡、lvs 节点故障时的自动主备切换以及镜像队列安全备份，保证服务的连续性和可靠性；
高安全性	安全性	起源于金融系统，支持多维度权限控制和 SSL 协议。
高可靠性	高可靠	使用了持久化、传输确认、发布确认等机制来保证可靠性；
开箱即用	全托管	用户可开箱即用，无需部署免运维，从而实现快速上云。

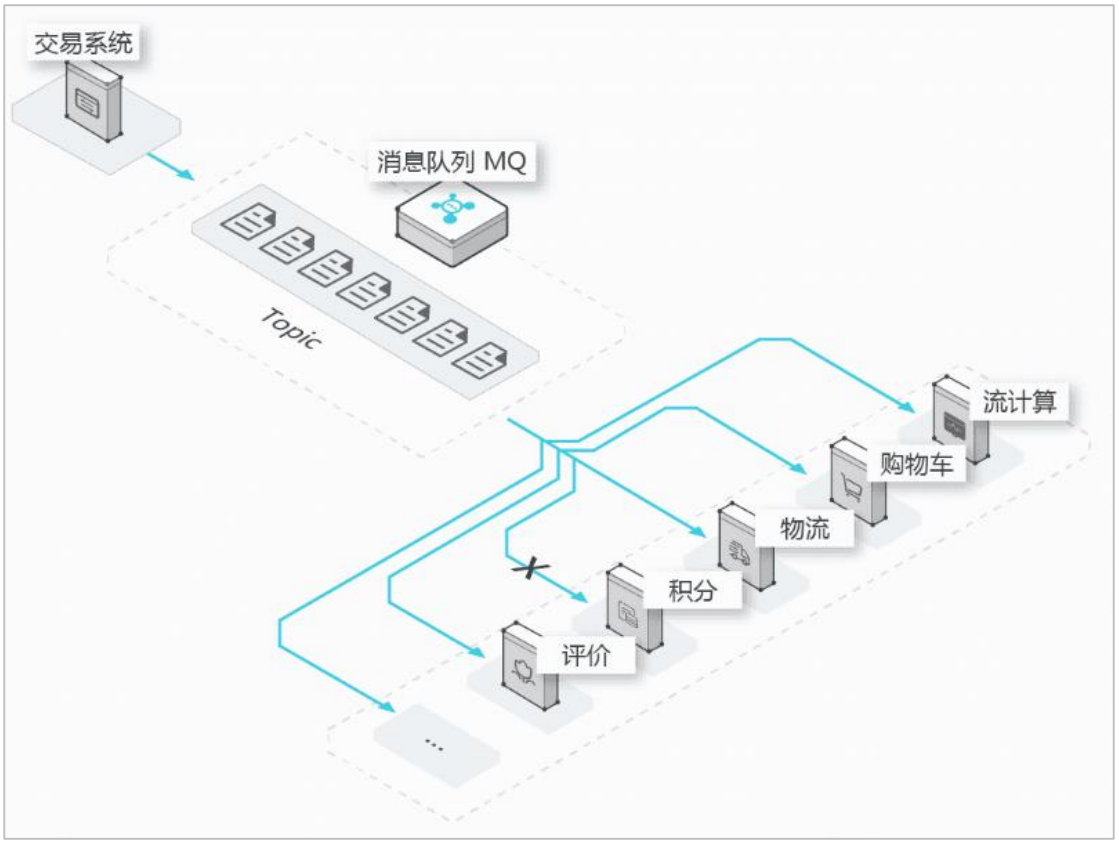
分布式消息服务 RabbitMQ 应用场景

应用场景一：应用解耦

适用场景：以电商秒杀、抢购等流量短时间内暴增场景为例，传统做法是，用户下单后，订单系统发送查询请求到库存系统，等待库存系统返回请求结果给订单系统。如果库存系

统发生故障，订单系统获取不到数据，订单失败。这种情况下，订单系统和库存系统两个子系统高耦合。

能够做到：应用系统解耦，通过上、下游业务系统的松耦合设计，即便下游子系统（如物流、积分等）出现不可用甚至宕机，都不会影响到核心交易系统的正常运转；

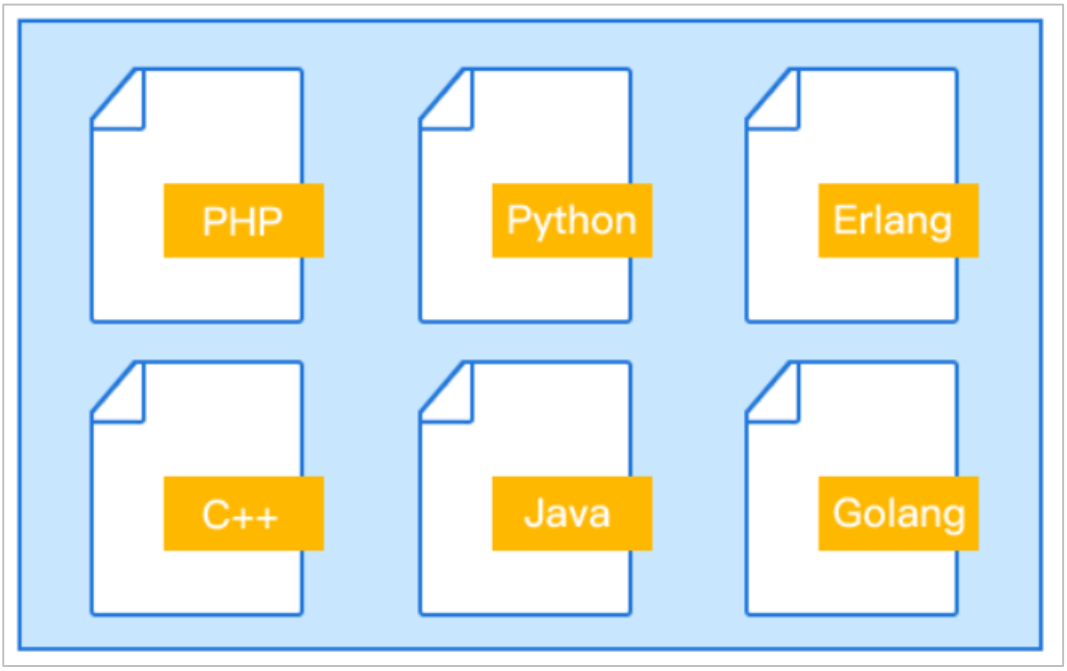


应用场景二：屏蔽平台差异

适用场景：当电商系统架构逐渐成长，差异性带来的问题可能逐渐凸显出来：假如订单系统（order_module）采用 Java 架构，库存系统（ inventory_module）采用 Erlang 架构，而发货系统使用的是 Python 架构.....使用传统的解决方案时，开发人员需要长期维护一些冗余的代码来将各个模块间传入的 HTTP 请求转化为应用程序中的函数调用。

能够做到：屏蔽不同平台，不同编程语言之间的差异。RabbitMQ 提供 Java、.NET、

Ruby、Python、Go 和 Node.js 等多种客户端接入，系统之间的数据交互变得异常简单。



分布式消息服务 RabbitMQ 相关术语解释

Vhost

虚拟主机（Virtual Host），类似于 Namespace 命名空间的概念，逻辑隔离，每个用户里可以创建多个 Vhost，每个 Vhost 可以创建若干个 Exchange 和 Queue。

Queue

消息队列，每个消息都会被投入到一个或者多个 Queue 里。

Producer

消息生产者，即投递消息的程序。

Consumer

消息消费者，即接受消息的程序。

Connection

TCP 连接，Producer 或 Consumer 与消息队列间的物理 TCP 连接。

Channel

在客户端的每个物理 TCP 连接里，可建立多个 Channel，每个 Channel 代表一个会话任务。

Exchange

Producer 将消息发送到 Exchange ，由 Exchange 将消息路由到一个或多个 Queue 中（或者丢弃），Exchange 按照相应的 Binding 逻辑将消息路由到 Queue。

Exchange 类型

- Fanout：该类型路由规则非常简单，会把所有发送到该 Exchange 的消息路由到所有与它绑定的 Queue 中，相当于广播功能。
- Direct：该类型路由规则会将消息路由到 Binding key 与 Routing key 完全匹配的 Queue 中。
- Topic：该类型与 Direct 类型相似，只是规则没有那么严格，可以模糊匹配和多条件匹配，即该类型 Exchange 使用 Routing key 模式匹配和字符串比较的方式将消息路由至绑定的 Queue。

Binding

一套绑定规则，用于告诉 Exchange 消息应该被存储到哪个 Queue。它的作用是把 Exchange 和 Queue 按照路由规则绑定起来。

Routing Key

Producer 在发送消息给 Exchange 时，需要指定一个 Routing key 来设定该消息的路由规则，而 Routing key 需要与 Exchange 类型及 Binding key 联合使用才能生效；一般情况下，Exchange 类型与 Binding key 提供配置好，Producer 在发送消息给 Exchange 时，可以通过指定 Routing key 来决定消息投放到哪个 Queue。

2 购买指南

资源节点

自研资源池

产品规格

目前基础版和高级版规格如下：

产品类型	产品规格
分布式消息服务 RabbitMQ-高级版本	三节点 8 核 16GB lvs 节点 4 核 8GB 50G 磁盘 总磁盘范围 300GB – 6000GB
分布式消息服务 RabbitMQ -基础版本	三节点 4 核 8G lvs 节点 4 核 8GB 50G 磁盘 总磁盘范围 300GB – 6000GB

产品价格

实例价格

实例类型	实例规格	节点数量	标准价格（元/月）
集群版	基础版 4C8G	3	3158
	高级版 8C16G	3	6231

存储价格

同云硬盘单价，目前提供普通 IO 存储类型

存储类型	标准价格（元/G/月）
普通 IO （SATA）	0.3

计费项

实例价格

实例类型	实例规格	节点数量	标准价格（元/月）
集群版	基础版 4C8G	3	3158
	高级版 8C16G	3	6231

存储价格

同云硬盘单价，目前提供普通 IO 存储类型

存储类型	标准价格（元/G/月）
普通 IO （SATA）	0.3

计费方式

提供包周期（包年/包月）计费

包年包月

按包周期实例计费：提供包月和包年的购买模式。

计费方式变更

暂时不提供

到期欠费

到期后暂停服务，数据保留，15 天后实例注销，数据将无法找回。

购买

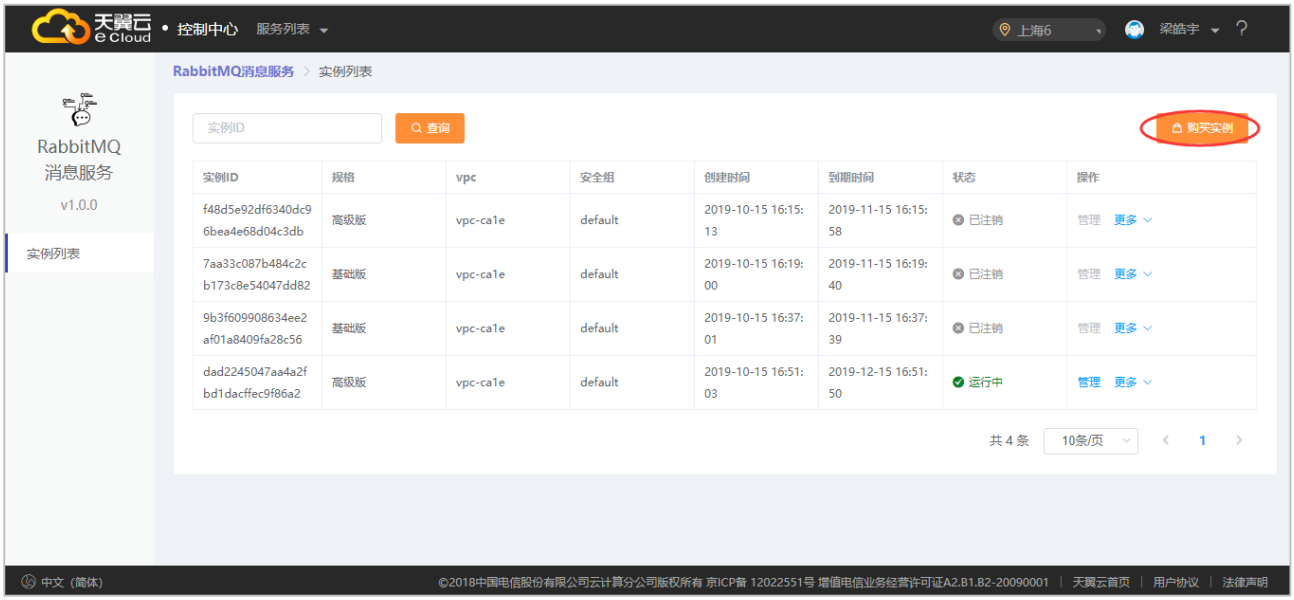
（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）在管理控制台右上角单击“地域名称”，选择区域。

此处请选择与您的应用服务相同的区域。

（4）点击“购买实例”跳转到购买页面。



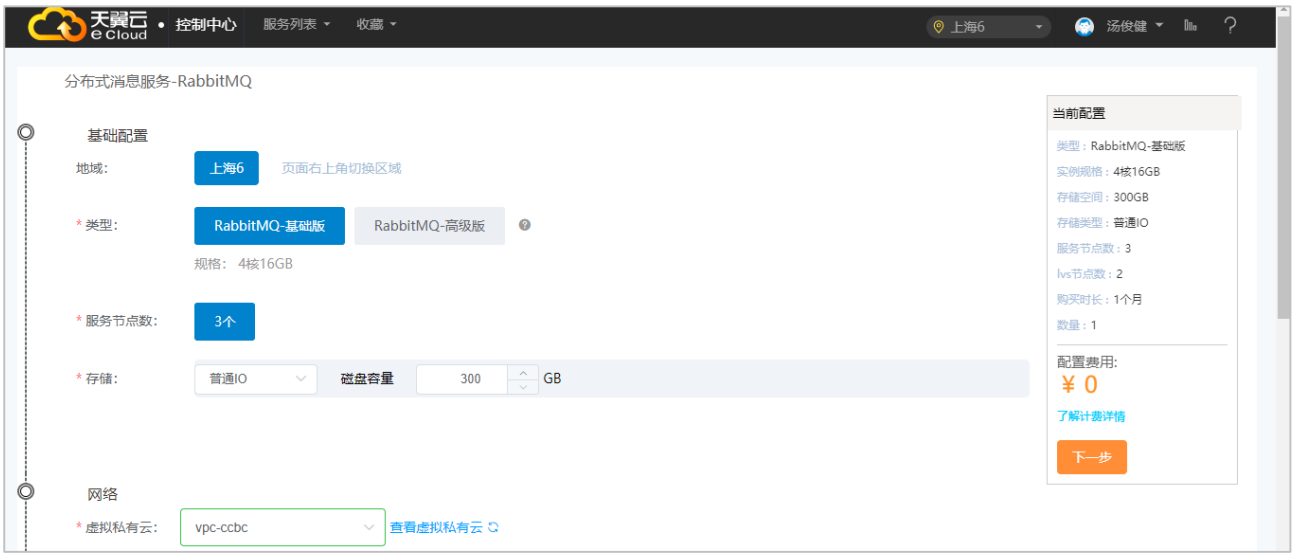
存储空间说明：目前基础版和高级版规格如下：

产品类型	**产品规格**
分布式消息服务 RabbitMQ-高级版本	三节点 8 核 16GB lvs 节点 4 核 8GB 总磁盘范围 300GB – 6000GB
分布式消息服务 RabbitMQ -基础版本	三节点 4 核 8G lvs 节点 4 核 8GB 总磁盘范围 300GB – 6000GB

在集群模式中，RabbitMQ 需要对消息持久化写入到磁盘中，因为，您在创建 RabbitMQ 实例选择存储空间时，建议根据业务消息体积预估以及镜像队列副本数量选择合适的存储

空间。镜像队列副本数最大为集群的节点数，目前都是 3。

例如：业务消息体积预估 100GB，则磁盘容量最少应为 100GB*3 + 预留磁盘大小 100GB。



下单购买

选择理解并接受《公测产品服务协议》，然后下一步订购支付完成购买。

变更

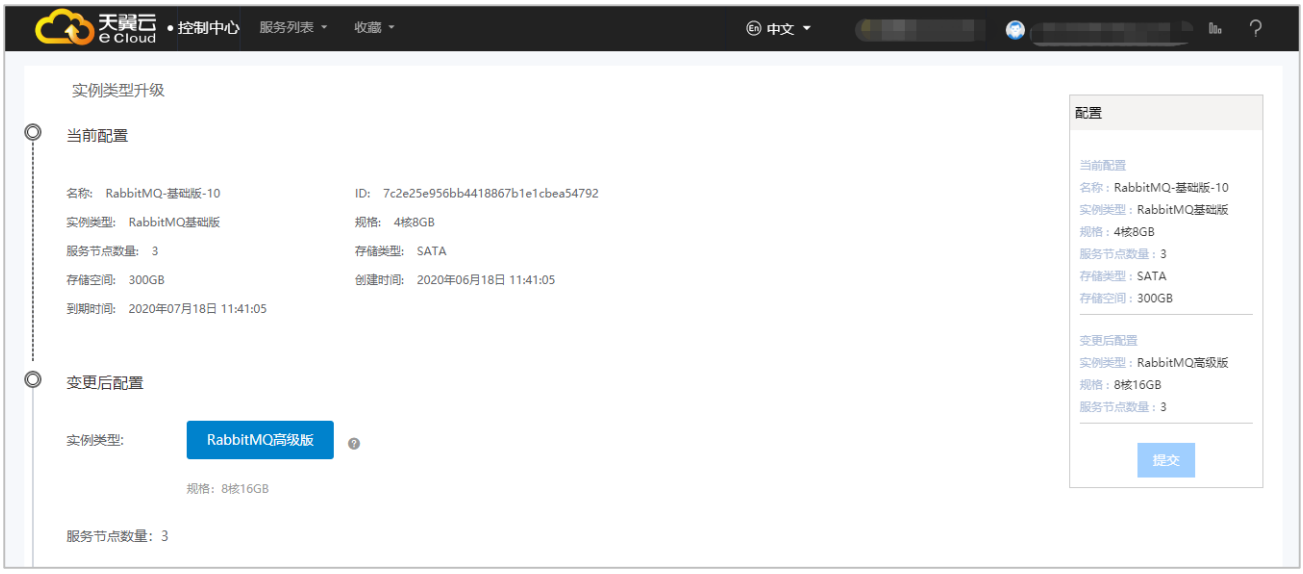
1） 规格变更

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）选择目标实例，点击【更多】，然后点击【规格扩容】。

（4）选择扩容的目标规格，点击【提交】。



（5）完成付费等待施工完成即可。

2） 扩容云硬盘变更

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）选择目标实例，点击【更多】，然后点击【磁盘扩容】。

（4）选择扩容的目标磁盘空间，点击提交。

（5）完成付费等待施工完成即可。

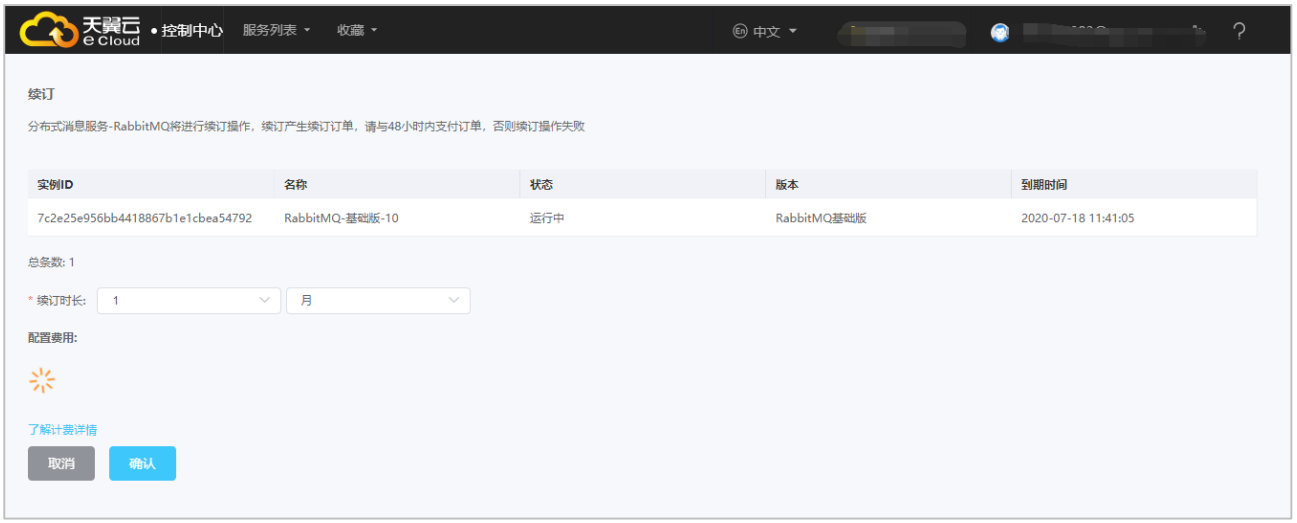
续订

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）选择目标实例，点击【更多】，然后点击【续订】。

（4）选择续订的时间，点击提交。



（5） 完成付费等待施工完成即可。

退订

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）选择目标实例，点击【更多】，然后点击【退订】。

（4）选择确定，点击提交，然后等待施工完成即可。

3 快速入门

注册天翼云账号

购买分布式消息服务 RabbitMQ

创建队列资源

个新的应用接入消息队列需要先创建相关资源，包括：Vhost、User、Exchange、

Queue

1）创建 Vhost

（1）登录消息队列 rabbitMQ 的控制台；

（2）进入相应实例的管理页面；

（3）点击左侧选项卡的集群管理，然后进入主界面的虚拟主机；

（4）点击新建，输入虚拟主机名称即可新增 Vhost。

2）创建 User 并且配置 Vhost 的权限

（1）登录消息队列 rabbitMQ 的控制台；

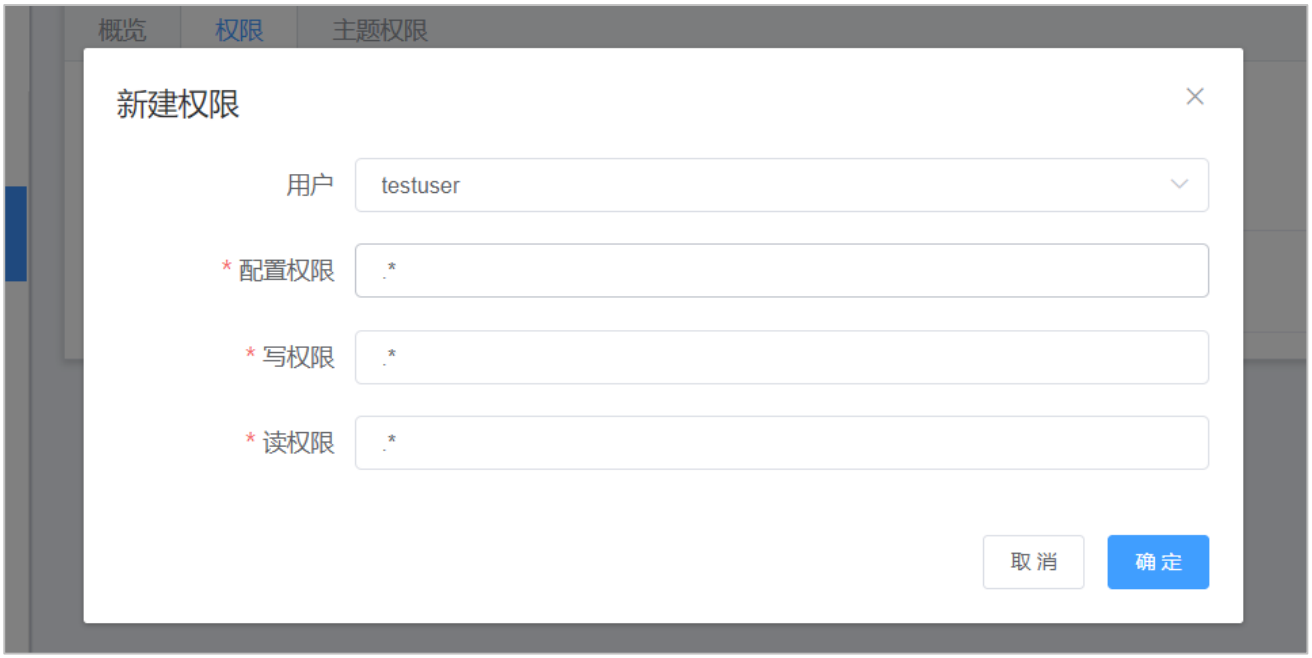
（2）进入相应实例的管理页面；

（3）点击左侧选项卡的集群管理，然后进入主界面的“用户界面”；

（4）创建成功后给用户配置 vhost 的权限，选择目标虚拟主机。

（5）创建成功后给用户配置 vhost 的权限，选择目标虚拟主机，然后进入主机详情，点击权限 tab 页，点击新增权限。

（6）创建成功后给用户配置 vhost 的权限，选择目标虚拟主机，然后进入主机详情，点击权限 tab 页，点击新增权限。

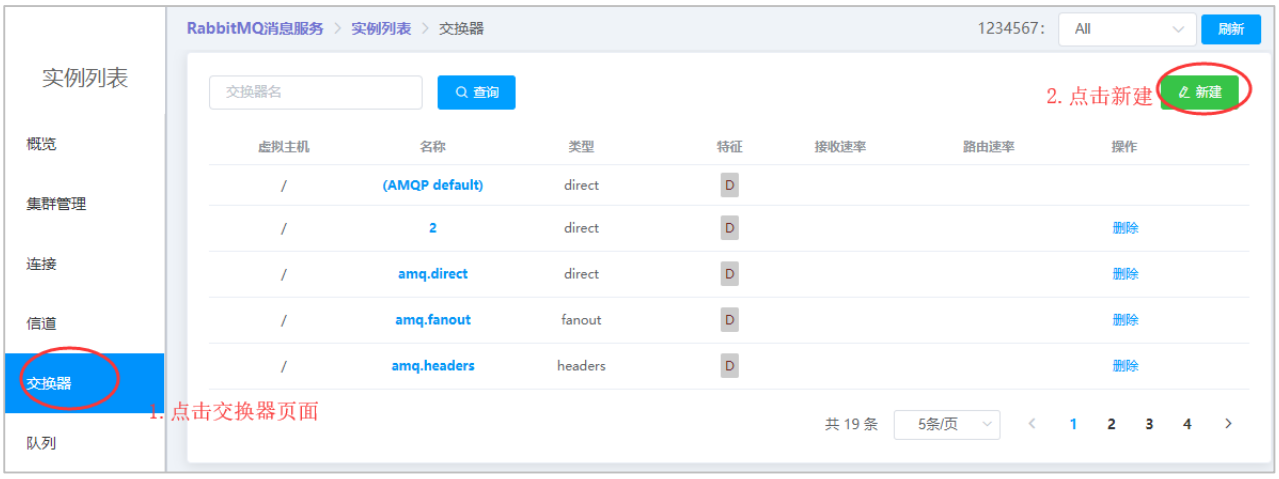


Tip：权限配置规则为正则表达式。例如 .* 表示所有权限

如'^(amq\.gen.*|amq\.default)$'可以匹配 server 生成的和默认的 exchange，'^$'不匹配任何资源。

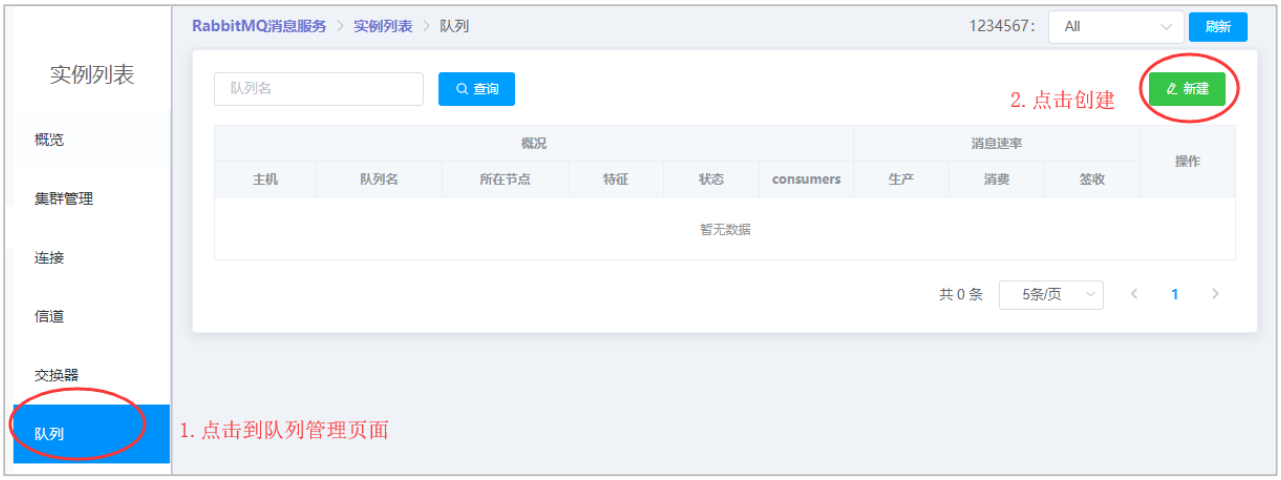
3）创建 Exchange

当创建 Vhost 时，会创建 5 个默认的 Exchange，如需要额外创建 Exchange，可进入实例列表的交换器选项卡进行新建。



4）创建 Queue

可以在代码中声明，会自动创建队列。也可以进入控制台，在实例列表的队列选项卡进行新建。



编译项目生产消费消息

引入依赖

```
1.      <dependency>
2.          <groupId>com.rabbitmq</groupId>
3.          <artifactId>amqp-client</artifactId>
```

```
        <version>5.7.0</version>
    </dependency>
```

可以通过下载 JAR 包来引入依赖。

绑定 BindingKey

代码示例：

```
import com.rabbitmq.client.BuiltinExchangeType;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import java.io.IOException;
import java.util.concurrent.TimeoutException;
public class RabbitmqBindingKey {
    private final static String EXCHANGE_NAME = "exchangeTest";
    private final static String QUEUE_NAME = "helloMQ";
    private final static String ROUTING_KEY = "test";
    public static void main(String[] args) throws IOException, TimeoutException {
        // 创建连接工厂
        ConnectionFactory factory = new ConnectionFactory();
        // 设置主机 ip
        factory.setHost("192.168.3.113");
        // 设置 amqp 的端口号
        factory.setPort(5672);
        // 设置用户名密码
        factory.setUsername("rabbitmq");
        factory.setPassword("r@bb!tMQ#3333323");
        // 设置 Vhost，需要在控制台先创建
```

```
        factory.setVirtualHost("vhost");
        //基于网络环境合理设置超时时间
        factory.setConnectionTimeout(30 * 1000);
        factory.setHandshakeTimeout(30 * 1000);
        factory.setShutdownTimeout(0);
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();
        channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.DIRECT, true);
        // 创建 ${QueueName}。Queue 可以在控制台创建，也可以用 API 创建
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);
        // Queue 与 Exchange 进行绑定，注册 BindingKeyTest
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, ROUTING_KEY);
        connection.close();
    }
}
```

完成后，可以在实例列表的交换器选项卡和队列选项卡查看结果。

生产消息

代码示例：

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
public class RabbitmqProducer {
```

```
9.      // private final static String EXCHANGE_NAME = "exchangeTest";
10.     private final static String QUEUE_NAME = "helloMQ";
11.     // private final static String ROUTING_KEY = "test";
12.     public static void main(String[] args) throws IOException, TimeoutException, Interru
    ptedException {
13.         // 创建连接工厂
14.         ConnectionFactory factory = new ConnectionFactory();
15.         // 设置主机 ip
16.         factory.setHost("192.168.3.113");
17.         // 设置 amqp 的端口号
18.         factory.setPort(5672);
19.         // 设置用户名密码
20.         factory.setUsername("username");
21.         factory.setPassword("password");
22.         // 设置 Vhost，需要在控制台先创建
23.         factory.setVirtualHost("test");
24.         //基于网络环境合理设置超时时间
25.         factory.setConnectionTimeout(30 * 1000);
26.         factory.setHandshakeTimeout(30 * 1000);
27.         factory.setShutdownTimeout(0);
28.         // 创建一个连接
29.         Connection connection = factory.newConnection();
30.         // 创建一个频道
31.         Channel channel = connection.createChannel();
32.         // 发送方消息确认
33.         // channel.confirmSelect();
34.         // 启用发送方事务机制
35.         // channel.txSelect();
```

```
        // 指定一个队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
        for (int i = 0; i < 100; i++) {
            // 发送的消息
            String message = "Hello rabbitMQ!_" + i;
            // 往队列中发送一条消息，使用默认的交换器
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));
            // 使用自定义交换器，需要在管理台预先建好，并设置 routing key
            // channel.basicPublish(EXCHANGE_NAME, ROUTING_KEY, null, message.getBytes(StandardCharsets.UTF_8));
            System.out.println(" [x] Sent '" + message + "'");
            TimeUnit.MILLISECONDS.sleep(100);
        }
        //关闭频道和连接
        channel.close();
        connection.close();
    }
}
```

消息发送后，可以进入控制台，在实例列表的队列选项卡查看消息发送状态。

消费消息

代码示例：

```
import com.rabbitmq.client.*;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.TimeoutException;
```

```
5.  public class RabbitmqConsumer {
6.      // 队列名称
7.      private final static String QUEUE_NAME = "helloMQ";
8.      public static void main(String[] args) throws IOException, TimeoutException {
9.          // 创建连接工厂
10.         ConnectionFactory factory = new ConnectionFactory();
11.         // 设置主机 ip
12.         factory.setHost("192.168.3.113");
13.         // 设置 amqp 的端口号
14.         factory.setPort(5672);
15.         // 设置用户名密码
16.         factory.setUsername("username");
17.         factory.setPassword("password");
18.         // 设置 Vhost，需要在控制台先创建
19.         factory.setVirtualHost("test");
20.         //基于网络环境合理设置超时时间
21.         factory.setConnectionTimeout(30 * 1000);
22.         factory.setHandshakeTimeout(30 * 1000);
23.         factory.setShutdownTimeout(0);
24.         Connection connection = factory.newConnection();
25.         Channel channel = connection.createChannel();
26.         //声明队列，主要为了防止消息接收者先运行此程序，队列还不存在时创建队列。
27.         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
28.         System.out.println(" [*] Waiting for messages. To exit press CTRL+C");
29.         Consumer consumer = new DefaultConsumer(channel) {
30.             @Override
31.             public void handleDelivery(String consumerTag, Envelope envelope, AMQP.Basic
    Properties properties, byte[] body) throws IOException {
```

```
            String message = new String(body, StandardCharsets.UTF_8);
            System.out.println(" [x] Received '" + message + "'");
        }
    };
    channel.basicConsume(QUEUE_NAME, true, consumer);
  }
}
```

完成上述步骤后，可以在控制台查看消费者是否启动成功。

完成以上所有步骤后，就成功接入了 RabbitMQ 服务，可以用消息队列进行消息发送和订阅了。

4 操作指导

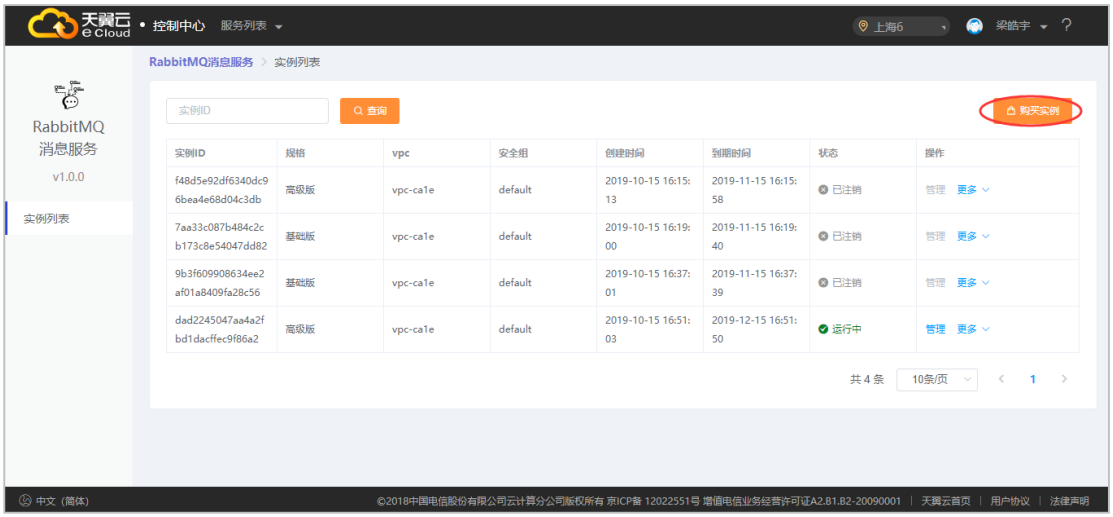
购买实例

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）在管理控制台右上角单击“地域名称”，选择区域（此处请选择与您的应用服务相同的区域）。

（4）点击“购买实例”跳转到购买页面。



（5）可以选择普通版和高级版。

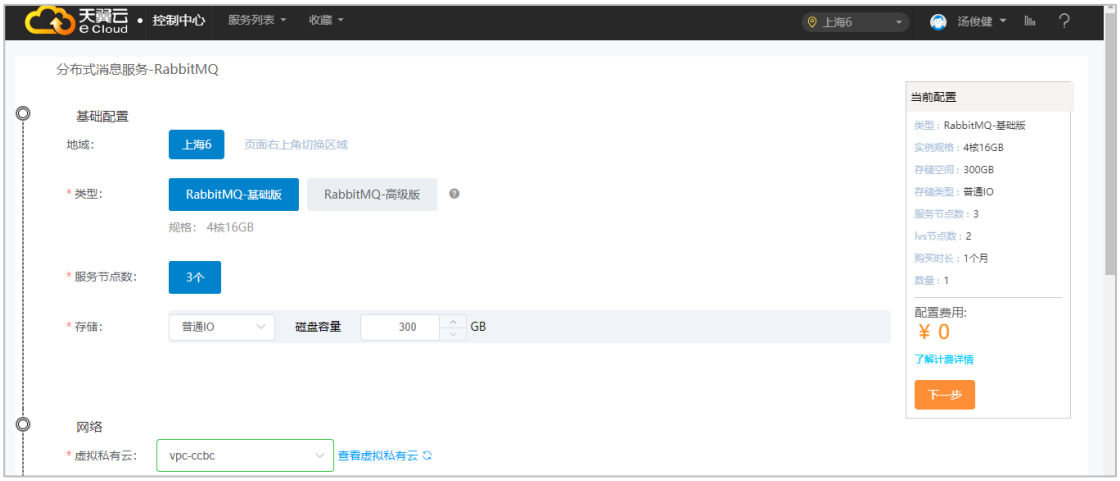
存储空间说明：目前基础版和高级版规格如下：

产品类型	**产品规格**
分布式消息服务 RabbitMQ-高级版本	三节点 8 核 16GB lvs 节点 4 核 8GB 总磁盘范围 300GB – 6000GB
分布式消息服务 RabbitMQ -基础版本	三节点 4 核 8G lvs 节点 4 核 8GB 总磁盘范围 300GB – 6000GB

在集群模式中，RabbitMQ 需要对消息持久化写入到磁盘中，因为，您在创建 RabbitMQ 实例选择存储空间时，建议根据业务消息体积预估以及镜像队列副本数量选择合适的存储

空间。镜像队列副本数最大为集群的节点数，目前都是 3。

例如：业务消息体积预估 100GB，则磁盘容量最少应为 100GB*3 + 预留磁盘大小 100GB。



（6）下单购买。

选择理解并接受《公测产品服务协议》，然后下一步订购支付完成购买。

查看实例

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）当前页面会列出所购买的 RabbitMQ 实例，并查看状态，状态说明如下

状态	说明
运行中	RabbitMQ 实例正常运行状态。 在这个状态的实例可以运行您的业务。
已关闭	RabbitMQ 实例处于故障的状态。
变更中	RabbitMQ 实例正在进行规格变更操作。
变更失败	RabbitMQ 实例处于规格变更失败状态。
暂停	RabbitMQ 专享版实例处于已冻结状态，用户可以在“更多”中续费开启冻结的 Kafka 实例。
注销	RabbitMQ 实例已经过期并关闭，需要重新购买实例。

创建和删除虚拟主机

虚拟主机（Virtual Host），类似于 Namespace 命名空间的概念，逻辑隔离，每个用户里可以创建多个 Vhost，每个 Vhost 可以创建若干个 Exchange 和 Queue。

（1）登录管理控制台，进入 RabbitMQ 管理控制台。

（2）在实例列表页在操作列，目标实例行点击“管理”。

（3）点击“集群管理”后点击“虚拟主机”到达虚拟主机管理页面，点击“新建”按钮。

（5）点击“新建”后出现以下创建，输入虚拟主机名称后点击确定。

（6）在虚拟主机管理页面，在目标虚拟主机行点击“删除” ，即可删除虚拟主机。

创建、修改和删除用户

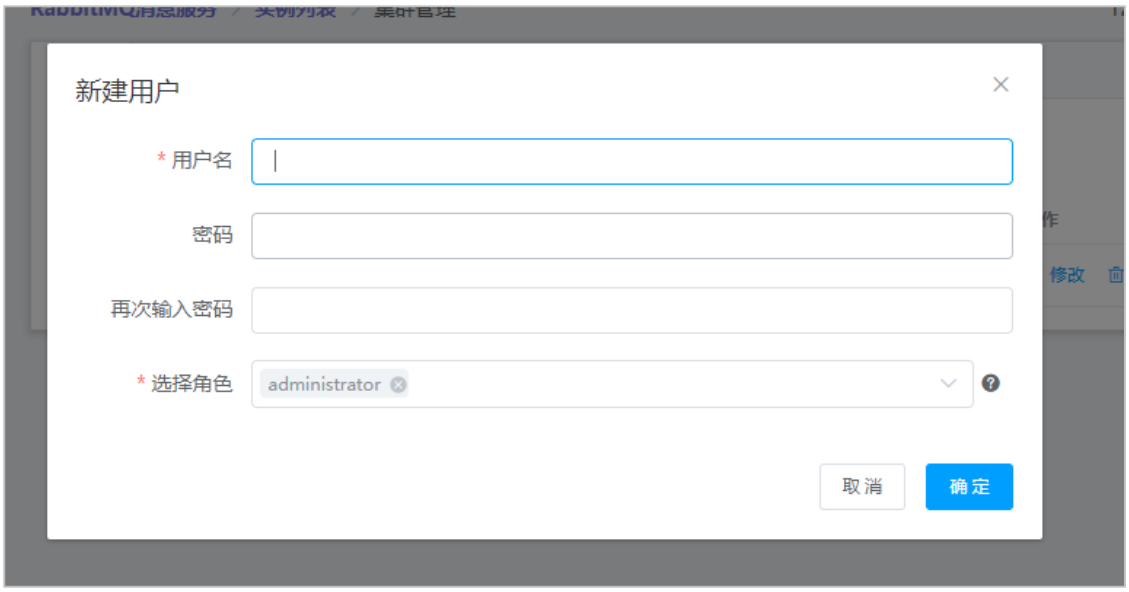
（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）在实例列表页在操作列，目标实例行点击“管理” 。

（4）点击“集群管理”后点击“用户”到达用户管理页面，点击“新建”按钮。

（5）点击“新建”后出现以下画面，输入用户密码选择角色后点击“确定”即可创建。



角色说明：

- administrator(系统管理员)：所有权限,登陆管理控制台，查看所有的信息，并且可以对用户，策略进行操作。
- monitoring(监控者)：登录,查看节点信息
- policymaker(策略者)：登陆，可以对 policy 进行管理。但无法查看节点的相关信息
- management(管理者)：仅可登陆，无法看到节点信息，也无法对策略进行管理

（6）在用户管理页面，在目标用户行点击“删除”或“修改”，即可删除或修改用户。

创建、修改和删除用户策略

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）在实例列表页在操作列，目标实例行点击“管理”。

（4）点击“集群管理”后点击“用户策略”到达用户策略管理页面，点击“新建”按钮。

（5）点击“新建”按钮后出现以下创建，选择虚拟主机，添加策略名、匹配符号，和策略内容。

Tips：

HA:

- ha-mode—— 镜像模式：all、exactly、nodes。
- ha-params—— ha-mode 为 all 时，不填；为 exactly 时,填数字; 为 nodes 时,是个节点名称数组，逗号分隔。
- ha-sync-mode——镜像同步模式：manual、automatic
- ha-promote-on-shutdown——主节点关闭后，选主策略：

when-synced：正常情况下（服务正常关闭，机器正常关机）从节点不同步时，不提升为主，否则提升为主，偏高可靠性。

Always：不管从节点是否已同步都提升为主，偏高可用性。

- ha-promote-on-failure——主节点挂掉后，选主策略：

Always：正常策略。

when-synced：无论 ha-promote-on-shutdown 设置为哪个，都不会提升未同步的从为主。

Federation：

- Federation upstream set——upstream 组名，默认为 all
- Federation upstream——联邦连接名（需开启 federation 插件）

Queues：

- Message TTL 消息过期时间：number 型(单位:ms)。
- Auto expire 队列过期时间，过期后队列自动删除：number 型(单位:ms)。
- Max length 队列能保存的最大消息数：number 型(单位:个)。
- Max length bytes 队列能保存的最大消息量：number 型(单位:字节)。
- Overflow behaviour 超过队列的最大设定值后消息接收策略：drop-head,reject-publish

  drop-head：删除头部消息,一般就是最早发送的消息，保证队列可用

  reject-publish：拒绝接受新的消息，保证消息不丢失。
- Dead letter exchange 死信交换器名称。
- Dead letter routing key 死信路由键。
- Lazy mode 队列惰性模式：default、lazy

  default：默认值，普通队列 。

  lazy：惰性队列，尽可能将消息存到磁盘中，会引起 I/O 操作比较多，内存消耗极少（有大量堆积的持久化消息建议使用）

  Master Locator 队列保存位置：client-local、min-masters、random

client-local：队列创建时所用连接的节点 。

min-masters： 集群中节点主数量最少的节点 。

random：由 rabbitmq 服务器随机指定一个节点。

Exchanges：

Alternate exchange——备份交换器，配置了该参数,如果消息无法路由到相应的队列,则路由到该交换器。

在目标用户策略所在行，点击“删除”或“修改”即可删除或修改当前用户策略。

创建、修改和删除操作策略

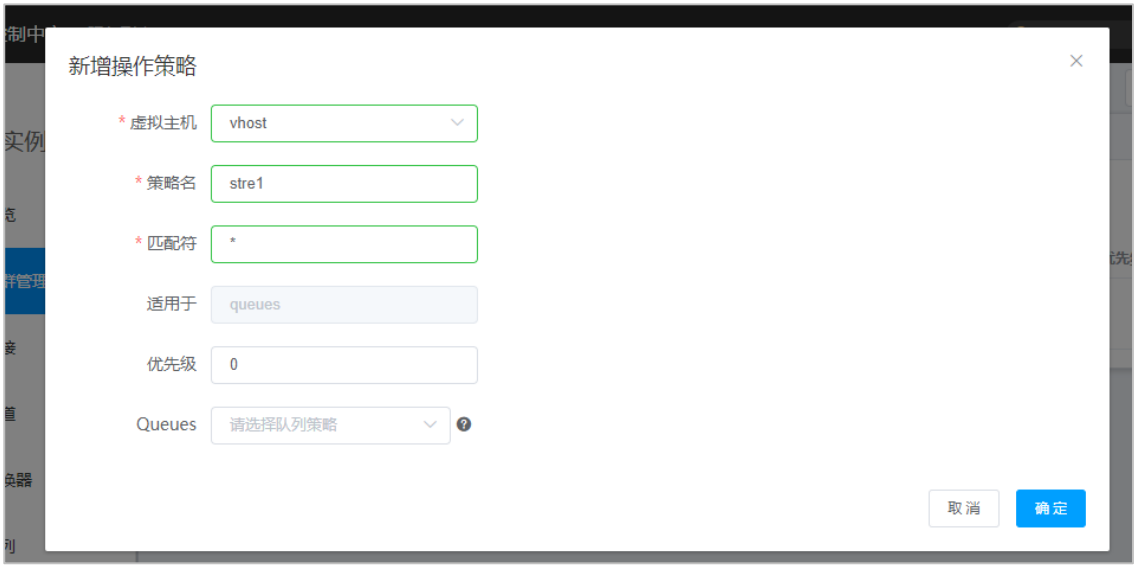
（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）在实例列表页在操作列，目标实例行点击“管理”。

（4）点击“集群管理”后点击“操作策略”到达操作策略管理页面，点击“新建”按钮。

（5）点击“新建”后出现以下界面，选择虚拟主机，添加策略名、匹配符号，和策略内容。



Queues 参数解释：

- Message TTL 消息过期时间：number 型(单位:ms)
- Auto expire 队列过期时间，过期后队列自动删除：number 型(单位:ms)
- Max length 队列能保存的最大消息数：number 型(单位:个)
- Max length bytes 队列能保存的最大消息量：number 型(单位:字节)

  Overflow behaviour 超过队列的最大设定值后消息接收策略：drop-head,reject-publish

  drop-head：删除头部消息,一般就是最早发送的消息，保证队列可用

  reject-publish：拒绝接受新的消息，保证消息不丢失

（6）在目标操作策略所在行，点击“删除”或“修改”即可删除或修改当前操作策略。

创建、修改和删除虚拟主机策略

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）在实例列表页在操作列，目标实例行点击“管理”。

（4）点击“集群管理”后点击“虚拟主机限制”到达虚拟主机页面，点击“新建”按钮。

（5）点击“新建”后出现以下界面，选择虚拟主机，添加策略内容。

有两个限制项：

max-connections：最大 TCP 连接。

max-queues：最大队列数。

（6）在目标虚拟主机限制行点击“删除”或“修改”，即可删除当前虚拟主机策略。

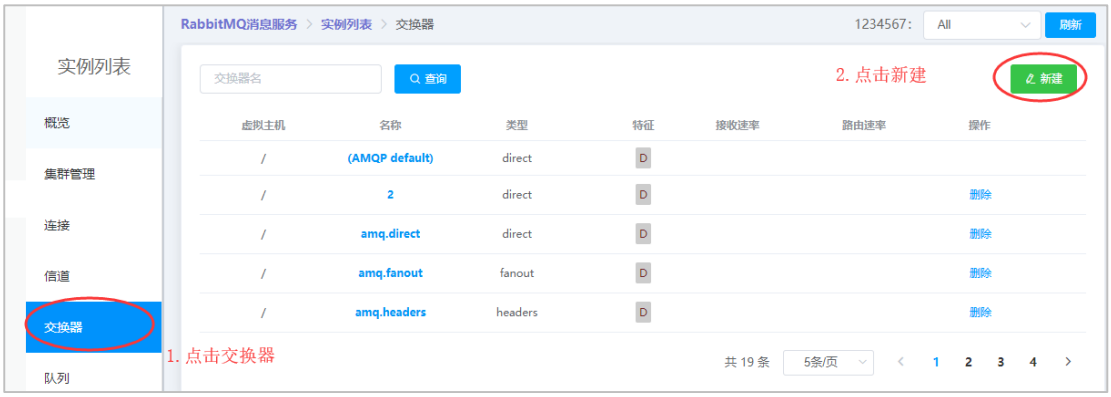
创建和删除交换器

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）在实例列表页在操作列，目标实例行点击“管理”。

（4）点击“交换器”后，点击“新建”按钮。

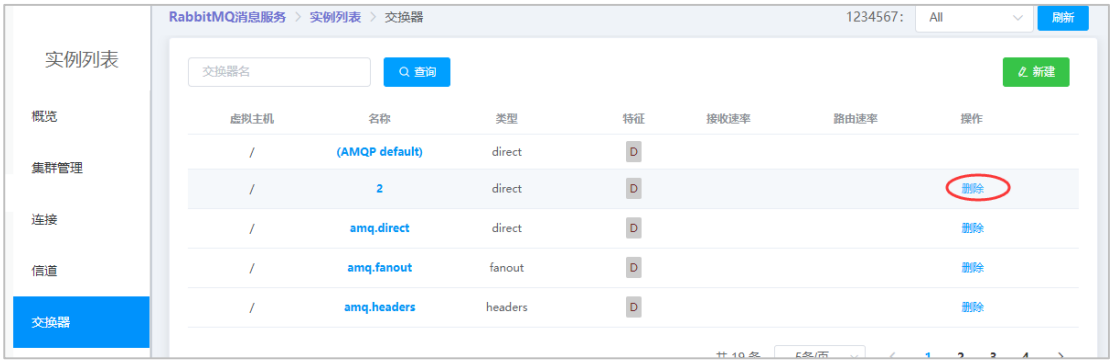


（5）点击“新建”后出现以下窗口，选择虚拟主机，添加交换器名字，选择交换器类型和其他参数，然后点击“确定”即可新建交换器。

点击新建后出现上图窗口，可以创建交换器。三种交换器解释如下：

- Direct exchange：完全根据 key 进行投递的叫做 Direct 交换机。如果 Routing key 匹配，那么 Message 就会被传递到相应的 queue 中。其实在 queue 创建时，它会自动的以 queue 的名字作为 routing key 来绑定那个 exchange。例如，绑定时设置了 Routing key 为”abc”，那么客户端提交的消息，只有设置了 key 为”abc”的才会投递到队列。
- Fanout exchange：不需要 key 的叫做 Fanout 交换机。它采取广播模式，一个消息进来时，投递到与该交换机绑定的所有队列。
- Topic exchange：对 key 进行模式匹配后进行投递的叫做 Topic 交换机。比如符号”#”匹配一个或多个词，符号”“匹配正好一个词。例如”abc.#”匹配”abc.def.ghi”，”abc.”只匹配”abc.def”。

（6）在目标交换器点击“删除”，即可删除交换器。



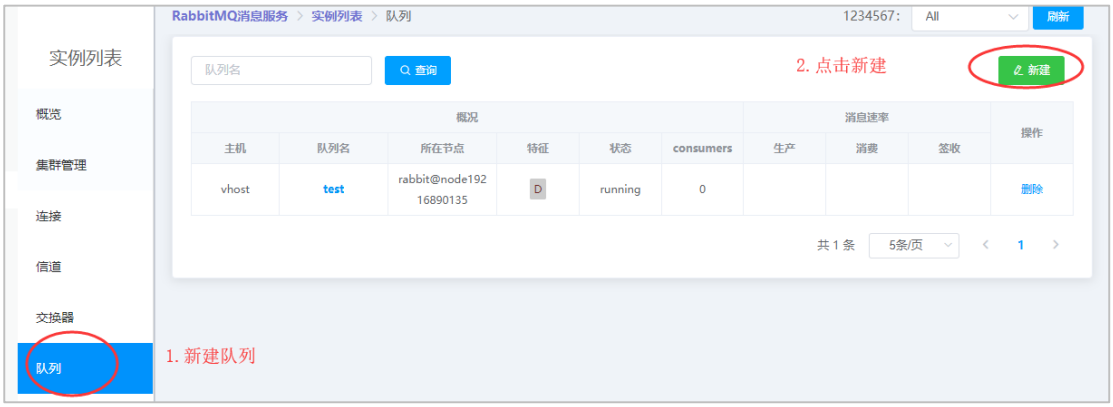
创建和删除队列

（1）登录管理控制台。

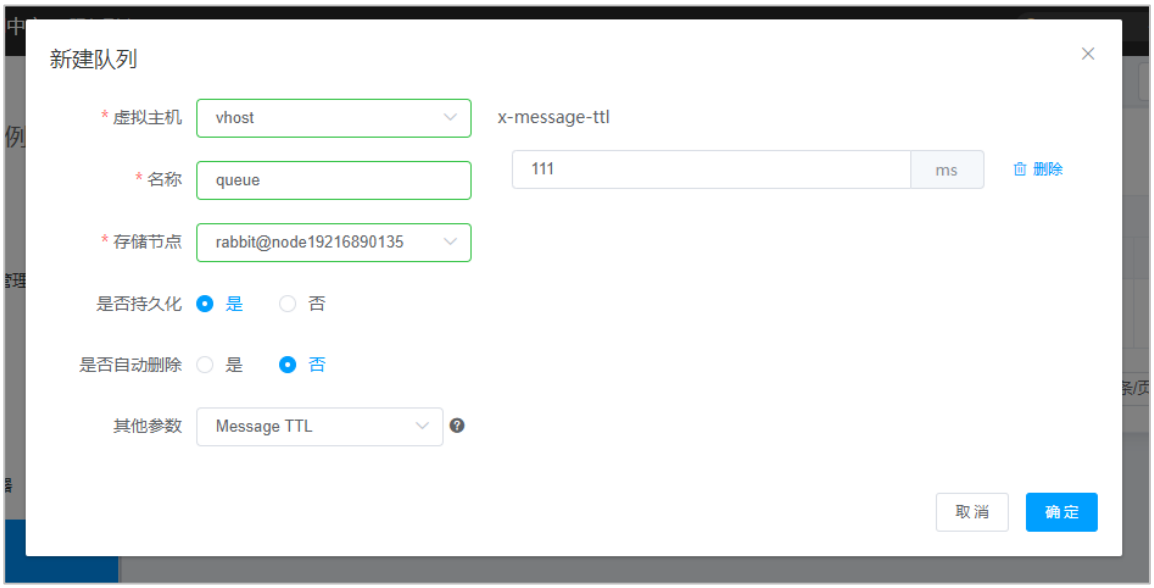
（2）进入 RabbitMQ 管理控制台。

（3）在实例列表页在操作列，目标实例行点击“管理”。

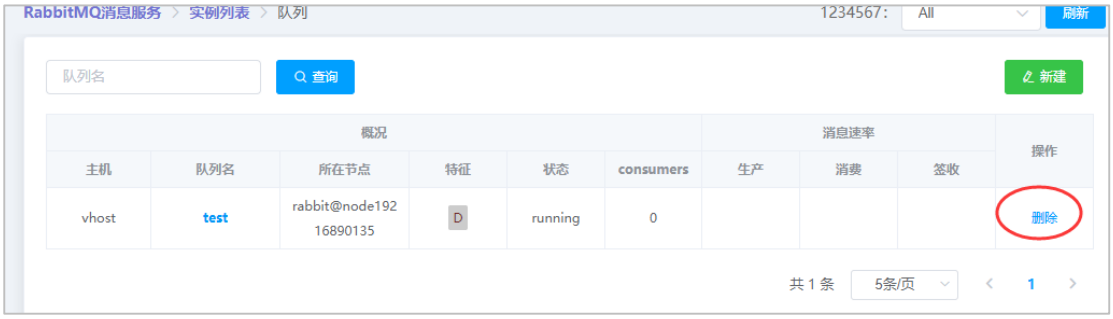
（4）点击“队列”后，点击“新建”按钮。



（5）点击“新建”后出现以下窗口，选择虚拟主机，输入队列名字，选择存储节点，然后点击确定即可创建队列。



（6）在目标队列点击删除，即可删除队列。



下载 ssl 证书

（1）登录管理控制台。

（2）进入 RabbitMQ 管理控制台。

（3）在实例列表页在操作列，目标实例行点击“管理”。

（4）点击“概览”后点击“导出服务”，点击“下载 ssl 文件”按钮即可下载 ssl 文件。

5 最佳实践

通过 ssl 认证生产与消费消息。

（1）在控制台下载当前集群 ssl 相关证书等文件

（2）运行生产者代码

替换 demo 代码的交换器、队列、ip、端口、用户名、密码、两个证书路径然后就可以运行了

```
package com.ctg.rabbitmq.server.test;

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.Connection;import

com.rabbitmq.client.ConnectionFactory;

import javax.net.ssl.KeyManagerFactory;
```

```
import javax.net.ssl.SSLContext;

import javax.net.ssl.TrustManagerFactory;

import java.io.FileInputStream;

import java.io.IOException;

import java.nio.charset.StandardCharsets;

import java.security.KeyStore;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.TimeoutException;

public class AMQPProducer {

    // 设置交换器，需要在控制台先创建

    private final static String EXCHANGE_NAME = "exchangeTest";

    // 设置队列名，需要在控制台先创建

    private final static String QUEUE_NAME = "helloMQ";

    private final static String ROUTING_KEY = "test";

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

        // 客户端证书密钥

        char[] keyPassphrase = "rabbit".toCharArray();

        KeyStore ks = KeyStore.getInstance("PKCS12");

        ks.load(new FileInputStream("F:\\tmp\\rabbit-client.keycert.p12"),
keyPassphrase);

        KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
```

```
        kmf.init(ks, keyPassphrase);
        // KeyStore 密钥
        char[] trustPassphrase = "rabbitstore".toCharArray();
        KeyStore tks = KeyStore.getInstance("JKS");
        tks.load(new FileInputStream("F:\\tmp\\rabbitstore"), trustPassphrase);
        TrustManagerFactory tmf =
TrustManagerFactory.getInstance("SunX509");
        tmf.init(tks);
        SSLContext c = SSLContext.getInstance("TLSv1.2");
        c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("192.168.90.135");
        // 输入端口（可在控制台查看）
        factory.setPort(5671);
        // 输入用户名（可在控制台创建）
        factory.setUsername("username");
        // 输入密码（可在控制台创建）
        factory.setPassword("password");
        factory.useSslProtocol(c);
        // 设置 Vhost，需要在控制台先创建
        factory.setVirtualHost("vhost");
```

```
            factory.setConnectionTimeout(30 * 1000);
            factory.setHandshakeTimeout(30 * 1000);
            factory.setShutdownTimeout(0);
            Connection connection = factory.newConnection();
            Channel channel = connection.createChannel();
            channel.queueDeclare(QUEUE_NAME, true, false, false, null);
            for (int i = 0; i < 100; i++) {
                String message = "Hello rabbitMQ!" + i;
                channel.basicPublish("", QUEUE_NAME, null,
message.getBytes(StandardCharsets.UTF_8));
                System.out.println(" [x] Sent '" + message +"'");
                TimeUnit.SECONDS.sleep(1);
            }
            //关闭频道和连接
            channel.close();
            connection.close();
        }
}
```

（3）运行消费者代码：

替换 demo 代码的队列、ip、端口、用户名、密码、两个证书路径然后就可以运行了

```
package com.ctg.rabbitmq.server.test;
```

```
import com.rabbitmq.client.*;

import java.io.IOException;

import java.nio.charset.StandardCharsets;

import java.util.concurrent.TimeoutException;

public class AMQPConsumer {

    private final static String QUEUE_NAME = "helloMQ";

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

        char[] keyPassphrase = "rabbit".toCharArray();

        KeyStore ks = KeyStore.getInstance("PKCS12");

        ks.load(new FileInputStream("F:\\tmp\\rabbit-client.keycert.p12"),
keyPassphrase);

        KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");

        kmf.init(ks, keyPassphrase);

        char[] trustPassphrase = "rabbitstore".toCharArray();

        KeyStore tks = KeyStore.getInstance("JKS");

        tks.load(new FileInputStream("F:\\tmp\\rabbitstore"), trustPassphrase);

        TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");

        tmf.init(tks);

        SSLContext c = SSLContext.getInstance("TLSv1.2");

        c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);

        // 创建连接工厂
```

```
        ConnectionFactory factory = new ConnectionFactory();
        // 设置主机 ip
        factory.setHost("192.168.90.135");
        // 设置 amqp 的端口号
        factory.setPort(5671);
        factory.useSslProtocol(c);
        // 设置用户名密码
        factory.setUsername("rabbitmq");
        factory.setPassword("r@bb!tMQ#3333323");
        // 设置 Vhost，需要在控制台先创建
        factory.setVirtualHost("vhost");
        //基于网络环境合理设置超时时间
        factory.setConnectionTimeout(30 * 1000);
        factory.setHandshakeTimeout(30 * 1000);
        factory.setShutdownTimeout(0);
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();
        //声明队列，主要为了防止消息接收者先运行此程序，队列还不存在时创建队
列。
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);
        System.out.println(" [*] Waiting for messages. To exit press CTRL+C");
```

```
        Consumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                String message = new String(body, StandardCharsets.UTF_8);
                System.out.println(" [x] Received '" + message + "'");
            }
        };
        channel.basicConsume(QUEUE_NAME, true, consumer);
    }
}
```

6 SDK 参考

说明：提供产品 SDK 参考。形式同 API 参考

Java

引入依赖：

```
<dependency>
    <groupId>com.rabbitmq</groupId>
     <artifactId>amqp-client</artifactId>
     <version>5.7.0</version>
</dependency>
```

创建连接

连接实例化

```
ConnectionFactory factory = new ConnectionFactory();

// "guest"/"guest" by default, limited to localhost connections

factory.setUsername(userName);

factory.setPassword(password);

factory.setVirtualHost(virtualHost);

factory.setHost(hostName);

factory.setPort(portNumber);

Connection conn = factory.newConnection();
```

SSL 连接实例化

```
//填写 keycert 密钥

char[] keyPassphrase = "rabbit".toCharArray();

KeyStore ks = KeyStore.getInstance("PKCS12");

//填写 keycert 密钥路径

ks.load(new FileInputStream("path\\to\\rabbit-client.keycert.p12"), keyPassphrase);

KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");

kmf.init(ks, keyPassphrase);

// 填写 keystore 密钥

char[] trustPassphrase = "rabbitstore".toCharArray();

KeyStore tks = KeyStore.getInstance("JKS");
```

```
// 填写 keystore 密钥路径
tks.load(new FileInputStream("path\\to \\rabbitstore"), trustPassphrase);
TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
tmf.init(tks);
SSLContext c = SSLContext.getInstance("TLSv1.2");
c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
ConnectionFactory factory = new ConnectionFactory();
factory.setUsername(userName);
factory.setPassword(password);
factory.setVirtualHost(virtualHost);
factory.setHost(hostName);
factory.setPort(portNumber);
factory.useSslProtocol(c);
Connection connection = factory.newConnection();
```

Factory 参数说明

```
setAutomaticRecoveryEnabled(boolean automaticRecovery)      //启用或禁用自动连
接恢复。
setChannelRpcTimeout(int channelRpcTimeout)//设置通道中 RPC 调用的继续超时。
setChannelShouldCheckRpcResponseType(boolean
channelShouldCheckRpcResponseType)          //设置为 true 时，渠道将检查响应类型
（例如
```

setClientProperties(Map<String,Object> clientProperties)//替换将在后续连接启动期间发送到服务器的客户端属性表。

setConnectionRecoveryTriggeringCondition(Predicate<ShutdownSignalException> connectionRecoveryTriggeringCondition)//允许决定是否触发自动连接恢复。

setConnectionTimeout(int timeout)//设置 TCP 连接超时时间。

setCredentialsProvider(CredentialsProvider credentialsProvider)//设置自定义凭据提供程序。

setErrorOnWriteListener(ErrorOnWriteListener errorOnWriteListener)//设置连接在尝试写入套接字时出现 IO 错误时调用的侦听器。

setExceptionHandler(ExceptionHandler exceptionHandler)//设置异常处理程序以用于新创建的连接。

setHandshakeTimeout(int timeout)//设置 AMQP0-9-1 协议握手超时。

setHeartbeatExecutor(ScheduledExecutorService executor)//设置执行程序以用于发送心跳帧。

setHost(String host)

setMetricsCollector(MetricsCollector metricsCollector)

setNetworkRecoveryInterval(int networkRecoveryInterval) //设置连接恢复间隔。

setNetworkRecoveryInterval(long networkRecoveryInterval) //设置连接恢复间隔。

setNioParams(NioParams nioParams) //使用 NIO 时设置参数。

setPassword(String password)//设置密码。

setPort(int port)//设置目标端口。

setRecoveryDelayHandler(RecoveryDelayHandler recoveryDelayHandler) //设置自动连接恢复延迟处理程序。

setRequestedChannelMax(int requestedChannelMax) //设置请求的最大频道数

setRequestedFrameMax(int requestedFrameMax)　　//设置请求的最大帧大小

setRequestedHeartbeat(int requestedHeartbeat)　　//设置请求的心跳超时。

setSaslConfig(SaslConfig saslConfig)//设置要在验证时使用的 sasl 配置

setSharedExecutor(ExecutorService executor)　　//将执行程序设置为默认情况下用于新创建连接的使用者操作调度。

setShutdownExecutor(ExecutorService executor)　　//设置执行程序以用于连接关闭。

setShutdownTimeout(int shutdownTimeout)//设置关机超时时间。

setSocketConfigurator(SocketConfigurator socketConfigurator)//设置套接字配置器。

setSocketFactory(SocketFactory factory)//设置用于创建新连接套接字的套接字工厂。

setSslContextFactory(SslContextFactory sslContextFactory)//创建 SSL 上下文的工厂。

setThreadFactory(ThreadFactory threadFactory)//设置用于实例化新线程的线程工厂。

setTopologyRecoveryEnabled(boolean topologyRecovery)//启用或禁用拓扑恢复

setTopologyRecoveryExecutor(ExecutorService topologyRecoveryExecutor)//设置执行程序以用于并行拓扑恢复。

setTopologyRecoveryFilter(TopologyRecoveryFilter topologyRecoveryFilter)//将过滤器设置为在拓扑恢复中包括/排除实体。

setTopologyRecoveryRetryHandler(RetryHandler topologyRecoveryRetryHandler)//设置重试处理程序以进行拓扑恢复。

setTrafficListener(TrafficListener trafficListener) //流量侦听器已通知入站和出站Command。

setUri(String uriString)//设置 AMQP URI 中的字段的便捷方法：主机，端口，用户名，密码和虚拟主机。

setUri(URI uri)//设置 AMQP URI 中的字段的便捷方法：主机，端口，用户名，密码和虚拟主机。

setUsername(String username)//设置用户名。

setVirtualHost(String virtualHost)//设置虚拟主机。

setWorkPoolTimeout(int workPoolTimeout)//工作池排队的超时时间（以毫秒为单位）。

useBlockingIo()//使用阻塞 IO 与服务器进行通信。

useNio()//使用非阻塞 IO（NIO）与服务器进行通信。

useSslProtocol()//使用默认的 TLS 协议集和信任的 TrustManager 配置 TLS 的便捷方法。

useSslProtocol(String protocol)//使用提供的协议和非常信任的 TrustManager 来配置 TLS 的便捷方法。

useSslProtocol(String protocol, TrustManager trustManager)//配置 TLS 的便捷方

法。

useSslProtocol(SSLContext context)//使用初始化设置 TLS SSLContext。

关闭连接

```
connection.close();
```

创建信道

```
Channel channel = connection.createChannel();
```

关闭信道

```
channel.close();
```

生产消息

```
String message = "Hello rabbitMQ!" + i;
channel.basicPublish("交换器名称", "路由键", null, message.getBytes(StandardCharsets.UTF_8));
```

消费消息

```
Consumer consumer = new DefaultConsumer(channel) {
@Override
public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body) throws IOException {
        String message = new String(body, StandardCharsets.UTF_8);
        System.out.println(" [x] Received '" + message + "'");
        }
 };
channel.basicConsume("队列名称", true, consumer);
```

代码示例

普通方式：

```
ConnectionFactory factory = new ConnectionFactory();
```

```
// 设置连接地址，可在控制台概览->节点查看，推荐用 vip，vpc 内可使用 vpcIp
factory.setHost("vip_address,ex:192.168.90.2");
factory.setPort(5671);
// 设置用户名密码，需要在控制台先创建
factory.setUsername("username");
factory.setPassword("password");
// 设置 Vhost，需要在控制台先创建
factory.setVirtualHost("/");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();
for (int i = 0; i < 100; i++) {
String message = "Hello rabbitMQ!" + i;
// 填写交换器名称 和 路由键
channel.basicPublish("交换器名称", "路由键", null,
message.getBytes(StandardCharsets.UTF_8));
    System.out.println(" [x] Sent '" + message +"'");
    TimeUnit.SECONDS.sleep(1);
    }
//关闭频道和连接
channel.close();
connection.close();
```

SSL 连接方式：

```
// 设置 keycert 密码，默认 rabbit
char[] keyPassphrase = "rabbit".toCharArray();
KeyStore ks = KeyStore.getInstance("PKCS12");
// 设置 keycert 路径，可在控制台下载
ks.load(new FileInputStream("F:\\tmp\\ssl\\client\\rabbit-client.keycert.p12"),
keyPassphrase);
KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
kmf.init(ks, keyPassphrase);
// 设置 keystore
char[] trustPassphrase = "rabbitstore".toCharArray();
KeyStore tks = KeyStore.getInstance("JKS");
// 设置 keystore 路径
tks.load(new FileInputStream("F:\\tmp\\ssl\\keystore\\rabbitstore"),
trustPassphrase);
TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
tmf.init(tks);
SSLContext c = SSLContext.getInstance("TLSv1.2");
c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
ConnectionFactory factory = new ConnectionFactory();
```

```
// 设置连接地址，可在控制台概览->节点查看，推荐用 vip，vpc 内可使用 vpcIp
factory.setHost("vip_address,ex:192.168.90.2");
factory.setPort(5671);
// 设置用户名密码，需要在控制台先创建
factory.setUsername("username");
factory.setPassword("password");
// 设置 Vhost，需要在控制台先创建
factory.setVirtualHost("/");
factory.setConnectionTimeout(30 * 1000);
factory.setHandshakeTimeout(30 * 1000);
factory.setShutdownTimeout(0);
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();
        for (int i = 0; i < 100; i++) {
            String message = "Hello rabbitMQ!" + i;
            channel.basicPublish("", "test", null,
            message.getBytes(StandardCharsets.UTF_8));
            System.out.println(" [x] Sent '" + message +"'");
            TimeUnit.SECONDS.sleep(1);
        }
        //关闭频道和连接
```

```
        channel.close();

        connection.close();
```

C\C++

以下例子使用 rabbitmq-c 开源 C、C++客户端 https://github.com/alanxz/rabbitmq-c

以下为各个场景使用例子：

examples.zip

创建连接

```
  char const *hostname;

  int port, status;

  amqp_socket_t *socket;

  amqp_connection_state_t conn;

  conn = amqp_new_connection();

  socket = amqp_ssl_socket_new(conn);

  amqp_ssl_socket_set_verify_peer(socket, 0);

  amqp_ssl_socket_set_verify_hostname(socket, 0);

  status = amqp_socket_open(socket, hostname, port);

  die_on_amqp_error(amqp_login(conn, "/", 0, 131072, 0,

AMQP_SASL_METHOD_PLAIN, "guest", "guest"),"Logging in");
```

关闭连接

```
die_on_amqp_error(amqp_connection_close(conn, AMQP_REPLY_SUCCESS),

"Closing connection");

die_on_error(amqp_destroy_connection(conn), "Ending connection");
```

创建信道

```
amqp_channel_open(conn, 1);

die_on_amqp_error(amqp_get_rpc_reply(conn), "Opening channel");
```

关闭信道

```
die_on_amqp_error(amqp_channel_close(conn, 1, AMQP_REPLY_SUCCESS),

"Closing channel");
```

生产消息

```
amqp_basic_properties_t props;

props._flags = AMQP_BASIC_CONTENT_TYPE_FLAG |

AMQP_BASIC_DELIVERY_MODE_FLAG;

props.content_type = amqp_cstring_bytes("text/plain");

props.delivery_mode = 2; /* persistent delivery mode */

die_on_error(amqp_basic_publish(conn,1,amqp_cstring_bytes(exchange),

amqp_cstring_bytes(routingkey),0,0,&props,amqp_cstring_bytes(messagebody)),

                    "Publishing");
```

消费消息

```
amqp_queue_declare_ok_t *r = amqp_queue_declare(conn, 1, amqp_empty_bytes,
```

```
0, 0, 0, 1,amqp_empty_table);
die_on_amqp_error(amqp_get_rpc_reply(conn), "Declaring queue");
queuename = amqp_bytes_malloc_dup(r->queue);
    if (queuename.bytes == NULL) {
        fprintf(stderr, "Out of memory while copying queue name");
        return 1;
    }
```

使用示例

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <amqp_ssl_socket.h>
#include <amqp_framing.h>
#include "utils.h"
int main(int argc, char const *const *argv)
{
  char const *hostname;
  int port, status;
  char const *exchange;
  char const *routingkey;
```

```
char const *messagebody;
amqp_socket_t *socket;
amqp_connection_state_t conn;
if (argc < 6) {
  fprintf(stderr, "Usage: amqps_sendstring host port exchange routingkey "
          "messagebody [cacert.pem [verifypeer] [verifyhostname] "
          "[key.pem cert.pem]]\n");
  return 1;
}
hostname = argv[1];
port = atoi(argv[2]);
exchange = argv[3];
routingkey = argv[4];
messagebody = argv[5];

conn = amqp_new_connection();
socket = amqp_ssl_socket_new(conn);
if (!socket) {
  die("creating SSL/TLS socket");
}
amqp_ssl_socket_set_verify_peer(socket, 0);
```

```
amqp_ssl_socket_set_verify_hostname(socket, 0);
if (argc > 6) {
  int nextarg = 7;
  status = amqp_ssl_socket_set_cacert(socket, argv[6]);
  if (status) {
    die("setting CA certificate");
  }
  if (argc > nextarg && !strcmp("verifypeer", argv[nextarg])) {
    amqp_ssl_socket_set_verify_peer(socket, 1);
    nextarg++;
  }
  if (argc > nextarg && !strcmp("verifyhostname", argv[nextarg])) {
    amqp_ssl_socket_set_verify_hostname(socket, 1);
    nextarg++;
  }
  if (argc > nextarg + 1) {
    status =
        amqp_ssl_socket_set_key(socket, argv[nextarg + 1], argv[nextarg]);
    if (status) {
      die("setting client cert");
    }
```

```
    }
  }
  status = amqp_socket_open(socket, hostname, port);
  if (status) {
    die("opening SSL/TLS connection");
  }
  die_on_amqp_error(amqp_login(conn, "/", 0, 131072, 0,
AMQP_SASL_METHOD_PLAIN, "guest", "guest"),"Logging in");
  amqp_channel_open(conn, 1);
  die_on_amqp_error(amqp_get_rpc_reply(conn), "Opening channel");
  {
    amqp_basic_properties_t props;
    props._flags = AMQP_BASIC_CONTENT_TYPE_FLAG |
AMQP_BASIC_DELIVERY_MODE_FLAG;
    props.content_type = amqp_cstring_bytes("text/plain");
    props.delivery_mode = 2; /* persistent delivery mode */
    die_on_error(amqp_basic_publish(conn,1,amqp_cstring_bytes(exchange),

amqp_cstring_bytes(routingkey),0,0,&props,amqp_cstring_bytes(messagebody)),
                 "Publishing");
  }
```

```
  die_on_amqp_error(amqp_channel_close(conn, 1, AMQP_REPLY_SUCCESS),
"Closing channel");
  die_on_amqp_error(amqp_connection_close(conn, AMQP_REPLY_SUCCESS),
"Closing connection");
  die_on_error(amqp_destroy_connection(conn), "Ending connection");
  return 0;
}
```

7 常见问题

计费类

1、支持哪些付费方式？

支持包年包月

2、收费依据有哪些？

根据规格和磁盘容量收费

3、产品规格可选择哪些？

可选高级版和基础版

4、高级版磁盘容量可选范围是多少？

高级版磁盘容量为 300GB-6000GB

5、基础版磁盘容量可选范围是多少？

基础版磁盘容量为 300GB-6000GB

购买类

1、到期后如何续费？

在集群列表中点击“续费”，进入购买时长页面，购买成功后自动续费

2、支持自营资源池与合营资源池吗？

目前支持自营资源池

3、如何退订实例？

在集群列表中点击“退订”，进入相关页面退订

4、如何变更实例？

暂不支持变更

5、如何选择磁盘空间？

存储空间说明：在集群模式中，RabbitMQ 需要对消息持久化写入到磁盘中，因为，您在创建 RabbitMQ 实例选择存储空间时，建议根据业务消息体积预估以及镜像队列副本数量选择合适的存储空间。镜像队列副本数最大为集群的节点数，目前都是 3。

例如：业务消息体积预估 100GB，则磁盘容量最少应为 100GB*3 + 预留磁盘大小 100GB。

系统类

1、无法被路由的消息，去了哪里？

如果没有任何设置，无法路由的消息会被直接丢弃。

无法路由的情况：Routing key 不正确。

解决方案：

（1）使用 mandatory=true 配合 ReturnListener，实现消息回发。

（2）声明交换机时，指定备份交换机。

2、多个消费者监听一个队列时，消息如何分发？

（1）Round-Robin（轮询）

默认的策略，消费者轮流、平均地收到消息。

（2）Fair dispatch（公平分发）

如果要实现根据消费者地处理能力来分发消息，给空闲地消费者发送更多消息，可以用 basicQos(int prefetch_count)来设置。prefetch_count 地含义：当消费者有多条消息没有响应 ACK 时，不再给这个消费者发送消息。

3、消息在什么时候会变成 Dead Letter(死信)？

（1）消息被拒绝并且没有设置重新入队：(NACK || Reject) && requeue == false

（2）消息过期(消息或者队列的 TTL 设置)

（3）消息堆积，并且队列达到最大长度，先入队的消息编程 DL。

解决方案：可以在声明队列时，指定一个 Dead Letter Exchange，来实现 Dead Letter 的转发，保证消息不会丢失。

4、如何进行消息持久化？

所谓持久化，就是 RabbitMQ 将内存中的数据（比如交换机、队列、消息等）固化到磁盘，以防止异常情况的发生时造成数据丢失。

RabbitMQ 持久化分为：

（1）交换机的持久化

在创建 Exchange 时设置 durable 参数参数。

```
channel.exchangeDeclare(EXCHANGE_NAME, "direct", true);
```

（2）队列的持久化

同样也是设置设置 durable 参数。

持久化的队列会存盘，在服务器重启的时候可以保证不丢失相关信息。

```
channel.queueDeclare(QUEUE_NAME, true, false, false, null);
```

（3）消息的持久化

即使交换机、队列持久化不会因为重启丢失，但是存储在队列中的消息仍然会丢失。

解决的办法就是设置消息的投递模式为 2，即代表持久化(JAVA)。

理论上，可以将所有的消息都设置为持久化，但是这会严重影响 RabbitMQ 性能，因为写入到磁盘的速度可比写入到内存的速度慢非常多。因此，在选择是否要持久化消息时，需要在可靠性和吞吐量之间做一个权衡。

8 相关协议

产品服务协议

https://www.ctyun.cn/h5/home/protocol/10008634

产品服务等级协议

https://www.ctyun.cn/h5/home/protocol/10403554