ADMQ-RabbitMQ认证功能说明和使用

功能描述

基于ADMQ Token认证方式实现RabbitMQ客户端的用户名密码认证，客户端发送和接收消息时需要提供正确的认证信息，否则ADMQ会断开连接。除了客户端连接权限，还可在界面上设置客户端拥有哪些vhost的读写权限。

使用方式

部署ADMQ并加载RabbitMQ插件，然后修改节点配置，添加认证配置即可。具体流程如下：

部署ADMQ并创建集群

解压admq-manager，执行bin/admq-manager start 启动。

登录管控台：https://ip:12306，默认用户名密码：admq/11111111

在【系统配置】-【软件包管理】中上传admq软件包。

在【系统配置】-【服务器管理】中添加服务器，测试环境添加一台即可。

在【集群管理】中创建集群，选择单机模式，选择软件包和服务器。

在【集群列表】-【节点】中部署节点。

加载RabbitMQ插件并开启认证

在【集群列表】-【插件】中开启AMQP插件。

在【集群列表】-【节点】-【更多】-【配置】中添加配置项：

amqpAuthenticationEnabled=true

之后点击【更新配置文件】，配置会同步到软件包目录。

启动节点

在【集群列表】-【节点】中启动节点，启动成功后【节点状态】会变成【运行中】

创建用户

在【用户管理】中创建新用户。

创建vhost等资源

在【插件管理】–【RabbitMQ插件】中创建以下内容：

租户：amqp-data，**这个为固定值，不要改成其他的**

虚拟主机：vhost01，可自定义

交换机：ex01，可自定义

队列：qu01，可自定义

路由：route01，可自定义

分配权限

在【插件管理】–【RabbitMQ插件】–【虚拟主机】中，点击【分配权限】，选择刚才的用户分配读写权限。

测试验证

系统是基于token认证的，token需要放到rabbitmq客户端的password字段。

其中的token是【用户管理】对应用户的【令牌】

测试代码如下：

Java | 复制代码

```java
public class SimpleTest {

    private Connection connect() throws Exception {
        String token = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJSM0I5MzBDdzRZN252NEFNIn0.a-C9ghFZCQHbQVjwp6_K823BncPQzZUJTpTwZ1pRTDg";
        token = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJSM0I5MzBDdzRZN252NEFNIn0.a-C9ghFZCQHbQVjwp6_K823BncPQzZUJTpTwZ1pRTDg";
//        token = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJnSzcyakw1ME5NMDI3QTAyIn0.pmWYBDrVdgWN-zIP_dt1NTelaamKL57pL1G7gbc8gyQ"; // 生产
//        token = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJYTjRYMTdXT3JrT0dRSkRZIn0.4aVd290M3mKd4XZHE8pQGQ3-4iZwrNyZW6Sujdm-Oac"; // 消费

        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("ip");
        factory.setVirtualHost("vhost1");
        factory.setUsername("admq");
        factory.setPassword("token:" + token);
        factory.setPort(5672);
        return factory.newConnection();
    }

    @Test
    public void testSendMessage() throws Exception {
        String exchange = "ex01";
        String queue = "qu01";

        Connection conn = connect();
        Channel channel = conn.createChannel();
        channel.exchangeDeclare(exchange, BuiltinExchangeType.DIRECT);
        channel.queueDeclare(queue, true, false, false, null);
        channel.queueBind(queue, exchange, "");

        channel.basicPublish(exchange, "", null, "123".getBytes(StandardCharsets.UTF_8));

        channel.close();
        conn.close();
    }

    @Test
    public void testReceiveMessage() throws Exception {
        String queue = "qu01";

        Connection conn = connect();
        Channel channel = conn.createChannel();
```

```
41
42          channel.basicConsume(queue, false, new DefaultConsumer(channel) {
43
44              @Override
45              public void handleDelivery(String consumerTag, Envelope envelo
    pe, AMQP.BasicProperties properties,
46                                         byte[] body) throws IOException {
47                  System.out.println("receive " + new String(body));
48              }
49          });
50
51          Thread.sleep(10000);
52
53          channel.close();
54          conn.close();
55      }
56
57      @Test
58      public void testSendAndReceiveMessage() throws Exception {
59          String queue = "qu01";
60          String exchange = "ex01";
61
62          Connection conn = connect();
63          Channel channel = conn.createChannel();
64
65          channel.basicConsume(queue, true, new DefaultConsumer(channel) {
66
67              @Override
68              public void handleDelivery(String consumerTag, Envelope envelo
    pe, AMQP.BasicProperties properties,
69                                         byte[] body) throws IOException {
70                  System.out.println("receive " + new String(body));
71              }
72          });
73
74          channel.basicPublish(exchange, "", null, "456".getBytes(StandardCh
    arsets.UTF_8));
75
76          Thread.sleep(10000);
77
78          channel.close();
79          conn.close();
80      }
81  }
```