SpringBoot 整合 ActiveMQ

笔记本:	activeMQ		
创建时间:	2023/12/11 13:38	**更新时间:**	2024/5/2 18:41

SpringBoot 整合 ActiveMQ

多个queue和多个topic实现，及前端VUE使用Stompjs接收

一、ActiveMQ简介

1). ActiveMQ

ActiveMQ是Apache所提供的一个开源的消息系统，完全采用Java来实现，

因此，它能很好地支持J2EE提出的JMS（Java Message Service,即Java消息服务）规范。

JMS是一组Java应用程序接口，它提供消息的创建、发送、读取等一系列服务。

JMS提供了一组公共应用程序接口和响应的语法，类似于Java数据库的统一访问接口JDBC,

它是一种与厂商无关的API，使得Java程序能够与不同厂商的消息组件很好地进行通信。

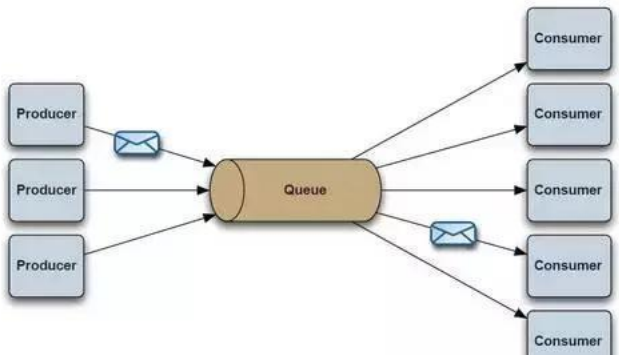
2). Java Message Service(JMS)

JMS支持两种消息发送和接收模型。

一种称为P2P(Ponit to Point)模型，即采用点对点的方式发送消息。

P2P模型是基于队列的，消息生产者发送消息到队列，消息消费者从队列中接收消息，

队列的存在使得消息的异步传输称为可能，P2P模型在点对点的情况下进行消息传递时采用。

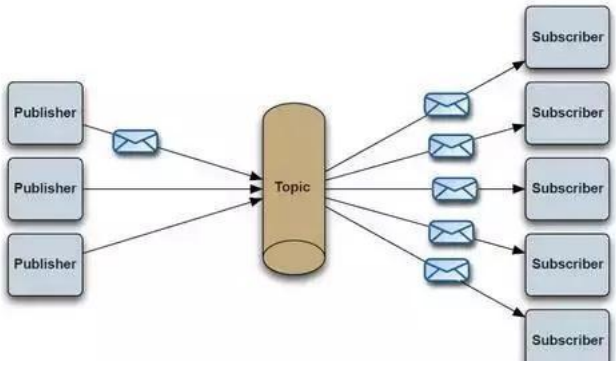


另一种称为Pub/Sub(Publish/Subscribe，即发布-订阅)模型，发布-订阅模型定义了如何向一个内容节点发

这个内容节点称为topic(主题)。主题可以认为是消息传递的中介，消息发布这将消息发布到某个主题，

而消息订阅者则从主题订阅消息。主题使得消息的订阅者与消息的发布者互相保持独立，

不需要进行接触即可保证消息的传递，发布-订阅模型在消息的一对多广播时采用。



3). JMS术语

Provider/MessageProvider：生产者

Consumer/MessageConsumer：消费者

PTP：Point To Point，点对点通信消息模型

Pub/Sub：Publish/Subscribe，发布订阅消息模型

Queue：队列，目标类型之一，和PTP结合

Topic：主题，目标类型之一，和Pub/Sub结合

ConnectionFactory：连接工厂，JMS用它创建连接

Connnection：JMS Client到JMS Provider的连接

Destination：消息目的地，由Session创建

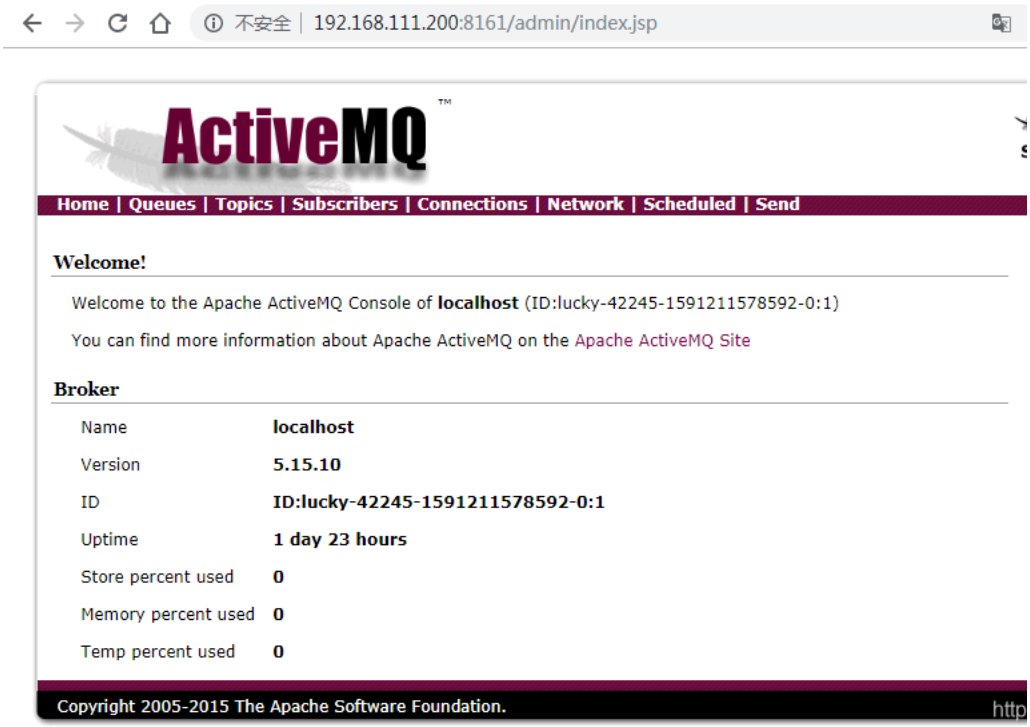
Session：会话，由Connection创建，实质上就是发送、接受消息的一个线程，因此生产者、消费者都

二、Windows安装 ActiveMq

1、下载地址：http://activemq.apache.org/download-archives.html，

本文用的是windows版的5.15.10版本，下载下来是压缩包。apache-activemq-5.15.10-bin.tar.gz

2、将压缩包解压一个到目录下，CMD进入到解压目录下的bin目录下，执行 activemq.bat start 启动。
http://localhost:8161/admin（用户名和密码默认为admin），则启动成功。



三、SpringBoot 集成 ActiveMQ

1. 创建一个springboot项目，添加依赖

```
<!--ActiveMq-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-activemq</artifactId>
    <!-- <version>2.1.0.RELEASE</version> -->
</dependency>
<!--消息队列连接池-->
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-pool</artifactId>
    <!-- <version>5.15.7</version> -->
</dependency>
```

2、application.yml文件的配置

```
server:
    port: 8090

spring:
    activemq:
        broker-url: tcp://192.168.111.200:61616
        user: admin
        password: admin
        close-timeout: 15s # 在考虑结束之前等待的时间
        in-memory: true # 默认代理URL是否应该在内存中。如果指定了显式代理，则忽略此值。
        non-blocking-redelivery: false # 是否在回滚回滚消息之前停止消息传递。这意味着当启用此命令时，
        send-timeout: 0 # 等待消息发送响应的时间。设置为0等待永远。
```

```
        queue-name: active.queue
                                        topic-name: active.topic.name.model

# packages:
# trust-all: true #不配置此项，会报错
    pool:
        enabled: true
        max-connections: 10 #连接池最大连接数
        idle-timeout: 30000 #空闲的连接过期时间，默认为30秒

# jms:
# pub-sub-domain: true #默认情况下activemq提供的是queue模式，若要使用topic模式需要配置此项

#其他配置项：
# 是否信任所有包
#spring.activemq.packages.trust-all=
# 要信任的特定包的逗号分隔列表（当不信任所有包时）
#spring.activemq.packages.trusted=
# 当连接请求和池满时是否阻塞。设置false会抛“JMSException异常”。
#spring.activemq.pool.block-if-full=true
# 如果池仍然满，则在抛出异常前阻塞时间。
#spring.activemq.pool.block-if-full-timeout=-1ms
# 是否在启动时创建连接。可以在启动时用于加热池。
#spring.activemq.pool.create-connection-on-startup=true
# 是否用Pooledconnectionfactory代替普通的ConnectionFactory。
#spring.activemq.pool.enabled=false
# 连接过期超时。
#spring.activemq.pool.expiry-timeout=0ms
# 连接空闲超时
#spring.activemq.pool.idle-timeout=30s
# 连接池最大连接数
#spring.activemq.pool.max-connections=1
# 每个连接的有效会话的最大数目。
#spring.activemq.pool.maximum-active-session-per-connection=500
# 当有"JMSException"时尝试重新连接
#spring.activemq.pool.reconnect-on-exception=true
# 在空闲连接清除线程之间运行的时间。当为负数时，没有空闲连接驱逐线程运行。
#spring.activemq.pool.time-between-expiration-check=-1ms
# 是否只使用一个MessageProducer
#spring.activemq.pool.use-anonymous-producers=true
```

3、启动类增加 @EnableJms 注解

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;

@SpringBootApplication
@EnableJms //启动消息队列
public class SpringbootActivemqApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootActivemqApplication.class, args);
    }
}
```

4、初始化和配置 ActiveMQ 的连接

```java
import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;
import org.apache.activemq.command.ActiveMQTopic;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.config.JmsListenerContainerFactory;
import org.springframework.jms.config.SimpleJmsListenerContainerFactory;
import org.springframework.jms.core.JmsMessagingTemplate;
import javax.jms.ConnectionFactory;
import javax.jms.Queue;
import javax.jms.Topic;

@Configuration
public class BeanConfig {

    @Value("${spring.activemq.broker-url}")
    private String brokerUrl;

    @Value("${spring.activemq.user}")
    private String username;

    @Value("${spring.activemq.topic-name}")
    private String password;

    @Value("${spring.activemq.queue-name}")
    private String queueName;

    @Value("${spring.activemq.topic-name}")
    private String topicName;

    @Bean(name = "queue")
    public Queue queue() {
        return new ActiveMQQueue(queueName);
    }

    //如果有多个queue主题，则继续增加@Bean，@Bean内的name可省略,省略后name为方法名（"queue2"），
    //注意：后续@Autowired注解“public Queue queue2;”的变量名为此处的name("queue2");
    //队列名称为"queueName2"，后续的队列消费时destination要和此队列名称一致。
    @Bean(name = "queue2")
    public Queue queue2() {
        return new ActiveMQQueue("queueName2");
    }

    //其他的queue主题定义
    //...

    @Bean(name = "topic")
    public Topic topic() {
        return new ActiveMQTopic(topicName);
    }

    @Bean(name = "topic2")
    public Topic topic2() {
        return new ActiveMQTopic("topicName2");
    }
```

```
    //其他的topic主题定义
                            //...

    @Bean
    public ConnectionFactory connectionFactory(){
        return new ActiveMQConnectionFactory(username, password, brokerUrl);
    }

    @Bean
    public JmsMessagingTemplate jmsMessageTemplate(){
        return new JmsMessagingTemplate(connectionFactory());
    }

    // 在Queue模式中，对消息的监听需要对containerFactory进行配置
    @Bean("queueListener")
    public JmsListenerContainerFactory<?> queueJmsListenerContainerFactory(ConnectionFactc
        SimpleJmsListenerContainerFactory factory = new SimpleJmsListenerContainerFactory()
        factory.setConnectionFactory(connectionFactory);
        factory.setPubSubDomain(false);
        return factory;
    }

    //在Topic模式中，对消息的监听需要对containerFactory进行配置
    @Bean("topicListener")
    public JmsListenerContainerFactory<?> topicJmsListenerContainerFactory(ConnectionFactc
        SimpleJmsListenerContainerFactory factory = new SimpleJmsListenerContainerFactory()
        factory.setConnectionFactory(connectionFactory);
        factory.setPubSubDomain(true);
        return factory;
    }
}
```

5、生产者（queue 和 topic）

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import javax.jms.Destination;
import javax.jms.Queue;
import javax.jms.Topic;

@RestController
public class ProducerController{
   @Autowired
   private JmsMessagingTemplate jmsMessagingTemplate;

   @Autowired
   private Queue queue;

   @Autowired
   private Queue queue2;

   //更多的Queue...
   @Autowired
   private Topic topic;

   //更多的topic...
```

```
    @PostMapping("/queue/test")
    public String sendQueue(@RequestBody String str) {
        this.sendMessage(this.queue, str);
        return "success";
    }

    @PostMapping("/queue2/test")
    public String sendQueue(@RequestBody String str) {
        this.sendMessage(this.queue2, str);
        return "queue2 success";
    }

    @PostMapping("/topic/test")
    public String sendTopic(@RequestBody String str) {
        this.sendMessage(this.topic, str);
        return "success";
    }

    // 发送消息，destination是发送到的队列，message是待发送的消息
    private void sendMessage(Destination destination, final String message){
        jmsMessagingTemplate.convertAndSend(destination, message);
    }
}
```

6、Queue模式的消费者

```
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class QueueConsumerListener{

    //queue模式的消费者
    @JmsListener(destination="${spring.activemq.queue-name}", containerFactory="queueListener")
    public void readActiveQueue(String message) {
        System.out.println("queue接受到：" + message);
    }

    //queue2模式的消费者,
    //destination为queue主题，和ActiveMQBeanConfig定义中的@Bean中的"new ActiveMQQueue("queueName2");"的
    @JmsListener(destination="queueName2", containerFactory="queueListener")
    public void readActiveQueue(String message) {
        System.out.println("queue2接受到：" + message);
    }

    //更多的queue模式的消费者。。。

}
```

7、topic模式的消费者

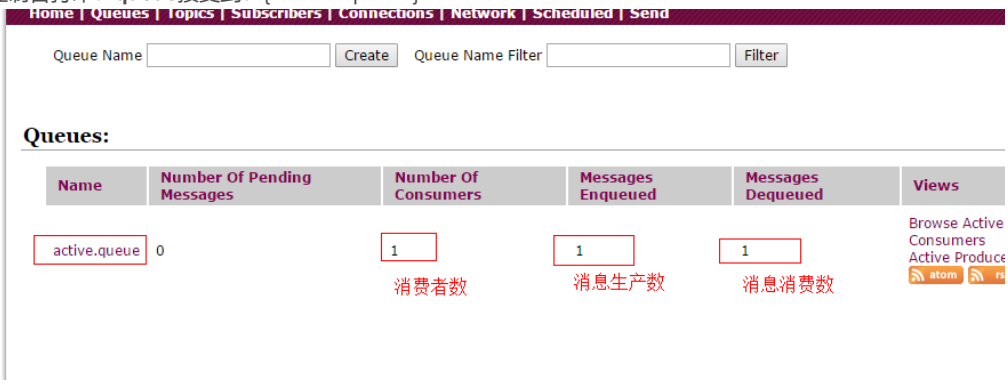
```
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class TopicConsumerListener{

    //topic模式的消费者
    @JmsListener(destination="${spring.activemq.topic-name}", containerFactory="topicListener")
    public void readActiveQueue(String message) {
```

```
        System.out.println("topic接受到：" + message);
    }

    //其他的topic模式的消费者。。。

}
```

8、测试（使用Postman发消息）

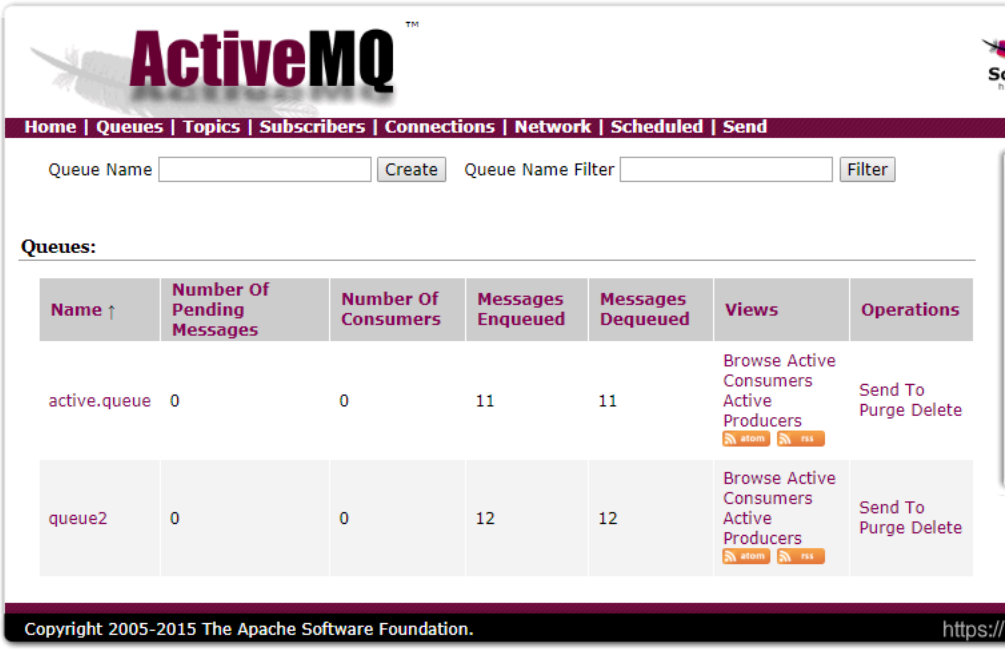
（1） POST： http://localhost:8090/**queue**/test　消息体：{"aaa" : "queue"}

控制台打印：**queue**接受到：{"aaa" : "queue"}



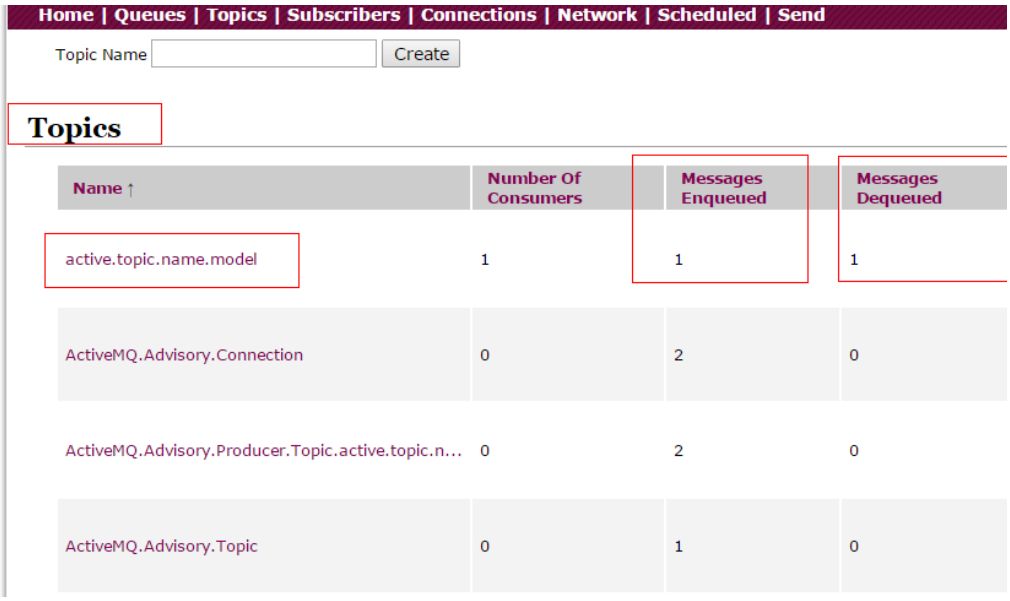
（2） POST： http://localhost:8090/**queue2**/test　消息体：{"bbb" : "queue2"}

控制台打印：**queue2**接受到：{"bbb" : "queue2"}



（3） POST： http://localhost:8090/**topic**/test　消息体：{"aaa" : "topic"}

控制台打印：**topic**接受到：{"aaa" : "topic"}



topic模式有普通订阅和持久化订阅

普通订阅：在消费者启动之前发送过来的消息，消费者启动之后不会去消费；

持久化订阅： 在消费者启动之前发送过来的消息，消费者启动之后会去消费；

四、前端VUE使用Stompjs接收ActiveMQ消息

引用原文链接：https://blog.csdn.net/yuandongy/article/details/103261001

1 创建vue项目

```
vue init webpack demo
```

2 进入demo文件夹，安装stompjs，net

```
cd demo
npm install stompjs
npm install net
```

3 在config下添加linkparam.js

```
export const MQ_SERVICE = 'ws://192.168.111.200:61614/stomp' // mq服务地址
export const MQ_USERNAME = 'admin' // mq连接用户名
export const MQ_PASSWORD = 'admin' //mq连接密码
```

4 在src/components下添加sock.vue文件

```
<template>
  <div>
    socket is loading...
  </div>
</template>
<script>
import Stomp from 'stompjs'
import { MQ_SERVICE, MQ_USERNAME, MQ_PASSWORD } from '../../config/linkparam.js'
```

```
export default {
  name: 'entry',
  data () {
    return {
      client: Stomp.client(MQ_SERVICE)
    }
  },
  created () {
    this.connect()
  },
  methods: {
    onConnected: function (frame) {
      console.log('Connected: ' + frame)
      var queue= 'active.queue'
      this.client.subscribe(queue, this.responseCallback, this.onFailed)
      var topic = 'active.topic.name.model'
      this.client.subscribe(topic, this.responseCallback, this.onFailed)
    },
    onFailed: function (frame) {
      console.log('Failed: ' + frame)
    },
    responseCallback: function (frame) {
      console.log('responseCallback msg=>' + frame.body)
      console.log('------')
    },
    connect: function () {
      var headers = {
        'login': MQ_USERNAME,
        'passcode': MQ_PASSWORD
      }
      this.client.connect(headers, this.onConnected, this.onFailed)
    }
  }
}
</script>
```

5 修改src/App.vue

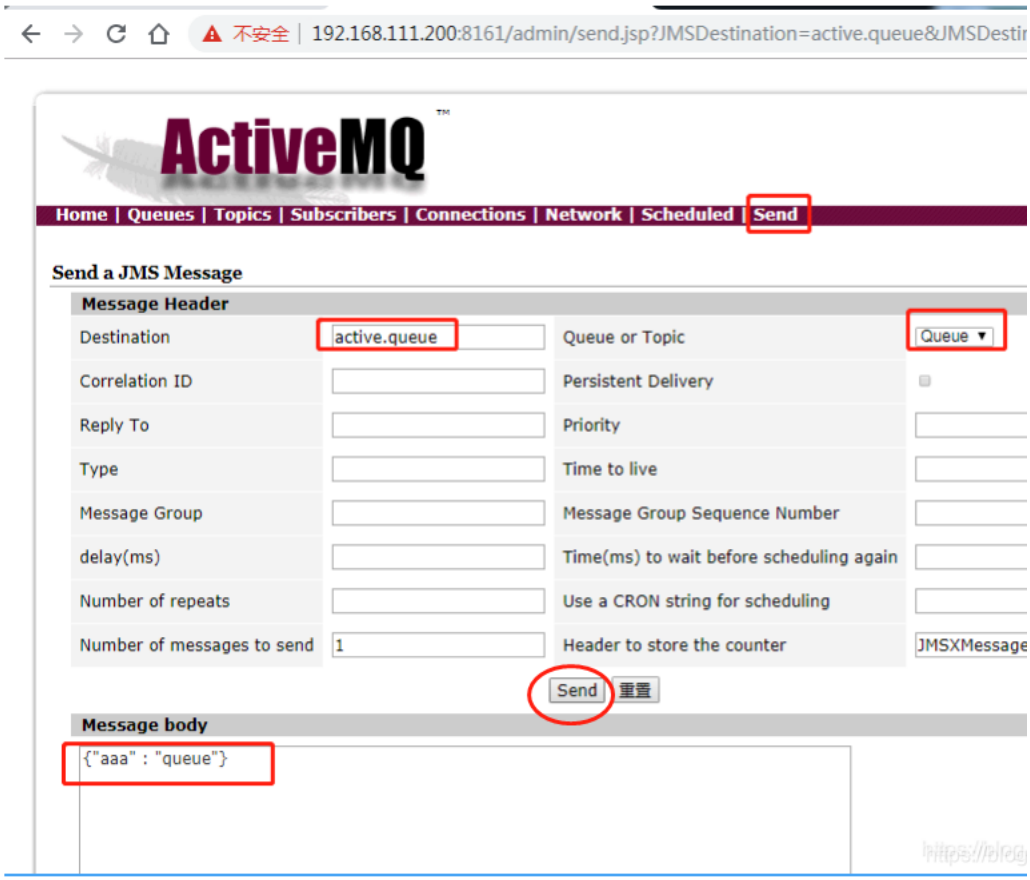
```
<template>
  <div id="app">
    <sock/>
  </div>
</template>
<script>
import HelloWorld from './components/HelloWorld'
import sock from './components/sock'
export default {
  name: 'App',
  components: {
    sock
  }
}
</script>
<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

6 运行vue

```
npm run dev
```

7 测试

进入ActiveMQ控制台，如下，



设置好后，点击“send”。

或使用Postman发消息：POST： http://localhost:8090/**queue**/test 消息体：{"aaa" : "queue"}

F12进入调试模式，看打印结果：

```
...
responseCallback msg=>{"aaa" : "queue"}
------
```

至此，全部完成