# Spring Boot 入门之消息中间件篇（五）

* 发表于 *2018-01-26* | * 分类于 后端 | * 阅读数: *1848* | * 评论数: 0

## 一、前言

在消息中间件中有 2 个重要的概念：消息代理和目的地。当消息发送者发送消息后，消息就被消息代理接管，消息代理保证消息传递到指定目的地。

我们常用的消息代理有 JMS 和 AMQP 规范。对应地，它们常见的实现分别是 ActiveMQ 和 RabbitMQ。

上篇文章[《Spring Boot 入门之缓存和 NoSQL 篇（四）》。

## 二、整合 ActiveMQ

### 2.1 添加依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>

<!-- 如果需要配置连接池，添加如下依赖 -->
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-pool</artifactId>
</dependency>
```

### 2.2 添加配置

```
# activemq 配置
spring.activemq.broker-url=tcp://192.168.2.61:61616
spring.activemq.user=admin
spring.activemq.password=admin
spring.activemq.pool.enabled=false
spring.activemq.pool.max-connections=50
# 使用发布/订阅模式时，下边配置需要设置成 true
spring.jms.pub-sub-domain=false
```

此处 spring.activemq.pool.enabled=false，表示关闭连接池。

## 2.3 编码

配置类：

```
@Configuration
public class JmsConfirguration {

  public static final String QUEUE_NAME = "activemq_queue";

  public static final String TOPIC_NAME = "activemq_topic";

  @Bean
    public Queue queue() {
        return new ActiveMQQueue(QUEUE_NAME);
    }

  @Bean
  public Topic topic() {
    return new ActiveMQTopic(TOPIC_NAME);
  }
}
```

负责创建队列和主题。

消息生产者：

```
@Component
public class JmsSender {

  @Autowired
  private Queue queue;

  @Autowired
  private Topic topic;

  @Autowired
  private JmsMessagingTemplate jmsTemplate;

  public void sendByQueue(String message) {
    this.jmsTemplate.convertAndSend(queue, message);
  }

  public void sendByTopic(String message) {
    this.jmsTemplate.convertAndSend(topic, message);
  }
}
```

消息消费者：

```
@Component
public class JmsReceiver {

  @JmsListener(destination = JmsConfirguration.QUEUE_NAME)
  public void receiveByQueue(String message) {
    System.out.println("接收队列消息:" + message);
  }

  @JmsListener(destination = JmsConfirguration.TOPIC_NAME)
  public void receiveByTopic(String message) {
    System.out.println("接收主题消息:" + message);
  }
}
```

消息消费者使用 @JmsListener 注解监听消息。

## 2.4 测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class JmsTest {

  @Autowired
  private JmsSender sender;

  @Test
  public void testSendByQueue() {
    for (int i = 1; i < 6; i++) {
      this.sender.sendByQueue("hello activemq queue " + i);
    }
  }

  @Test
  public void testSendByTopic() {
    for (int i = 1; i < 6; i++) {
      this.sender.sendByTopic("hello activemq topic " + i);
    }
  }
}
```

打印结果：

```
接收队列消息:hello activemq queue 1
接收队列消息:hello activemq queue 2
接收队列消息:hello activemq queue 3
接收队列消息:hello activemq queue 4
接收队列消息:hello activemq queue 5
```

测试发布/订阅模式时，设置 **spring.jms.pub-sub-domain=true**

```
接收主题消息:hello activemq topic 1
接收主题消息:hello activemq topic 2
接收主题消息:hello activemq topic 3
接收主题消息:hello activemq topic 4
接收主题消息:hello activemq topic 5
```

# 三、整合 RabbitMQ

## 3.1 添加依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

## 3.2 添加配置

```
spring.rabbitmq.host=192.168.2.71
spring.rabbitmq.port=5672
spring.rabbitmq.username=light
spring.rabbitmq.password=light
spring.rabbitmq.virtual-host=/test
```

## 3.3 编码

配置类：

```
@Configuration
public class AmqpConfirguration {

  //=============简单、工作队列模式===============

  public static final String SIMPLE_QUEUE = "simple_queue";

  @Bean
  public Queue queue() {
    return new Queue(SIMPLE_QUEUE, true);
  }

  //===============发布/订阅模式============

  public static final String PS_QUEUE_1 = "ps_queue_1";
  public static final String PS_QUEUE_2 = "ps_queue_2";
  public static final String FANOUT_EXCHANGE = "fanout_exchange";

  @Bean
  public Queue psQueue1() {
    return new Queue(PS_QUEUE_1, true);
  }

  @Bean
  public Queue psQueue2() {
    return new Queue(PS_QUEUE_2, true);
  }

  @Bean
  public FanoutExchange fanoutExchange() {
    return new FanoutExchange(FANOUT_EXCHANGE);
  }

  @Bean
  public Binding fanoutBinding1() {
    return BindingBuilder.bind(psQueue1()).to(fanoutExchange());
  }

  @Bean
  public Binding fanoutBinding2() {
```

```
    return BindingBuilder.bind(psQueue2()).to(fanoutExchange());
  }

  //===============路由模式============

  public static final String ROUTING_QUEUE_1 = "routing_queue_1";
  public static final String ROUTING_QUEUE_2 = "routing_queue_2";
  public static final String DIRECT_EXCHANGE = "direct_exchange";

  @Bean
  public Queue routingQueue1() {
    return new Queue(ROUTING_QUEUE_1, true);
  }

  @Bean
  public Queue routingQueue2() {
    return new Queue(ROUTING_QUEUE_2, true);
  }

  @Bean
  public DirectExchange directExchange() {
    return new DirectExchange(DIRECT_EXCHANGE);
  }

  @Bean
  public Binding directBinding1() {
    return
BindingBuilder.bind(routingQueue1()).to(directExchange()).with("user");
  }

  @Bean
  public Binding directBinding2() {
    return
BindingBuilder.bind(routingQueue2()).to(directExchange()).with("order");
  }

  //===============主题模式============

  public static final String TOPIC_QUEUE_1 = "topic_queue_1";
  public static final String TOPIC_QUEUE_2 = "topic_queue_2";
  public static final String TOPIC_EXCHANGE = "topic_exchange";
```

```
  @Bean
  public Queue topicQueue1() {
    return new Queue(TOPIC_QUEUE_1, true);
  }

  @Bean
  public Queue topicQueue2() {
    return new Queue(TOPIC_QUEUE_2, true);
  }

  @Bean
  public TopicExchange topicExchange() {
    return new TopicExchange(TOPIC_EXCHANGE);
  }

  @Bean
  public Binding topicBinding1() {
    return
BindingBuilder.bind(topicQueue1()).to(topicExchange()).with("user.add");
  }

  @Bean
  public Binding topicBinding2() {
    return
BindingBuilder.bind(topicQueue2()).to(topicExchange()).with("user.#");
  }

}
```

RabbitMQ 有多种工作模式，因此配置比较多。想了解相关内容的读者可以查看本站的[《RabbitMQ 工作模式介绍》或者自行百度相关资料。

消息生产者：

```
@Component
public class AmqpSender {

  @Autowired
  private AmqpTemplate amqpTemplate;

  /**
   * 简单模式发送
   *
   * @param message
   */
  public void simpleSend(String message) {
    this.amqpTemplate.convertAndSend(AmqpConfirguration.SIMPLE_QUEUE,
message);
  }

  /**
   * 发布/订阅模式发送
   *
   * @param message
   */
  public void psSend(String message) {
    this.amqpTemplate.convertAndSend(AmqpConfirguration.FANOUT_EXCHANGE,
"", message);
  }

  /**
   * 路由模式发送
   *
   * @param message
   */
  public void routingSend(String routingKey, String message) {
    this.amqpTemplate.convertAndSend(AmqpConfirguration.DIRECT_EXCHANGE,
routingKey, message);
  }

  /**
   * 主题模式发送
   *
   * @param routingKey
```

```
    * @param message
    */
   public void topicSend(String routingKey, String message) {
     this.amqpTemplate.convertAndSend(AmqpConfirguration.TOPIC_EXCHANGE,
routingKey, message);
   }
}
```

消息消费者：

```
@Component
public class AmqpReceiver {

  /**
   * 简单模式接收
   *
   * @param message
   */
  @RabbitListener(queues = AmqpConfirguration.SIMPLE_QUEUE)
  public void simpleReceive(String message) {
    System.out.println("接收消息:" + message);
  }

  /**
   * 发布/订阅模式接收
   *
   * @param message
   */
  @RabbitListener(queues = AmqpConfirguration.PS_QUEUE_1)
  public void psReceive1(String message) {
    System.out.println(AmqpConfirguration.PS_QUEUE_1 + "接收消息:" +
message);
  }

  @RabbitListener(queues = AmqpConfirguration.PS_QUEUE_2)
  public void psReceive2(String message) {
    System.out.println(AmqpConfirguration.PS_QUEUE_2 + "接收消息:" +
message);
  }

  /**
   * 路由模式接收
   *
   * @param message
   */
  @RabbitListener(queues = AmqpConfirguration.ROUTING_QUEUE_1)
  public void routingReceive1(String message) {
    System.out.println(AmqpConfirguration.ROUTING_QUEUE_1 + "接收消息:" +
message);
  }
```

```
  @RabbitListener(queues = AmqpConfirguration.ROUTING_QUEUE_2)
  public void routingReceive2(String message) {
    System.out.println(AmqpConfirguration.ROUTING_QUEUE_2 + "接收消息:" +
message);
  }

  /**
   * 主题模式接收
   *
   * @param message
   */
  @RabbitListener(queues = AmqpConfirguration.TOPIC_QUEUE_1)
  public void topicReceive1(String message) {
    System.out.println(AmqpConfirguration.TOPIC_QUEUE_1 + "接收消息:" +
message);
  }

  @RabbitListener(queues = AmqpConfirguration.TOPIC_QUEUE_2)
  public void topicReceive2(String message) {
    System.out.println(AmqpConfirguration.TOPIC_QUEUE_2 + "接收消息:" +
message);
  }
}
```

消息消费者使用 @RabbitListener 注解监听消息。

## 3.4 测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class AmqpTest {

  @Autowired
  private AmqpSender sender;

  @Test
  public void testSimpleSend() {
    for (int i = 1; i < 6; i++) {
      this.sender.simpleSend("test simpleSend " + i);
    }
  }

  @Test
  public void testPsSend() {
    for (int i = 1; i < 6; i++) {
      this.sender.psSend("test psSend " + i);
    }
  }

  @Test
  public void testRoutingSend() {
    for (int i = 1; i < 6; i++) {
      this.sender.routingSend("order", "test routingSend " + i);
    }
  }

  @Test
  public void testTopicSend() {
    for (int i = 1; i < 6; i++) {
      this.sender.topicSend("user.add", "test topicSend " + i);
    }
  }
}
```

测试结果略过。。。

踩坑提醒**1：ACCESS_REFUSED - Login was refused using authentication mechanism PLAIN**

解决方案：

1. 请确保用户名和密码是否正确，需要注意的时用户名和密码的值是否包含空格或制表符。
2. 如果测试账户使用的是 guest，需要修改 rabbitmq.conf 文件。在该文件中添加 “loopback_users = none” 配置。

踩坑提醒**2**：**Cannot prepare queue for listener. Either the queue doesn't exist or the broker will not allow us to use it**

解决方案：

我们可以登陆 RabbitMQ 的管理界面，在 Queue 选项中手动添加对应的队列。

# 四、源码下载

- [Spring Boot 入门之消息中间件篇测试源码]

# 五、参考资料

- 消息中间件简单介绍
- Spring Boot 官方文档
- Rabbit MQ 访问控制相关