SpringBoot整合RabbitMQ高级特性

消息可靠性投递

一、消息可靠性投递原理

在使用 RabbitMQ 的时候，作为消息发送方希望杜绝任何消息丢失或者投递失败场景。RabbitMQ 为我们提供了两种方式用来控制消息的投递可靠性模式。

confirm 确认模式

return 退回模式

rabbitmq 整个消息投递的路径为：

producer--->rabbitmq broker--->exchange--->queue--->consumer

消息从 producer 到 exchange 则会返回一个 confirmCallback 。

消息从 exchange-->queue 投递失败则会返回一个 returnCallback 。

我们将利用这两个 callback 控制消息的可靠性投递

二、创建maven项目并且引入依赖

XML | 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.6-SNAPSHOT</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.jzc.rabbitmq</groupId>
  <artifactId>springboot-rabbitmq-producer</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>springboot-rabbitmq-producer</name>
  <description>springboot-rabbitmq-producer</description>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.amqp</groupId>
      <artifactId>spring-rabbit-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
```

```
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

三、配置producer的yml文件

YAML | 复制代码

```yaml
# 配置rabbitmq服务器相关信息
spring:
  rabbitmq:
    host: 172.26.41.194
    port: 5672
    username: admin
    password: admin
    virtual-host: /hzero
```

四、创建配置类

Java | 复制代码

```java
/**
* 高级特性（消息可靠性投递）-配置类
*
* @author zhichao.jiang01@hand-china.com 2022/10/25 10:04
*/
@Configuration
    public class RabbitMQSeniorConfig {

        public static final String EXCHANGE_SENIOR_NAME = "boot_senior_exchange";
        public static final String QUEUE_SENIOR_NAME = "boot_senior_queue";

        /**
* 高级特性测试-交换机
*
* @author zhichao.jiang01@hand-china.com 2022/10/25 15:31
*/
        @Bean("bootSeniorExchange")
        public Exchange bootSeniorExchange() {
            return ExchangeBuilder.topicExchange(EXCHANGE_SENIOR_NAME).durable(true).build();
        }

        /**
* 高级特性测试-队列
*
* @author zhichao.jiang01@hand-china.com 2022/10/25 15:31
*/
        @Bean("bootSeniorQueue")
        public Queue bootSeniorQueue(){
            return QueueBuilder.durable(QUEUE_SENIOR_NAME).build();
        }

        /**
*队列和交换机绑定关系
* 1. 知道哪个队列
* 2. 知道哪个交换机
* 3. routing key
*
* @author zhichao.jiang01@hand-china.com 2022/10/25 15:32
*/
        @Bean
        public Binding bindSeniorQueueExchange(@Qualifier("bootSeniorQueue") Queue queue,@Qualifier("bootSeniorExchange") Exchange exchange){
```

```
            return BindingBuilder.bind(queue).to(exchange).with("senior.#").noargs();
        }
    }
```

五、开启确认模式和返回模式

YAML | 复制代码

```yaml
# 配置rabbitmq服务器相关信息
spring:
  rabbitmq:
    host: 172.26.41.194
    port: 5672
    username: admin
    password: admin
    virtual-host: /hzero
    #NONE值是禁用发布确认模式，是默认值
    #CORRELATED值是发布消息成功到交换器后会触发回调方法，如1示例
    #SIMPLE值经测试有两种效果，其一效果和CORRELATED值一样会触发回调方法，其二在发布消息成功后使用rabbitTemplate调用waitForConfirms或waitForConfirmsOrDie方法等待broker节点返回发送结果，根据返回结果来判定下一步的逻辑，要注意的点是waitForConfirmsOrDie方法如果返回false则会关闭channel，则接下来无法发送消息到broker;
    publisher-confirm-type: correlated
    # 开启返回模式
    publisher-returns: true
    template:
      mandatory: true


```

六、创建测试类发送消息

Java | 复制代码

```java

/**
 * 高级特性-发送消息（消息可靠性投递）
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/25 15:35
 */
@SpringBootTest
@RunWith(SpringRunner.class)
class RabbitmqProducerSeniorTests {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    /**
     * 确认模式：
     * 步骤：
     * 1. 确认模式开启：配置yml文件中publisher-confirm-type: correlated
     * 2. 在rabbitTemplate定义回调函数
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/25 15:36
     */
    @Test
    void testConfirm() {
        // 2.定义回调
        rabbitTemplate.setConfirmCallback(new RabbitTemplate.ConfirmCallback() {
            /**
             *
             * @param correlationData 相关配置信息
             * @param b exchange交换机是否接收到消息 false/true
             * @param s 失败原因
             */
            @Override
            public void confirm(CorrelationData correlationData, boolean b, String s) {
                System.out.println("confirm模式成功执行");
                if (b) {
                    System.out.println("接收消息成功" + s);
                } else {
                    System.out.println("接收消息失败" + s);
                }
            }
        });
        // 3.发送消息

```

```
        rabbitTemplate.convertAndSend(RabbitMQSeniorConfig.EXCHANGE_SENIOR_NAME,"senior.sshh","confirm。。。");
    }

    /**
     * 回退模式:当消息发送给Exchange后，Exchange路由到Queue失败是才会执行ReturnCallBack步骤:
     * 1. 开启回退模式: publisher-returns="true"
     * 2. 设置Return CallBack
     * 3. 设置Exchange处理消息的模式:
     *        1. 如果消息没有路由到Queue，则丢弃消息（默认）
     *        2. 如果消息没有路由到Queue，返回给消息发送方Return callBack
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/25 15:36
     */
    @Test
    void testReturn() {
        // 2.定义回调
        rabbitTemplate.setReturnsCallback(new RabbitTemplate.ReturnsCallback() {
            @Override
            public void returnedMessage(ReturnedMessage returnedMessage) {
                System.out.println("return 执行了。。。。");
                System.out.println(returnedMessage);
            }
        });
        // 3.发送消息
        rabbitTemplate.convertAndSend(RabbitMQSeniorConfig.EXCHANGE_SENIOR_NAME,"senior.fa","confirm。。。");
    }

}
```

消息的可靠投递小结

- 设置ConnectionFactory的publisher-confirms="true" 开启 确认模式。

- 使用rabbitTemplate.setConfirmCallback设置回调函数。当消息发送到exchange后回调confirm方法。在方法中判断ack，如果为true，则发送成功，如果为false，则发送失败，需要处理。

- 设置ConnectionFactory的publisher-returns="true" 开启 退回模式。

- 使用rabbitTemplate.setReturnCallback设置退回函数，当消息从exchange路由到queue失败后，如果设置了rabbitTemplate.setMandatory(true)参数，则会将消息退回给producer。并执行回调函数returnedMessage。

- 在RabbitMQ中也提供了事务机制，但是性能较差（差25倍）。

  使用channel下列方法，完成事务控制：

  txSelect(), 用于将当前channel设置成transaction模式

  txCommit()，用于提交事务

  txRollback(),用于回滚事务

ConsumerAck

一、ConsumerAck原理

ack指Acknowledge，确认。 表示消费端收到消息后的确认方式。

有三种确认方式：

•自动确认：acknowledge="none"

•手动确认：acknowledge="manual"

•根据异常情况确认：acknowledge="auto"，（这种方式使用麻烦）

其中自动确认是指，当消息一旦被Consumer接收到，则自动确认收到，并将相应 message 从 RabbitMQ 的消息缓存中移除。但是在实际业务处理中，很可能消息接收到，业务处理出现异常，那么该消息就会丢失。如果设置了手动确认方式，则需要在业务处理成功后，调用channel.basicAck()，手动签收，如果出现异常，则调用channel.basicNack()方法，让其自动重新发送消息。

二、配置pom文件和创建Maven工程

参考：

此处为语雀内容卡片，点击链接查看：https://www.yuque.com/knightjzc/vk0a70/bad4r0?view=doc_embed

三、配置consumer的yml文件

YAML | 复制代码

```yaml
# 配置rabbitmq服务器相关信息
spring:
  rabbitmq:
    host: 172.26.41.194
    port: 5672
    username: admin
    password: admin
    virtual-host: /hzero
    # 开启手动确认模式
    listener:
      direct:
        acknowledge-mode: manual
```

四、创建监听类并声明@RabbitListener注解

Java | 复制代码

```java

/**
 * springboot-rabbitmq-监听类
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/26 13:35
 */
@Component
public class RabbitMQListener {

    /**
     * Consumer ACK机制:
     * 1. 设置手动签收。listener:
     *                   direct:
     *                     acknowledge-mode: manual(yml文件设置)
     * 2. @RabbitListener添加ackMode模式为MANUAL
     * 3. 如果消息成功处理，则调用channel的basicAck()签收
     * 4. 如果消息处理失败，则调用channel的basicNack()拒绝签收，broker重新发送给consumer
     *
     */
    @RabbitListener(queues = "boot_senior_queue",ackMode = "MANUAL")
    public void ListenerSeniorQueue(Message message, Channel channel) throws IOException {
        long deliveryTag = message.getMessageProperties().getDeliveryTag();
        try {
            //1.接收转换消息
            System.out.println(new String(message.getBody()));

            //2.处理业务逻辑
            System.out.println("处理业务逻辑。。。");
            //假如出错
            //int a=3/0;
            //3.手动签收消息
            channel.basicAck(deliveryTag, true);
        }catch (Exception e){
            //4.拒绝签收
            /*
            第三个参数requeue：重回队列。如果设置为true，则消息重新回到queue，broker会重新发送该消息给消费端
             */
            channel.basicNack(deliveryTag,true,true);
        }
    }

```

```
42  }
43
```

ConsumerAck小结

- 在rabbit:listener-container标签中设置acknowledge属性，设置ack方式 none：自动确认，manual：手动确认

- 如果在消费端没有出现异常，则调用channel.basicAck(deliveryTag,false);方法确认签收消息

- 如果出现异常，则在catch中调用 basicNack或 basicReject，拒绝消息，让MQ重新发送消息。

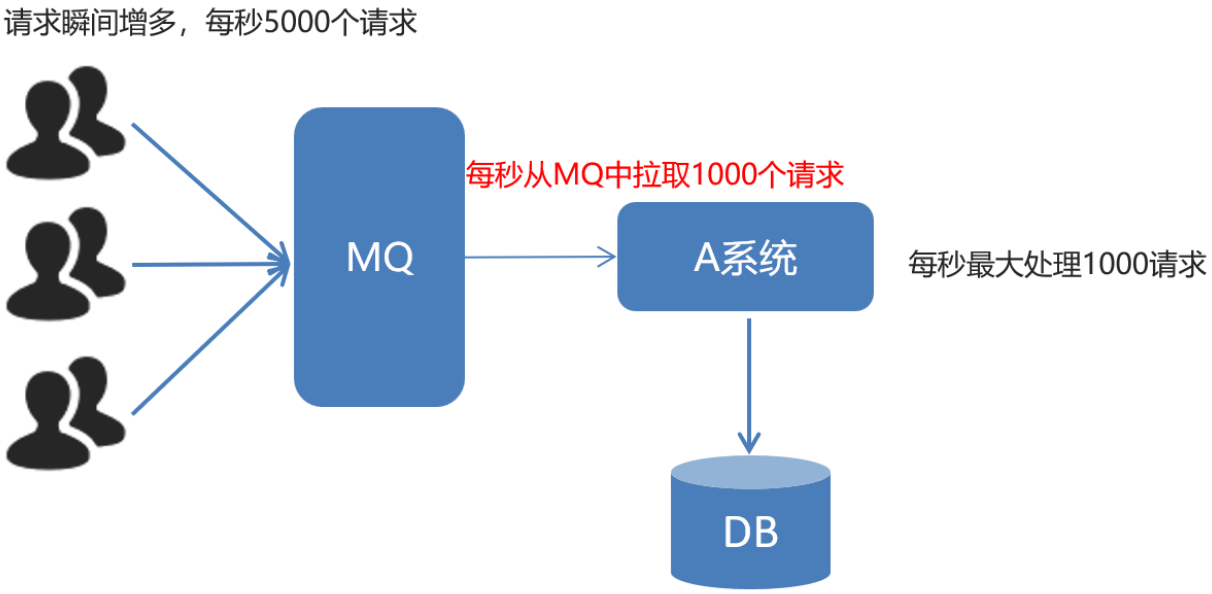
消息可靠性总结

1. 持久化
- exchange要持久化
- queue要持久化
- message要持久化
2. 生产方确认Confirm
3. 消费方确认Ack
4. Broker高可用

消费端限流

一、消费端限流原理

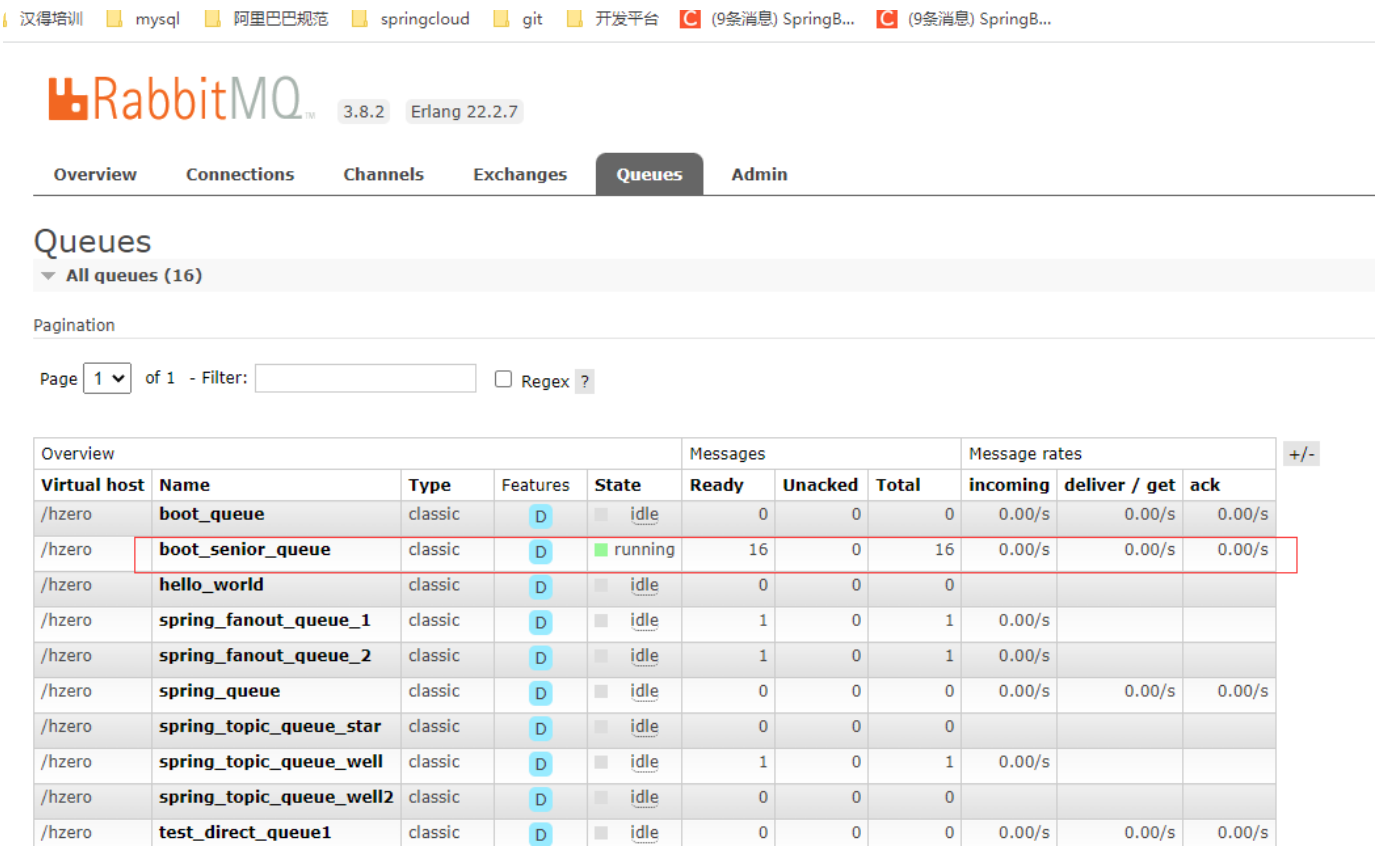
- 在yml中配置 prefetch属性设置消费端一次拉取多少消息
- 消费端的确认模式一定为手动确认。acknowledge="manual"



二、producer发送10条信息

Java | 复制代码

```
@Test
    void testReturn() {
        // 2.定义回调
        rabbitTemplate.setReturnsCallback(new RabbitTemplate.ReturnsCallback() {
            @Override
            public void returnedMessage(ReturnedMessage returnedMessage) {
                System.out.println("return 执行了。。。。");
                System.out.println(returnedMessage);
            }
        });
        // 3.发送消息
        for(int i = 0; i < 10; i++) {
            rabbitTemplate.convertAndSend(RabbitMQSeniorConfig.EXCHANGE_SENIOR_NAME,"senior.fa","confirm。。。");
        }
    }
```



三、配置consumer的yml，限制接收消息条数

前提：必须开启手动确认

prefetch设置限流条数

Java 复制代码

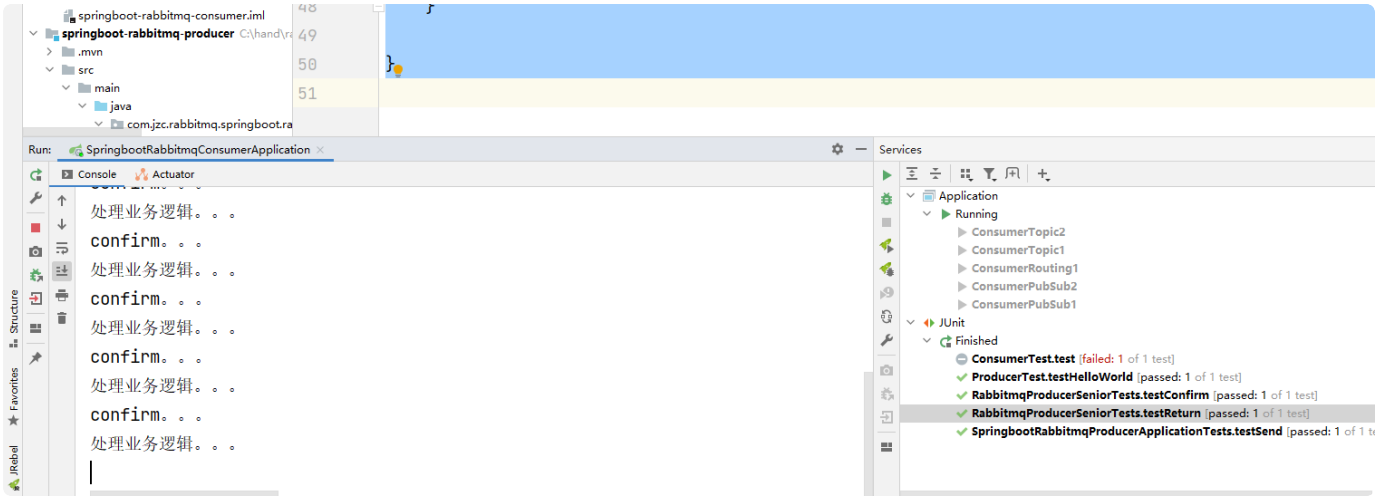
```yaml
# 配置rabbitmq服务器相关信息
spring:
  rabbitmq:
    host: 172.26.41.194
    port: 5672
    username: admin
    password: admin
    virtual-host: /hzero
    # 开启手动确认模式
    listener:
      direct:
        acknowledge-mode: manual
        prefetch: 1
```

四、创建对应consumer监听器

Java | 复制代码

```java
/**
 * springboot-rabbitmq-监听类
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/25 13:35
 */
@Component
public class QosListener {
    /**
     * Consumer ACK机制:
     * 1. 设置手动签收。listener:
     *                   direct:
     *                     acknowledge-mode: manual(yml文件设置)
     * 2. @RabbitListener添加ackMode模式为MANUAL
     * 3. 如果消息成功处理，则调用channel的basicAck()签收
     * 4. 如果消息处理失败，则调用channel的basicNack()拒绝签收，broker重新发送给consumer
     *
     */
    @RabbitListener(queues = "boot_senior_queue",ackMode = "MANUAL")
    public void listenerSeniorQueue(Message message, Channel channel) throws IOException, InterruptedException {
        Thread.sleep(1000);
        long deliveryTag = message.getMessageProperties().getDeliveryTag();
        try {
            //1.接收转换消息
            System.out.println(new String(message.getBody()));

            //2.处理业务逻辑
            System.out.println("处理业务逻辑。。。");
            //假如出错
            //int a=3/0;
            //3.手动签收消息
            channel.basicAck(deliveryTag, true);
        }catch (Exception e){
            //4.拒绝签收
            /*
            第三个参数requeue：重回队列。如果设置为true，则消息重新回到queue，broker会重新发送该消息给消费端
             */
            channel.basicNack(deliveryTag,true,true);
        }
    }

}
```

```
42
```



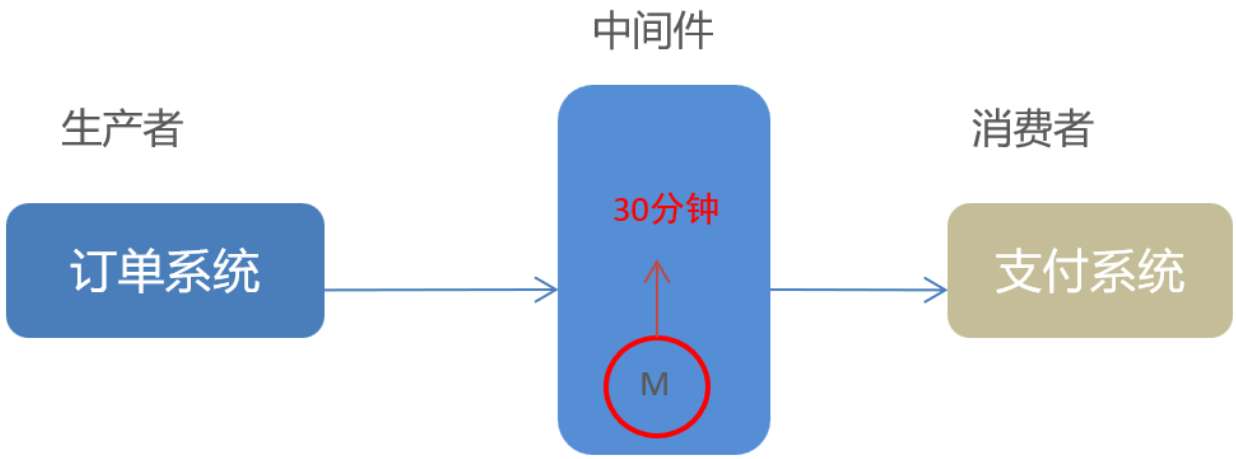
TTL消息过期

一、TTL消息过期原理

TTL 全称 Time To Live（存活时间/过期时间）。

当消息到达存活时间后，还没有被消费，会被自动清除。

RabbitMQ可以对消息设置过期时间，也可以对整个队列（Queue）设置过期时间。



二、producer创建TTL配置类

设置队列过期时间：withArgument("x-message-ttl",10000) //10秒

Java | 复制代码

```java
/**
 * 高级特性（ttl消息过期）-配置类
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/26 19:04
 */
@Configuration
public class TtlSeniorConfig {

    public static final String EXCHANGE_TTL_NAME = "boot_ttl_exchange";
    public static final String QUEUE_TTL_NAME = "boot_ttl_queue";

    /**
     * 高级特性测试-交换机
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/25 15:31
     */
    @Bean("bootTtlExchange")
    public Exchange bootTtlExchange() {
        return ExchangeBuilder.topicExchange(EXCHANGE_TTL_NAME).durable(true).build();
    }

    /**
     * 高级特性测试-队列
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/26 19:31
     */
    @Bean("bootTtlQueue")
    public Queue bootTtlQueue(){
        //x-message-ttl 指的是过期时间
        return QueueBuilder.durable(QUEUE_TTL_NAME).withArgument("x-message-ttl",10000).build();
    }

    /**
     *队列和交换机绑定关系
     * 1. 知道哪个队列
     * 2. 知道哪个交换机
     * 3. routing key
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/26 19:32
     */
    @Bean
    public Binding bindTtlQueueExchange(@Qualifier("bootTtlQueue") Queue queue,@Qualifier("bootTtlExchange") Exchange exchange){
```

```
43          return BindingBuilder.bind(queue).to(exchange).with("ttl.#").noarg
    s();
44      }
45  }
46
```

三、producer创建发送消息测试类

Java | 复制代码

```java

/**
 * 高级特性（ttl消息过期）-发送消息测试类
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/26 19:35
 */
@SpringBootTest
@RunWith(SpringRunner.class)
class RabbitmqProducerTtlTests {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    /**
     * TTL:过期时间
     * 队列统一过期
     *
     * 2.消息单独过期
     * 如果设置了消息的过期时间，也设置了队列的过期时间，它以时间短的为准。队列过期后，会将队列所有消息全部移除。
     * 消息过期后,只有消息在队列顶端,才会判断其是否过期(移除掉)
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/26 19:36
     */
    @Test
    void testTtl() {
        MessagePostProcessor message=new MessagePostProcessor() {
            @Override
            public Message postProcessMessage(Message message) throws AmqpException {
                //设置message消息
                //过期时间
                message.getMessageProperties().setExpiration("5000");
                //返回消息
                return message;
            }
        };
        // 3.发送消息
        for(int i = 0; i < 10; i++) {
            if (i == 5) {
                // 单独过期的消息
                rabbitTemplate.convertAndSend(TtlSeniorConfig.EXCHANGE_TTL_NAME, "ttl.fa", "ttl测试", message);
            }else {
                // 不过期消息
```

```
43                  rabbitTemplate.convertAndSend(TtlSeniorConfig.EXCHANGE_TTL
    _NAME, "ttl.fa", "ttl测试");
44              }
45          }
46      }
47
48  }
49
```

TTL小结

设置队列过期时间使用参数：x-message-ttl，单位：ms(毫秒)，会对整个队列消息统一过期。

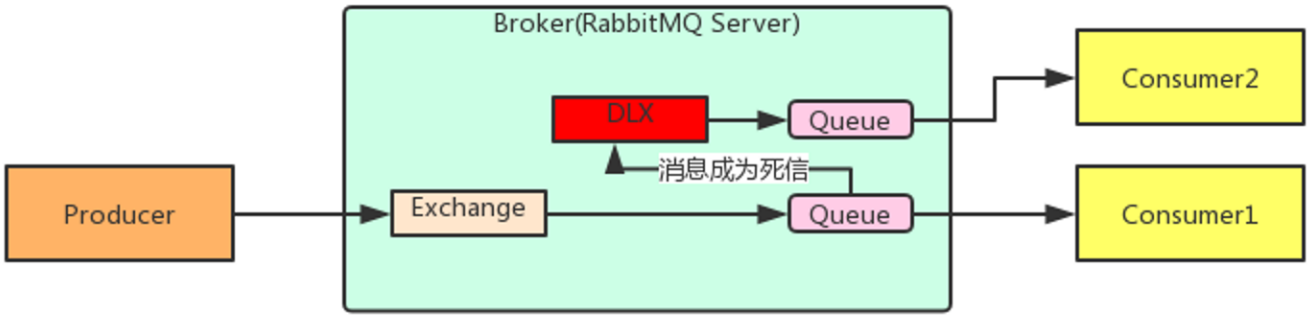
设置消息过期时间使用参数：expiration。单位：ms(毫秒)，当该消息在队列头部时（消费时），会单独判断这一消息是否过期。

如果两者都进行了设置，以时间短的为准。

死信队列

一、死信队列原理

死信队列，英文缩写：DLX 。Dead Letter Exchange（死信交换机），当消息成为Dead message后，可以被重新发送到另一个交换机，这个交换机就是DLX。



消息成为死信的三种情况：

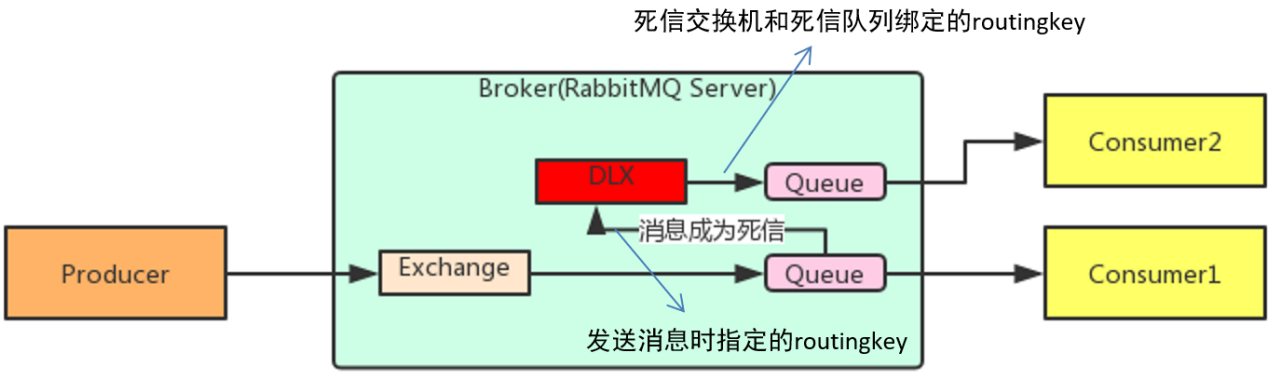
1. 队列消息长度到达限制；

2. 消费者拒接消费消息，basicNack/basicReject,并且不把消息重新放入原目标队列,requeue=false；

3. 原队列存在消息过期设置，消息到达超时时间未被消费；

队列绑定死信交换机：

给队列设置参数： x-dead-letter-exchange 和 x-dead-letter-routing-key



二、producer创建死信队列配置类

1. 声明正常的队列(boot_dlx_queue)和交换机(boot_dlx_exchange)
2. 声明死信队列(dlx_queue)和死信交换机(dlx_exchange)
3. 正常队列绑定死信交换机

   设置两个参数:

   x-dead-letter-exchange:死信交换机名称

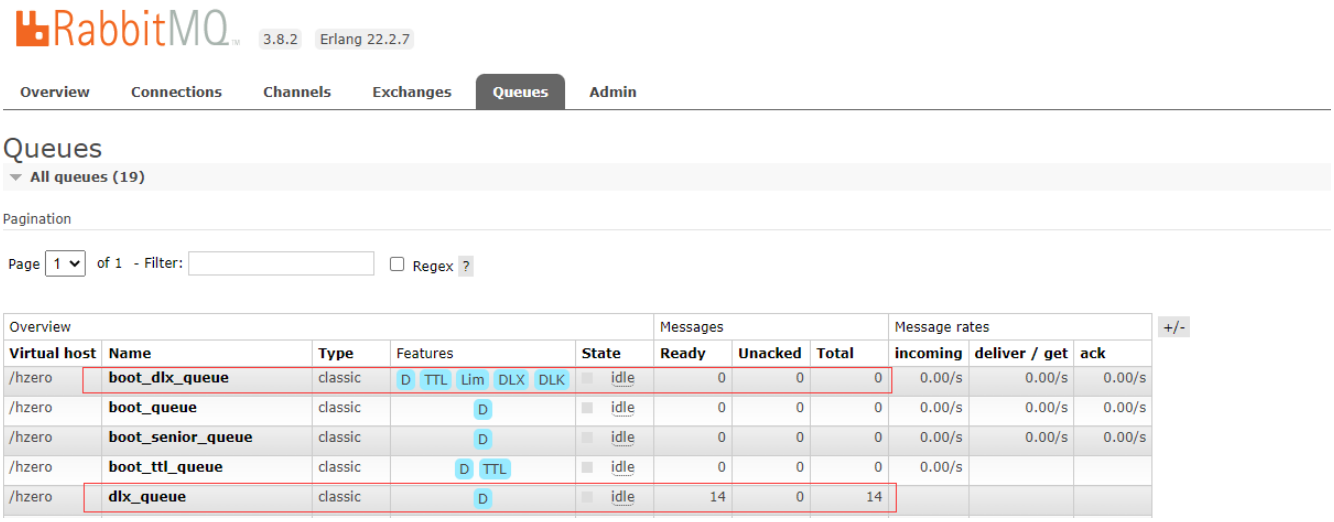
   x-dead-letter-routing-key:发送给死信交换机的routingkey

Java | 复制代码

```java

/**
 * 高级特性（死信队列）-配置类
 *
 * 1. 声明正常的队列(boot_dlx_queue)和交换机(boot_dlx_exchange)
 * 2. 声明死信队列(dlx_queue)和死信交换机(dlx_exchange)
 * 3. 正常队列绑定死信交换机
 *    设置两个参数:
 *        x-dead-letter-exchange:死信交换机名称
 *        x-dead-letter-routing-key:发送给死信交换机的routingkey
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/27 14:04
 */
@Configuration
public class DlxSeniorConfig {

    public static final String EXCHANGE_F_DLX_NAME = "boot_dlx_exchange";
    public static final String EXCHANGE_DLX_NAME = "dlx_exchange";
    public static final String QUEUE_F_DLX_NAME = "boot_dlx_queue";
    public static final String QUEUE_DLX_NAME = "dlx_queue";

    /**
     * 正常交换机
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/27 15:31
     */
    @Bean("bootDlxExchange")
    public Exchange bootDlxExchange() {
        return ExchangeBuilder.topicExchange(EXCHANGE_F_DLX_NAME).durable(true).build();
    }

    /**
     * 死信交换机
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/27 15:31
     */
    @Bean("dlxExchange")
    public Exchange dlxExchange() {
        return ExchangeBuilder.topicExchange(EXCHANGE_DLX_NAME).durable(true).build();
    }

    /**
     * 正常队列绑定死信交换机
```

```
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/27 15:31
     */
    @Bean("bootDlxQueue")
    public Queue bootDlxQueue(){
        Map<String,Object> dlxMap = new HashMap<>(16);
        //x-message-ttl 指的是过期时间
        dlxMap.put("x-message-ttl",10000);
        //x-max-length  指的是队列最大数
        dlxMap.put("x-max-length",10);
        //x-dead-letter-exchange 指的是死信交换机
        dlxMap.put("x-dead-letter-exchange",DlxSeniorConfig.EXCHANGE_DLX_NAME);
        //x-dead-letter-routing-key 指的是死信routingKey
        dlxMap.put("x-dead-letter-routing-key","dlx.hehe");
        return QueueBuilder.durable(QUEUE_F_DLX_NAME).withArguments(dlxMap).build();
    }

    /**
     * 死信队列
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/27 15:31
     */
    @Bean("dlxQueue")
    public Queue dlxQueue(){
        return QueueBuilder.durable(QUEUE_DLX_NAME).build();
    }

    /**
     *正常队列和交换机绑定关系
     * 1. 知道哪个队列
     * 2. 知道哪个交换机
     * 3. routing key
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/27 15:32
     */
    @Bean
    public Binding bindBootDlxQueueExchange(@Qualifier("bootDlxQueue") Queue queue,@Qualifier("bootDlxExchange") Exchange exchange){
        return BindingBuilder.bind(queue).to(exchange).with("boot.dlx.#").noargs();
    }


    /**
     *死信队列和死信交换机绑定关系
     * 1. 知道哪个队列
```

```
88          * 2. 知道哪个交换机
89          * 3. routing key
90          *
91          * @author zhichao.jiang01@hand-china.com 2022/10/27 15:32
92          */
93         @Bean
94         public Binding bindDlxQueueExchange(@Qualifier("dlxQueue") Queue queue
    ,@Qualifier("dlxExchange") Exchange exchange){
95             return BindingBuilder.bind(queue).to(exchange).with("dlx.#").noarg
    s();
96         }
97     }
98
```



三、producer创建发送消息测试类

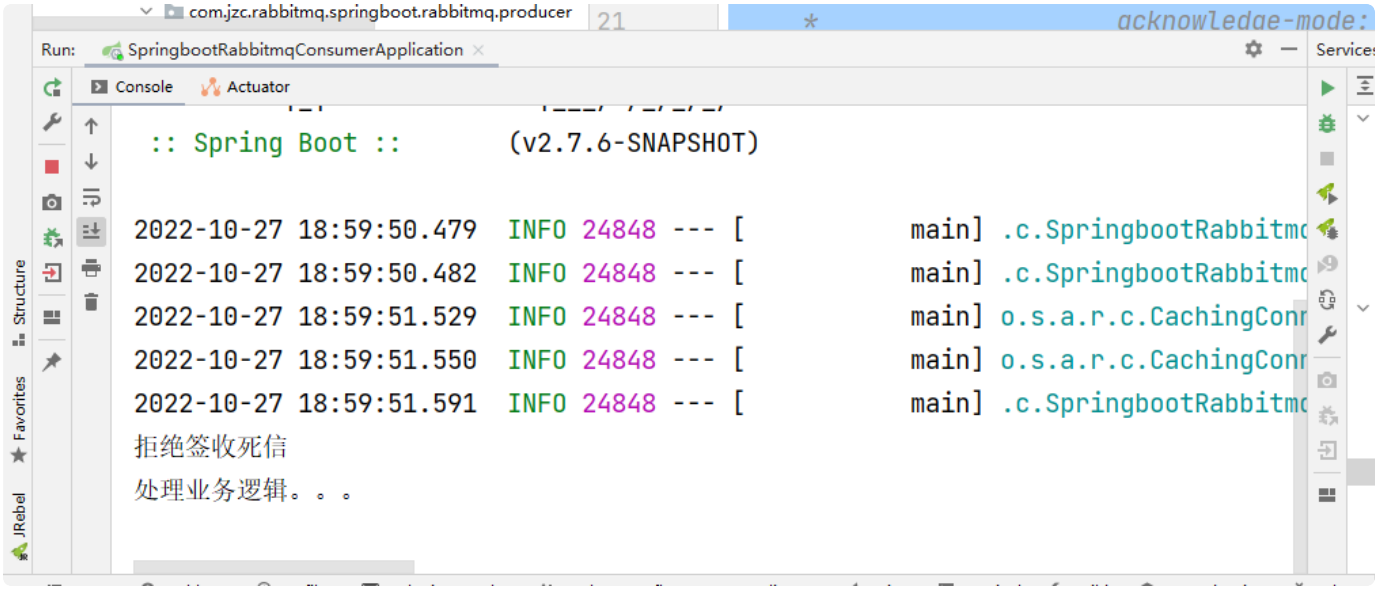
```java
/**
 * 高级特性（dxl死信队列）-发送消息测试类
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/27 16:35
 */
@SpringBootTest
@RunWith(SpringRunner.class)
class RabbitmqProducerDlxTests {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    /**
     * 发送测试死信消息
     * 1.过期时间
     * 2.队列长度限制
     * 3.拒收消息
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/27 16:36
     */
    @Test
    void testDlx() {
        // 1.过期时间
//        rabbitTemplate.convertAndSend(DlxSeniorConfig.EXCHANGE_F_DLX_NAME, "boot.dlx.aaa", "过期时间死信");
        // 2.队列长度限制
//        for (int i = 0; i < 11; i++) {
//            rabbitTemplate.convertAndSend(DlxSeniorConfig.EXCHANGE_F_DLX_NAME, "boot.dlx.fa", "dlx测试");
//        }
        //3. 拒绝签收
        rabbitTemplate.convertAndSend(DlxSeniorConfig.EXCHANGE_F_DLX_NAME, "boot.dlx.ccc", "拒绝签收死信");
    }


}
```

四、consumer创建消息监听类

- 拒绝签收消息并不发送原队列->发送至死信交换机

Java | 复制代码

```java
/**
 * springboot-rabbitmq-死信队列-监听类
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/25 13:35
 */
@Component
public class DlxListener {
    /**
     * Consumer ACK机制:
     * 1. 设置手动签收。listener:
     *                   direct:
     *                     acknowledge-mode: manual(yml文件设置)
     * 2. @RabbitListener添加ackMode模式为MANUAL
     * 3. 如果消息成功处理，则调用channel的basicAck()签收
     * 4. 如果消息处理失败，则调用channel的basicNack()拒绝签收，broker重新发送给consumer
     *
     */
    @RabbitListener(queues = "boot_dlx_queue",ackMode = "MANUAL")
    public void listenerSeniorQueue(Message message, Channel channel) throws IOException, InterruptedException {
        Thread.sleep(1000);
        long deliveryTag = message.getMessageProperties().getDeliveryTag();
        try {
            //1.接收转换消息
            System.out.println(new String(message.getBody()));

            //2.处理业务逻辑
            System.out.println("处理业务逻辑。。。");
            //假如出错
            int a=3/0;
            //3.手动签收消息
            channel.basicAck(deliveryTag, true);
        }catch (Exception e){
            //4.拒绝签收
            /*
            第三个参数requeue：重回队列。如果设置为false，则消息回到死信交换机
             */
            channel.basicNack(deliveryTag,true,false);
        }
    }

}
```



死信队列小结

1. 死信交换机和死信队列和普通的没有区别
2. 当消息成为死信后，如果该队列绑定了死信交换机，则消息会被死信交换机重新路由到死信队列
3. 消息成为死信的三种情况：

- 队列消息长度到达限制；
- 消费者拒接消费消息，并且不重回队列；
- 原队列存在消息过期设置，消息到达超时时间未被消费；

延迟队列

一、延迟队列原理

延迟队列，即消息进入队列后不会立即被消费，只有到达指定时间后，才会被消费。

需求：

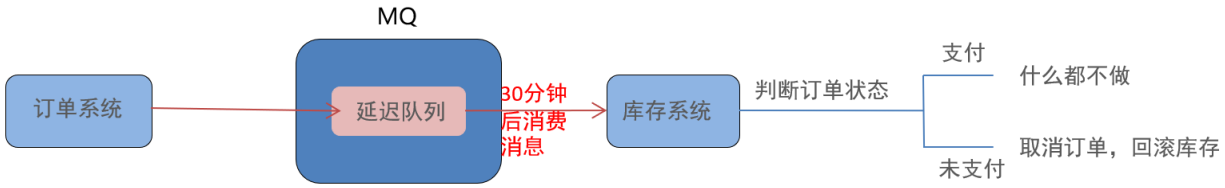
1. 下单后，30分钟未支付，取消订单，回滚库存。

2. 新用户注册成功7天后，发送短信问候。

实现方式：

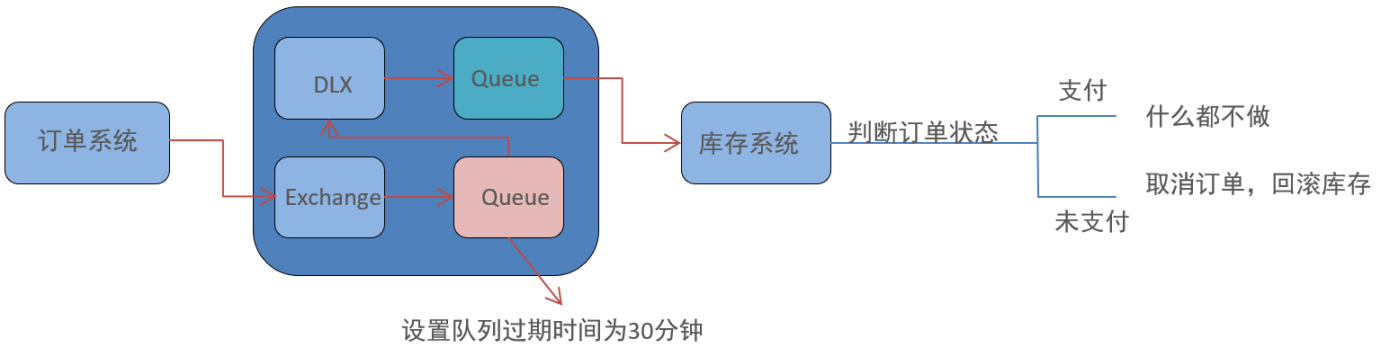
1. 定时器 （不够优雅）

2. 延迟队列（ttl+dlx）



很可惜，在RabbitMQ中并未提供延迟队列功能。

但是可以使用：TTL+死信队列 组合实现延迟队列的效果。



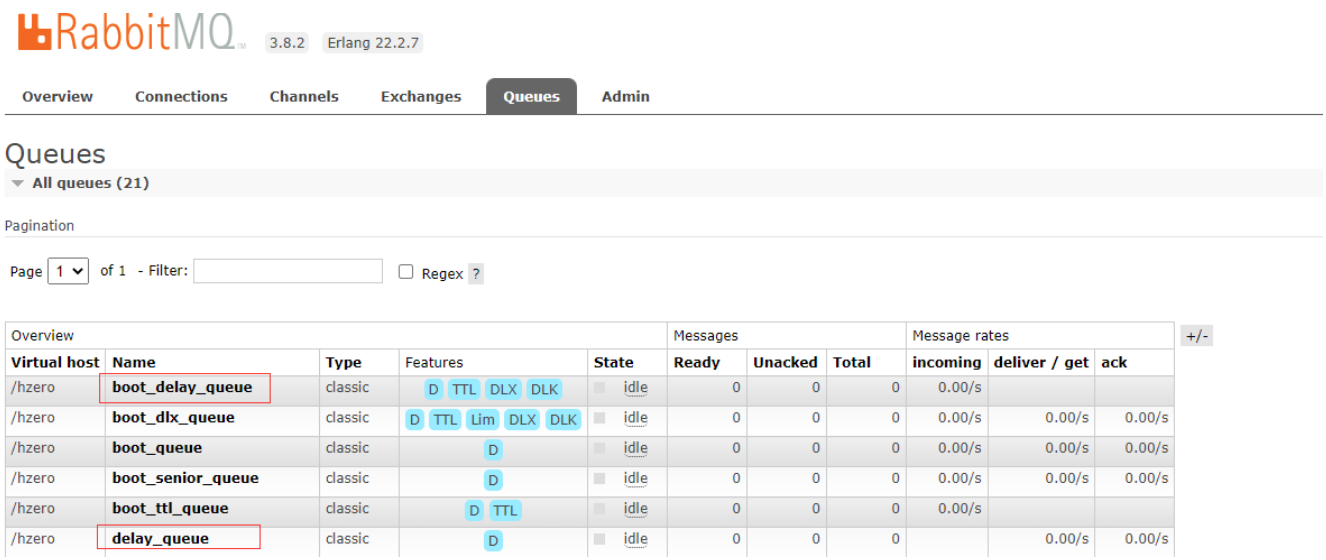
二、producer创建延迟队列配置类

Java | 复制代码

```java
/**
 * 高级特性（延迟队列）-配置类
 * ttl+dlx
 *
 * 1. 声明正常的队列(boot_delay_queue)和交换机(boot_delay_exchange)
 * 2. 声明死信队列(delay_queue)和死信交换机(delay_exchange)
 * 3. 正常队列绑定死信交换机
 *    设置两个参数:
 *        x-dead-letter-exchange:死信交换机名称
 *        x-dead-letter-routing-key:发送给死信交换机的routingkey
 *        x-message-ttl:设置延迟时间
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/28 10:04
 */
@Configuration
public class DelaySeniorConfig {

    public static final String EXCHANGE_F_DELAY_NAME = "boot_delay_exchange";
    public static final String EXCHANGE_DELAY_NAME = "delay_exchange";
    public static final String QUEUE_F_DELAY_NAME = "boot_delay_queue";
    public static final String QUEUE_DELAY_NAME = "delay_queue";

    /**
     * 正常交换机
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/28 10:31
     */
    @Bean("bootDelayExchange")
    public Exchange bootDelayExchange() {
        return ExchangeBuilder.topicExchange(EXCHANGE_F_DELAY_NAME).durable(true).build();
    }

    /**
     * 死信交换机
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/28 10:31
     */
    @Bean("delayExchange")
    public Exchange delayExchange() {
        return ExchangeBuilder.topicExchange(EXCHANGE_DELAY_NAME).durable(true).build();
    }

```

```
    /**
     * 正常队列绑定死信交换机
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/28 10:31
     */
    @Bean("bootDelayQueue")
    public Queue bootDelayQueue(){
        Map<String,Object> dlxMap = new HashMap<>(16);
        //x-message-ttl 指的是过期时间(作用为延迟时间)
        dlxMap.put("x-message-ttl",10000);
        //x-dead-letter-exchange 指的是死信交换机
        dlxMap.put("x-dead-letter-exchange", DelaySeniorConfig.EXCHANGE_DELAY_NAME);
        //x-dead-letter-routing-key 指的是死信routingKey
        dlxMap.put("x-dead-letter-routing-key","delay.hehe");
        return QueueBuilder.durable(QUEUE_F_DELAY_NAME).withArguments(dlxMap).build();
    }

    /**
     * 死信队列
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/28 10:31
     */
    @Bean("delayQueue")
    public Queue delayQueue(){
        return QueueBuilder.durable(QUEUE_DELAY_NAME).build();
    }

    /**
     *正常队列和交换机绑定关系
     * 1. 知道哪个队列
     * 2. 知道哪个交换机
     * 3. routing key
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/28 10:32
     */
    @Bean
    public Binding bindBootDelayQueueExchange(@Qualifier("bootDelayQueue") Queue queue,@Qualifier("bootDelayExchange") Exchange exchange){
        return BindingBuilder.bind(queue).to(exchange).with("boot.delay.#").noargs();
    }



    /**
     *死信队列和死信交换机绑定关系
     * 1. 知道哪个队列
```

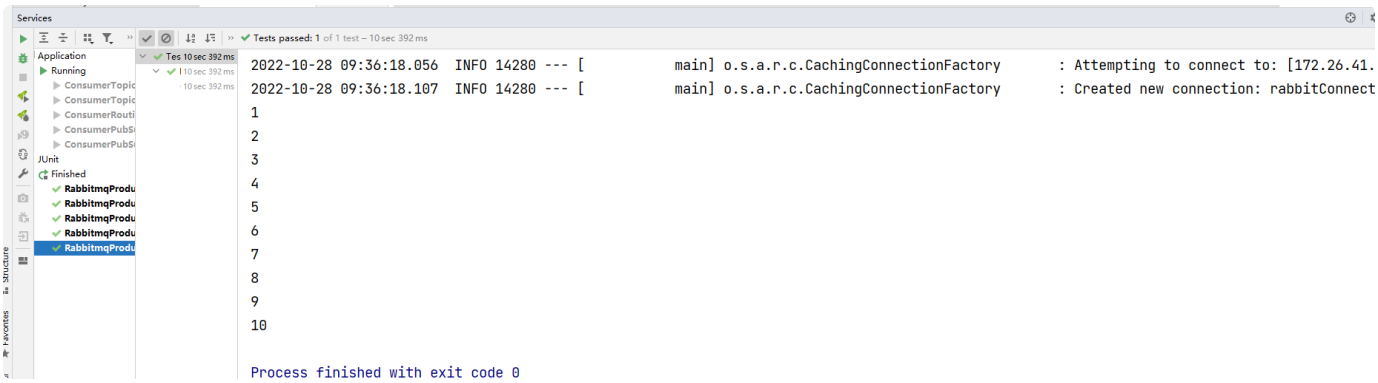
```
     * 2. 知道哪个交换机
     * 3. routing key
     *
     * @author zhichao.jiang01@hand-china.com 2022/10/28 10:32
     */
    @Bean
    public Binding bindDelayQueueExchange(@Qualifier("delayQueue") Queue queue,@Qualifier("delayExchange") Exchange exchange){
        return BindingBuilder.bind(queue).to(exchange).with("delay.#").noargs();
    }
}
```



三、producer创建发送消息测试类

Java | 复制代码

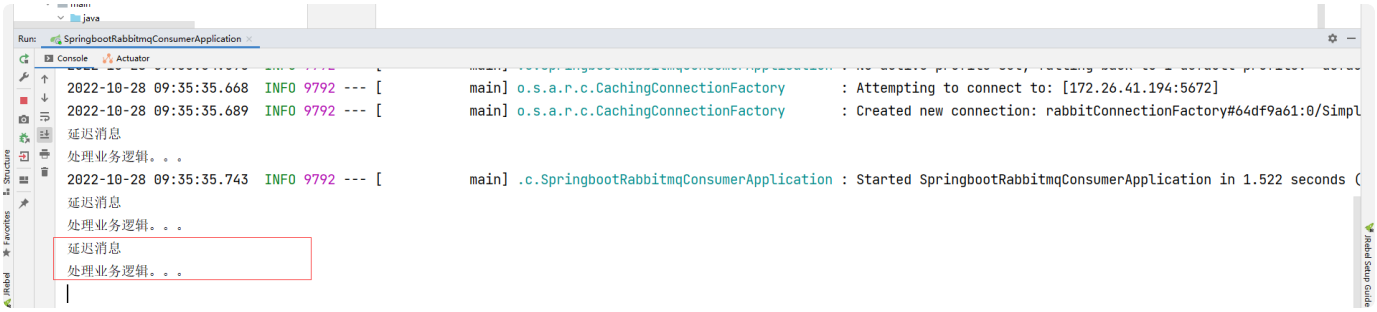
```java
/**
 * 高级特性（延迟队列）-发送消息测试类
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/28 10:35
 */
@SpringBootTest
@RunWith(SpringRunner.class)
class RabbitmqProducerDelaytTests {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    /**
     * 发送测试延迟消息
     * 1.过期时间
     *
     * @author zhichao.jiang01@hand-china.com  2022/10/28 10:36
     */
    @Test
    void testDelay() throws InterruptedException {
        // 1.过期时间
        rabbitTemplate.convertAndSend(DelaySeniorConfig.EXCHANGE_F_DELAY_NAME, "boot.delay.ccc", "延迟消息");
        //倒计时
        for (int i = 1; i <= 10; i++) {
            Thread.sleep(1000);
            System.out.println(i);
        }
    }


}
```



四、consumer创建消息监听类

Java | 复制代码

```java
/**
 * springboot-rabbitmq-延迟队列-监听类
 * 绑定死信队列
 *
 * @author zhichao.jiang01@hand-china.com 2022/10/28 10:35
 */
@Component
public class DelayListener {
    /**
     * Consumer ACK机制:
     * 1. 设置手动签收。listener:
     *                   direct:
     *                     acknowledge-mode: manual(yml文件设置)
     * 2. @RabbitListener添加ackMode模式为MANUAL
     * 3. 如果消息成功处理，则调用channel的basicAck()签收
     * 4. 如果消息处理失败，则调用channel的basicNack()拒绝签收，broker重新发送给consumer
     *
     */
    @RabbitListener(queues = "delay_queue",ackMode = "MANUAL")
    public void listenerSeniorQueue(Message message, Channel channel) throws IOException{
        long deliveryTag = message.getMessageProperties().getDeliveryTag();
        try {
            //1.接收转换消息
            System.out.println(new String(message.getBody()));

            //2.处理业务逻辑
            System.out.println("处理业务逻辑。。。");
            //假如出错
            //int a=3/0;
            //3.手动签收消息
            channel.basicAck(deliveryTag, true);
        }catch (Exception e){
            //4.拒绝签收
            /*
            第三个参数requeue：重回队列。如果设置为true，则消息回到死信队列
             */
            channel.basicNack(deliveryTag,true,true);
        }
    }

}
```



延迟消息小结

1. 延迟队列 指消息进入队列后，可以被延迟一定时间，再进行消费。

2. RabbitMQ没有提供延迟队列功能，但是可以使用： TTL + DLX 来实现延迟队列效果。

RabbitMQ日志

一、RabbitMQ日志简介

RabbitMQ默认日志存放路径： /var/log/rabbitmq/rabbit@xxx.log

没有权限在命令前面加上sudo

日志包含了RabbitMQ的版本号、Erlang的版本号、RabbitMQ服务节点名称、cookie的hash值、RabbitMQ配置文件地址、内存限制、磁盘限制、默认账户guest的创建以及权限配置等等。

二、rabbitmqctl监控与日志

没有权限加上sudo

查看队列

rabbitmqctl list_queues

查看exchanges

rabbitmqctl list_exchanges

查看用户

rabbitmqctl list_users

查看连接

rabbitmqctl list_connections

查看消费者信息

rabbitmqctl list_consumers

查看环境变量

rabbitmqctl environment

查看未被确认的队列

rabbitmqctl list_queues name messages_unacknowledged

查看单个队列的内存使用

rabbitmqctl list_queues name memory

查看准备就绪的队列

rabbitmqctl list_queues name messages_ready

三、消息追踪

在使用任何消息中间件的过程中，难免会出现某条消息异常丢失的情况。对于RabbitMQ而言，可能是因为生产者或消费者与RabbitMQ断开了连接，而它们与RabbitMQ又采用了不同的确认机制；也有可能是因为交换器与队列之间不同的转发策略；甚至是交换器并没有与任何队列进行绑定，生产者又不感知或者没有采取相应的措施；另外RabbitMQ本身的集群策略也可能导致消息的丢失。这个时候就需要有一个较好的机制跟踪记录消息的投递过程，以此协助开发和运维人员进行问题的定位。

在RabbitMQ中可以使用Firehose和rabbitmq_tracing插件功能来实现消息追踪。

消息追踪-Firehose

firehose的机制是将生产者投递给rabbitmq的消息，rabbitmq投递给消费者的消息按照指定的格式发送到默认的exchange上。这个默认的exchange的名称为amq.rabbitmq.trace，它是一个topic类型的exchange。发送到这个exchange上的消息的routing key为 publish.exchangename 和 deliver.queuename。其中exchangename和queuename为实际exchange和queue的名称，分别对应生产者投递到exchange的消息，和消费者从queue上获取的消息。

注意：打开 trace 会影响消息写入功能，适当打开后请关闭。

sudo rabbitmqctl trace_on：开启Firehose命令

sudo rabbitmqctl trace_off：关闭Firehose命令

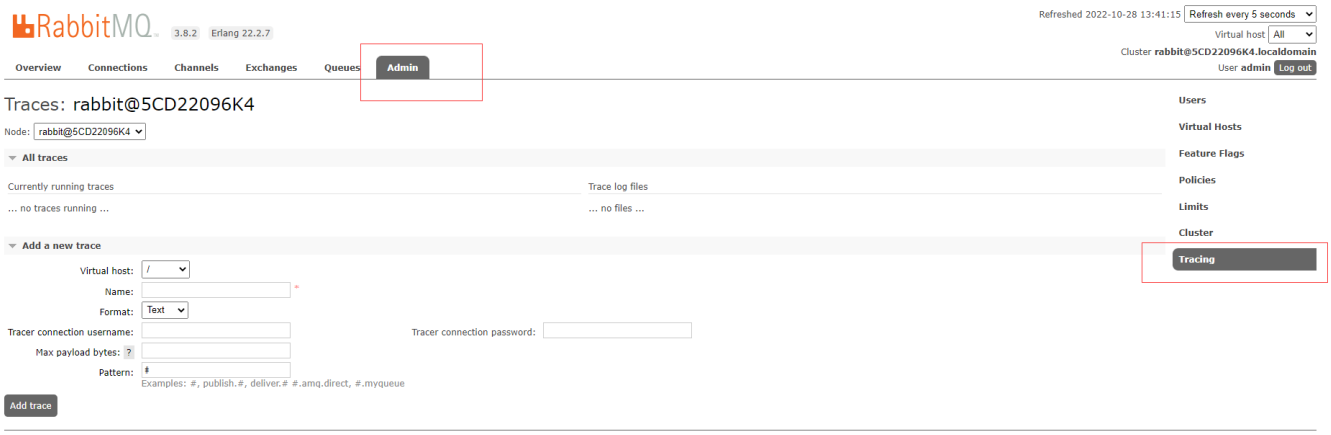
消息追踪-rabbitmq_tracing

rabbitmq_tracing和Firehose在实现上如出一辙，只不过rabbitmq_tracing的方式比Firehose多了一层GUI的包装，更容易使用和管理。

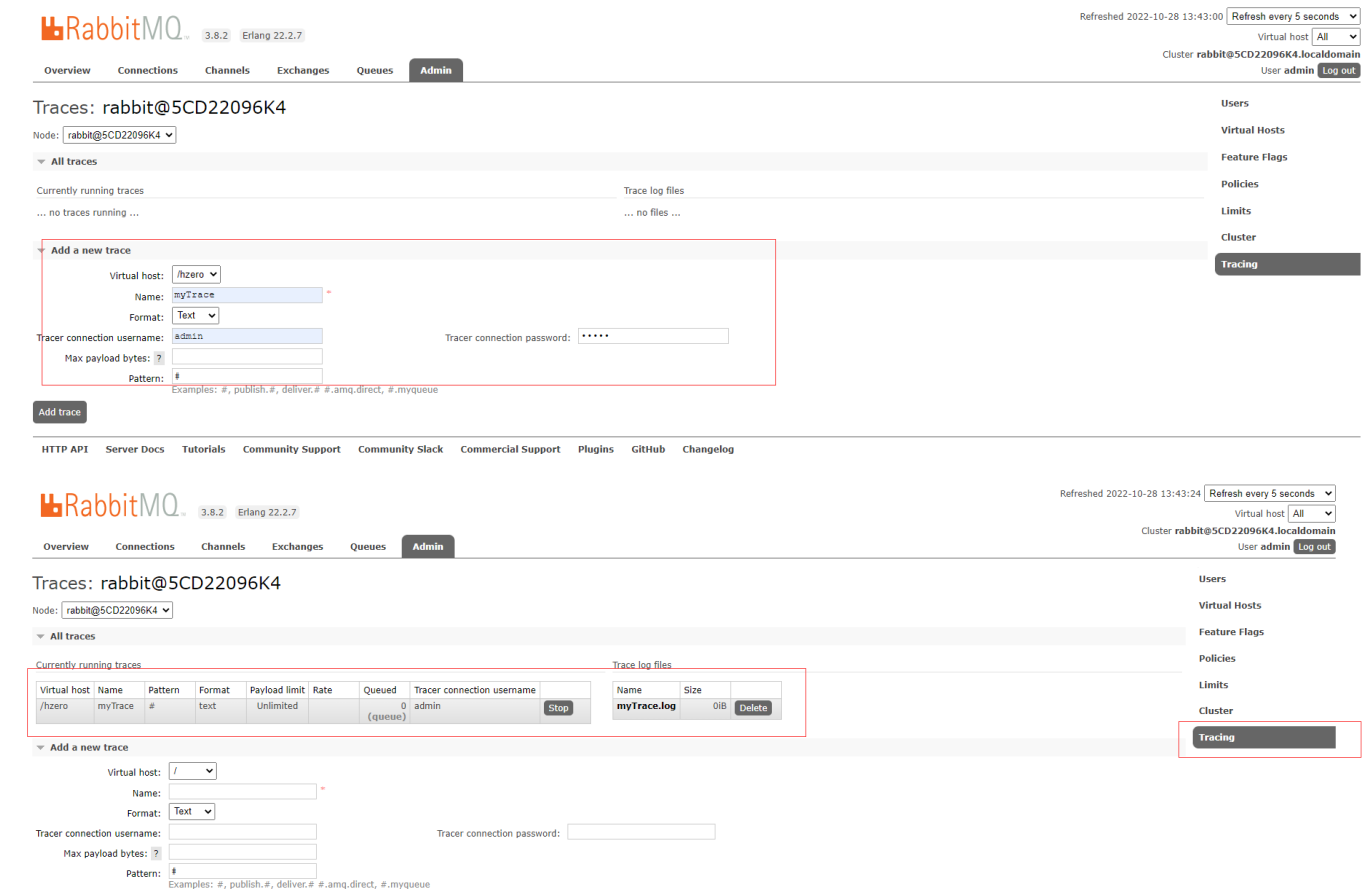
启用插件：sudo rabbitmq-plugins enable rabbitmq_tracing

```
/
/hzero
ubuntu@5CD22096K4:/var/log/rabbitmq$ sudo rabbitmqctl trace_on
[sudo] password for ubuntu:
Starting tracing for vhost "/" ...
Trace enabled for vhost /
ubuntu@5CD22096K4:/var/log/rabbitmq$ sudo rabbitmq-plugins enable rabbitmq_tracing
Enabling plugins on node rabbit@5CD22096K4:
rabbitmq_tracing
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_tracing
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@5CD22096K4 ...
The following plugins have been enabled:
  rabbitmq_tracing

started 1 plugins.
```

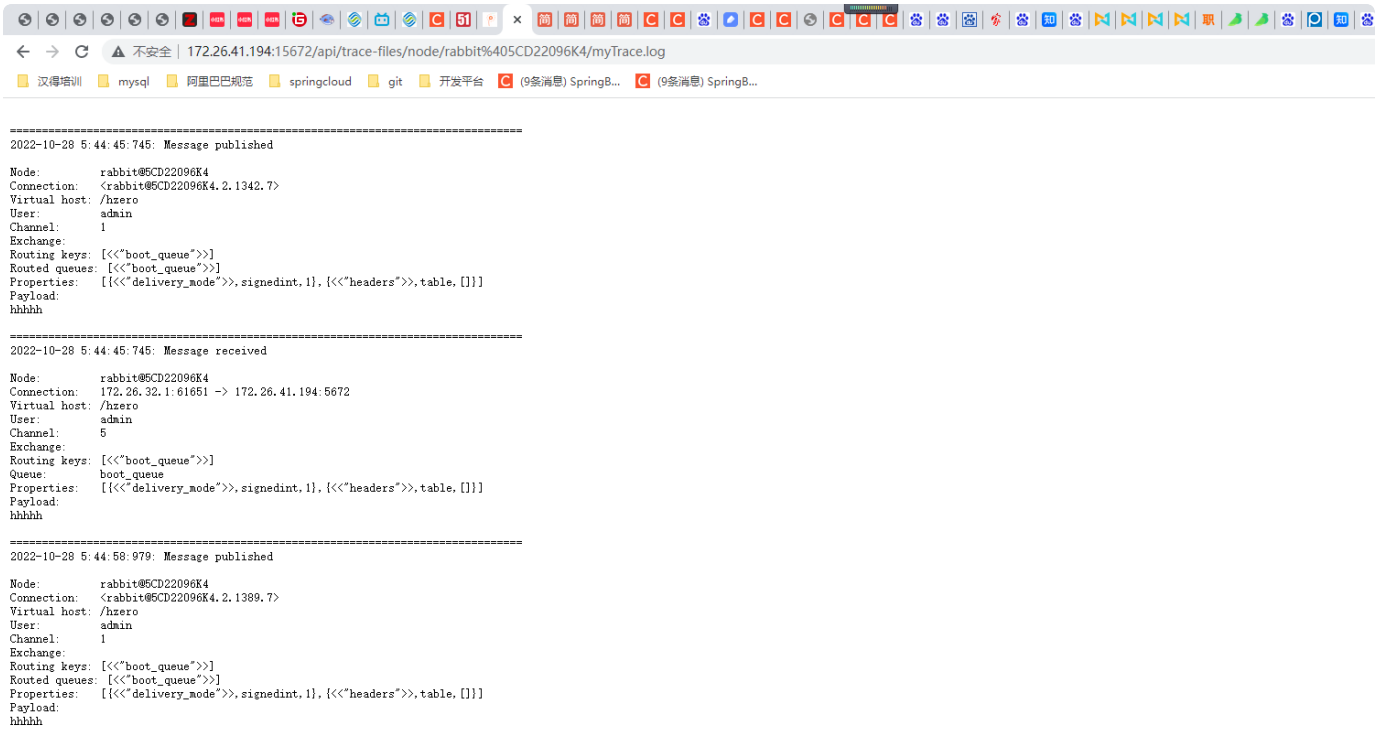
平台就可以现在这个GUI界面了


RabbitMQ 3.8.2 Erlang 22.2.7
Refreshed 2022-10-28 13:41:15 Refresh every 5 seconds
Virtual host All
Cluster rabbit@5CD22096K4.localdomain
User admin Log out
Overview Connections Channels Exchanges Queues Admin
Traces: rabbit@5CD22096K4
Node: rabbit@5CD22096K4
All traces
Currently running traces
... no traces running ...
Trace log files
... no files ...
Add a new trace
Virtual host: /
Name:
Format: Text
Tracer connection username:
Tracer connection password:
Max payload bytes:
Pattern: #
Examples: #, publish.#, deliver.# #.amq.direct, #.myqueue
Add trace
Users
Virtual Hosts
Feature Flags
Policies
Limits
Cluster
Tracing

创建消息追踪日志文件


RabbitMQ 3.8.2 Erlang 22.2.7
Refreshed 2022-10-28 13:43:00 Refresh every 5 seconds
Virtual host All
Cluster rabbit@5CD22096K4.localdomain
User admin Log out
Overview Connections Channels Exchanges Queues Admin
Traces: rabbit@5CD22096K4
Node: rabbit@5CD22096K4
All traces
Currently running traces
... no traces running ...
Trace log files
... no files ...
Add a new trace
Virtual host: /hzero
Name: myTrace
Format: Text
Tracer connection username: admin
Tracer connection password: •••••
Max payload bytes:
Pattern: #
Examples: #, publish.#, deliver.# #.amq.direct, #.myqueue
Add trace
Users
Virtual Hosts
Feature Flags
Policies
Limits
Cluster
Tracing
HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

RabbitMQ 3.8.2 Erlang 22.2.7
Refreshed 2022-10-28 13:43:24 Refresh every 5 seconds
Virtual host All
Cluster rabbit@5CD22096K4.localdomain
User admin Log out
Overview Connections Channels Exchanges Queues Admin
Traces: rabbit@5CD22096K4
Node: rabbit@5CD22096K4
All traces
Currently running traces
Virtual host Name Pattern Format Payload limit Rate Queued Tracer connection username
/hzero myTrace # text Unlimited 0 (queue) admin Stop
Trace log files
Name Size
myTrace.log 0iB Delete
Add a new trace
Virtual host: /
Name:
Format: Text
Tracer connection username:
Tracer connection password:
Max payload bytes:
Pattern: #
Examples: #, publish.#, deliver.# #.amq.direct, #.myqueue
Users
Virtual Hosts
Feature Flags
Policies
Limits
Cluster
Tracing



注意：在生产环境慎用，会影响性能！用完就关闭

关闭插件：sudo rabbitmq-plugins disable rabbitmq_tracing