5-7 利用RabbitMQ实现消息投递削峰填谷

一、导入依赖库

在 `pom.xml` 文件中添加RabbitMQ的依赖库

```
<dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>5.9.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

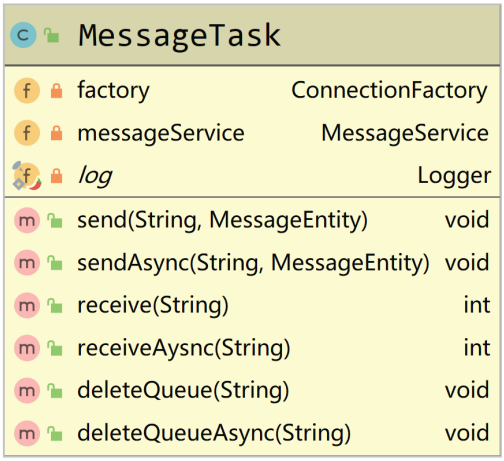
二、创建RabbitMQ配置类

连接 `RabbitMQ` 需要用到 `ConnectionFactory` ，所以我们要自己创建好 `ConnectionFactory` 对象然后注册给Spring框架，这就需要我们创建 `RabbitMQConfig` 类。

```
@Configuration
public class RabbitMQConfig {
    @Bean
    public ConnectionFactory getFactory() {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("192.168.99.101"); //Linux主机的IP地址
        factory.setPort(5672); //RabbitMQ端口号
        return factory;
    }
}
```

三、创建消息任务类

以前我们使用异步多线程的方式发送邮件，那么这次我们要创建的多线程任务类是用来收发RabbitMQ消息的，而且内部包含了同步执行和异步执行两种方式。



```
@Slf4j
@Component
public class MessageTask {
    @Autowired
    private ConnectionFactory factory;
    @Autowired
    private MessageService messageService;
    /**
     * 同步发送消息
     *
     * @param topic  主题
     * @param entity 消息对象
     */
    public void send(String topic, MessageEntity entity) {
        String id = messageService.insertMessage(entity); //向MongoDB保存消息数据，返回消息ID
        //向RabbitMQ发送消息
        try (Connection connection = factory.newConnection();
             Channel channel = connection.createChannel()) {
            //连接到某个Topic
            channel.queueDeclare(topic, true, false, false, null);
            HashMap header = new HashMap(); //存放属性数据
            header.put("messageId", id);
            //创建AMQP协议参数对象，添加附加属性
            AMQP.BasicProperties properties = new AMQP.BasicProperties().builder().headers(header).build();
            channel.basicPublish("", topic, properties, entity.getMsg().getBytes());
            log.debug("消息发送成功");
        } catch (Exception e) {
            log.error("执行异常", e);
            throw new EmosException("向MQ发送消息失败");
        }
    }
    /**
     * 异步发送消息
     *
     * @param topic  主题
     * @param entity
     */
    @Async
    public void sendAsync(String topic, MessageEntity entity) {
        send(topic, entity);
    }
    /**
     * 同步接收数据
     *
     * @param topic 主题
     * @return 接收消息数量
     */
    public int receive(String topic) {
        int i = 0;
        try (//接收消息数据
             Connection connection = factory.newConnection();
             Channel channel = connection.createChannel()) {
            // 从队列中获取消息，不自动确认
```

```
54.             channel.queueDeclare(topic, true, false, false, null);
55.             //Topic中有多少条数据未知，所以使用死循环接收数据，直到接收不到消息，退出死循环
56.             while (true) {
57.                 //创建响应接收数据，禁止自动发送Ack应答
58.                 GetResponse response = channel.basicGet(topic, false);
59.                 if (response != null) {
60.                     AMQP.BasicProperties properties = response.getProps();
61.                     Map<String, Object> header = properties.getHeaders(); //获取附加属性对
    象
62.                     String messageId = header.get("messageId").toString();
63.                     byte[] body = response.getBody();//获取消息正文
64.                     String message = new String(body);
65.                     log.debug("从RabbitMQ接收的消息：" + message);
66.                     MessageRefEntity entity = new MessageRefEntity();
67.                     entity.setMessageId(messageId);
68.                     entity.setReceiverId(Integer.parseInt(topic));
69.                     entity.setReadFlag(false);
70.                     entity.setLastFlag(true);
71.                     messageService.insertRef(entity); //把消息存储在MongoDB中
72.                     //数据保存到MongoDB后，才发送Ack应答，让Topic删除这条消息
73.                     long deliveryTag = response.getEnvelope().getDeliveryTag();
74.                     channel.basicAck(deliveryTag, false);
75.                     i++;
76.                 } else {
77.                     break; //接收不到消息，则退出死循环
78.                 }
79.             }
80.         } catch (Exception e) {
81.             log.error("执行异常", e);
82.         }
83.         return i;
84.     }
85.     /**
86.      * 异步接收数据
87.      *
88.      * @param topic
89.      * @return
90.      */
91.     @Async
92.     public int receiveAysnc(String topic) {
93.         return receive(topic);
94.     }
95.     /**
96.      * 同步删除消息队列
97.      *
98.      * @param topic 主题
99.      */
100.    public void deleteQueue(String topic) {
101.        try (Connection connection = factory.newConnection();
102.             Channel channel = connection.createChannel()) {
103.            channel.queueDelete(topic);
104.            log.debug("消息队列成功删除");
105.        } catch (Exception e) {
106.            log.error("删除队列失败", e);
107.            throw new EmosException("删除队列失败");
```

```
108.         }
109.     }
110.     /**
111.      * 异步删除消息队列
112.      *
113.      * @param topic 主题
114.      */
115.     @Async
116.     public void deleteQueueAsync(String topic) {
117.         deleteQueue(topic);
118.     }
119. }
```