01_RabbitMQ+SpringBoot

Java

消息队列

SpringBoot + RabbitMQ

时间：2022年12月31日22:25:35

官方文档：

- API: https://www.rabbitmq.com/api-guide.html

AMQP 0-9-1 Model Explained — RabbitMQ

springboot整合rabbitmq 消费者Consumer 手动进行ack确认_小哇666的博客-CSDN博客_channel.basicack

RabbitMQ Java Client Library — RabbitMQ

使用 Java client 操作 RabbitMQ

rabbitMq 批量消费(pull 拉取模式)_craywen的博客-CSDN博客_rabbitmq批量消费

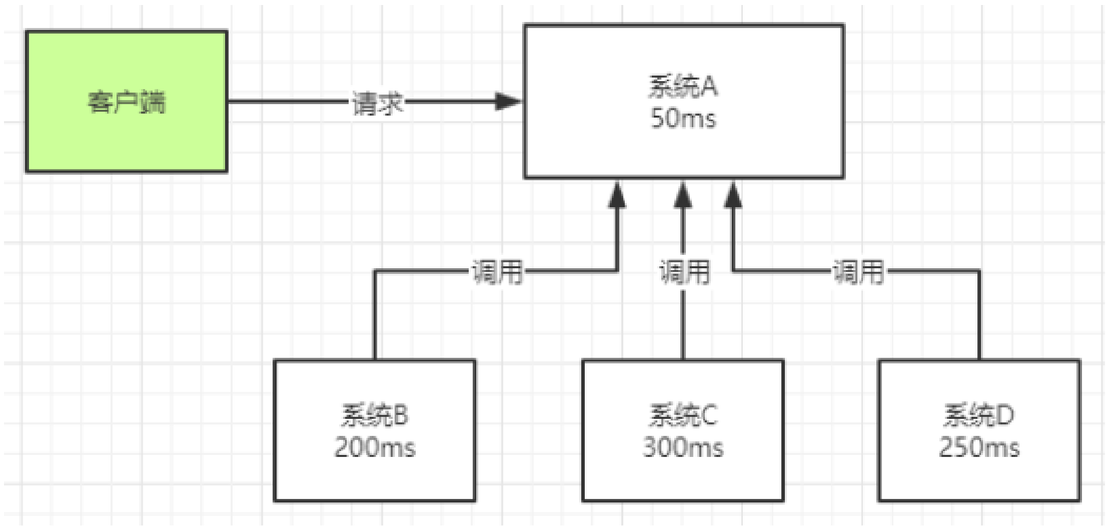
一、为什么要使用消息队列

① 解耦

> 假设有系统B、C、D都需要系统A的数据，于是系统A调用三个方法发送数据到B、C、D。这时，系统D不需要了，那就需要在系统A把相关的代码删掉。假设这时有个新的系统E需要数据，这时系统A又要增加调用系统E的代码。为了降低这种强耦合，就可以使用MQ，**系统A只需要把数据发送到MQ，其他系统如果需要数据，则从MQ中获取即可。**

② 异步

> 一个客户端请求发送进来，系统A会调用系统B、C、D三个系统，同步请求的话，响应时间就是系统A、B、C、D的总和，也就是800ms。**如果使用MQ，系统A发送数据到MQ，然后就可以返回响应给客户端，不需要再等待系统B、C、D的响应，可以大大地提高性能**。对于一些非必要的业务，比如发送短信，发送邮件等等，就可以采用MQ。



③ 削峰

> 这其实是MQ一个很重要的应用。降低数据库请求峰值以避免数据库崩溃导致的服务瘫痪。
>
> 假设系统A在某一段时间请求数暴增，有5000个请求发送过来，系统A这时就会发送5000条SQL进入MySQL进行执行，MySQL对于如此庞大的请求当然处理不过来，MySQL就会崩溃，导致系统瘫痪。**如果使用MQ，系统A不再是直接发送SQL到数据库，而是把数据发送到MQ，MQ短时间积压数据是可以接受的，然后由消费者每次拉取2000条进行处理，防止在请求峰值时期大量的请求直接发送到MySQL导致系统崩溃。**

二、什么是RabbitMQ?

1. 基本介绍

> RabbitMQ是一款使用Erlang语言开发的，实现AMQP(高级消息队列协议)的开源消息中间件。
>
> - Erlang → 一款面向并发的编程语言
>   - 为什么国内Erlang不是很火?
>   - https://developer.aliyun.com/article/229322

2. 什么优点?

- 可靠性
    - 支持持久化、传输确认、发布确认
- 灵活的消息分发策略
    - 简单模式、工作队列模式、发布订阅模式、路由模式、通配符模式
- 支持集群部署
- 支持多语言
- 支持多消息队列协议
    - STOMP、MQTT
- 支持插件机制

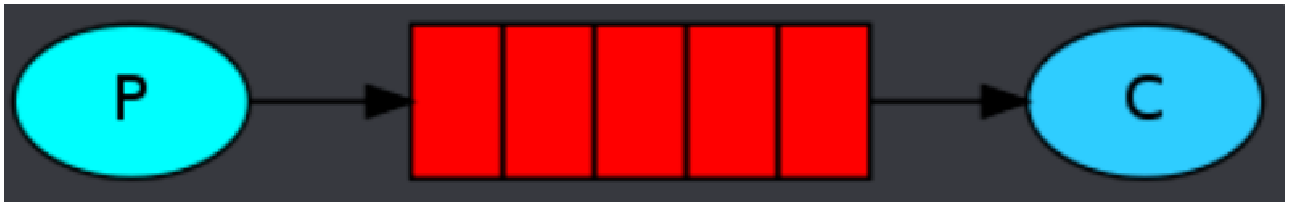
- 可视化管理界面

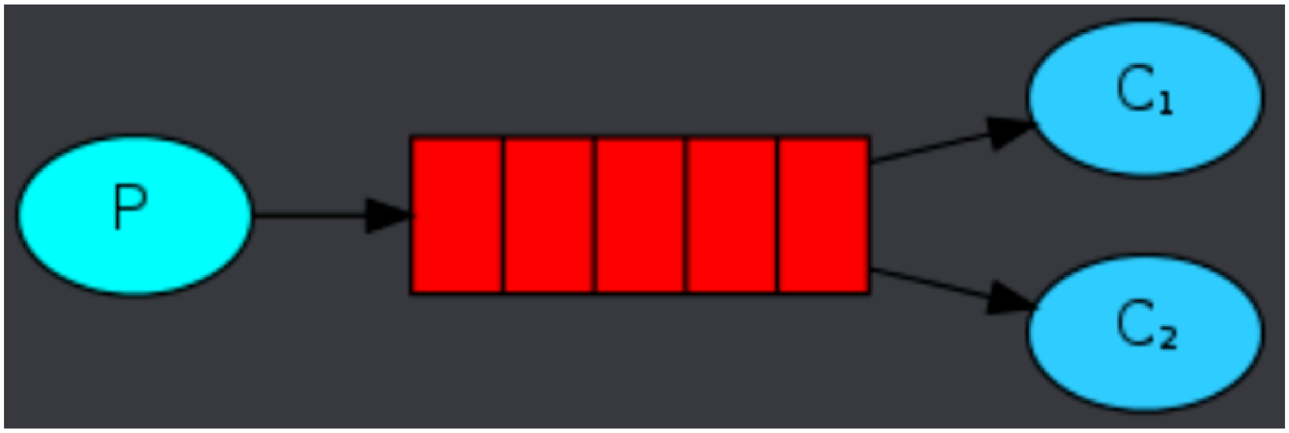
3. RabbitMQ组成部分

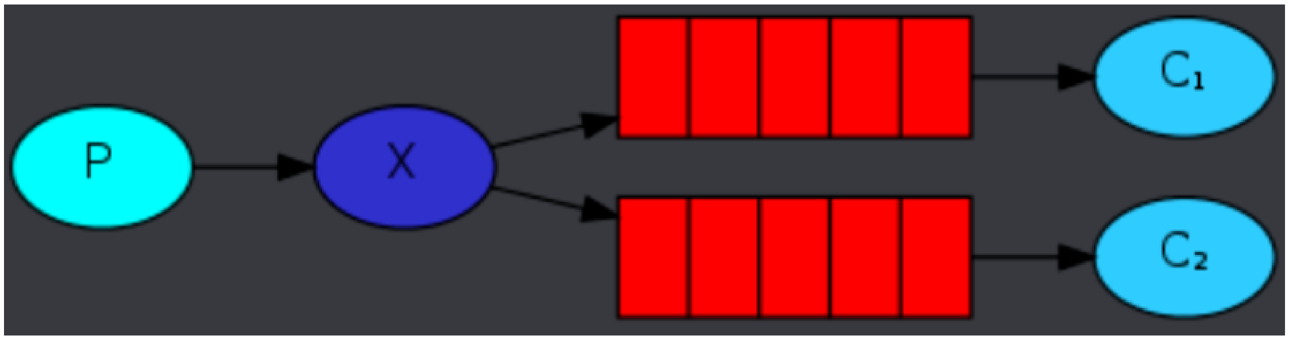
- Broker： 消息队列服务进程。此进程包括两个部分：Exchange和Queue。
- Exchange： 消息队列交换机。**按一定的规则将消息路由转发到某个队列。**
- Queue： 消息队列，存储消息的队列。
- Producer： 消息生产者。生产方客户端将消息同交换机路由发送到队列中。
- Consumer： 消息消费者。消费队列中存储的消息。

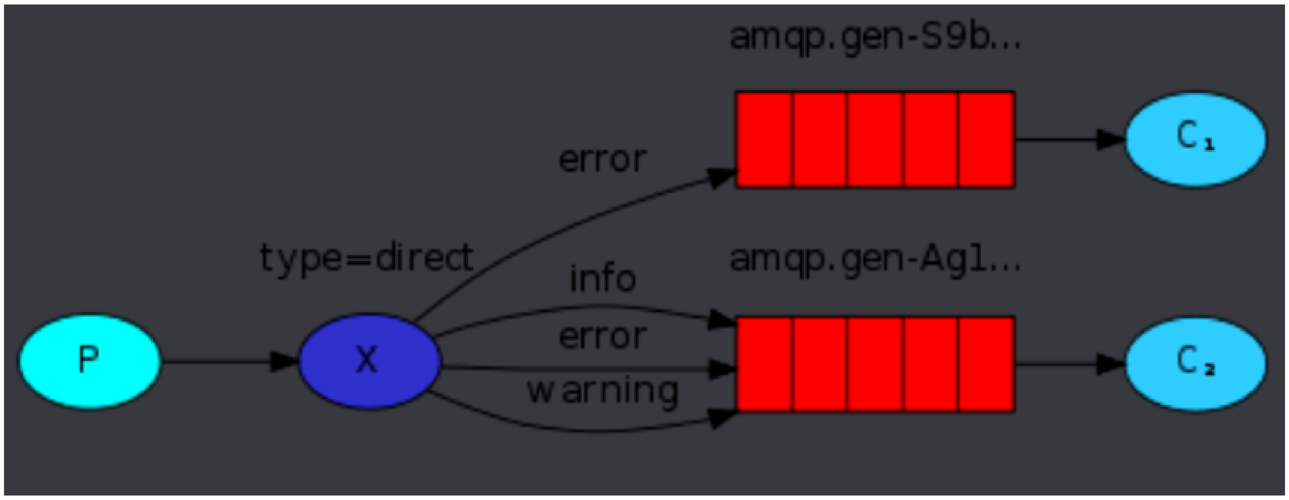
4. 消息发送模式（消息分发策略）

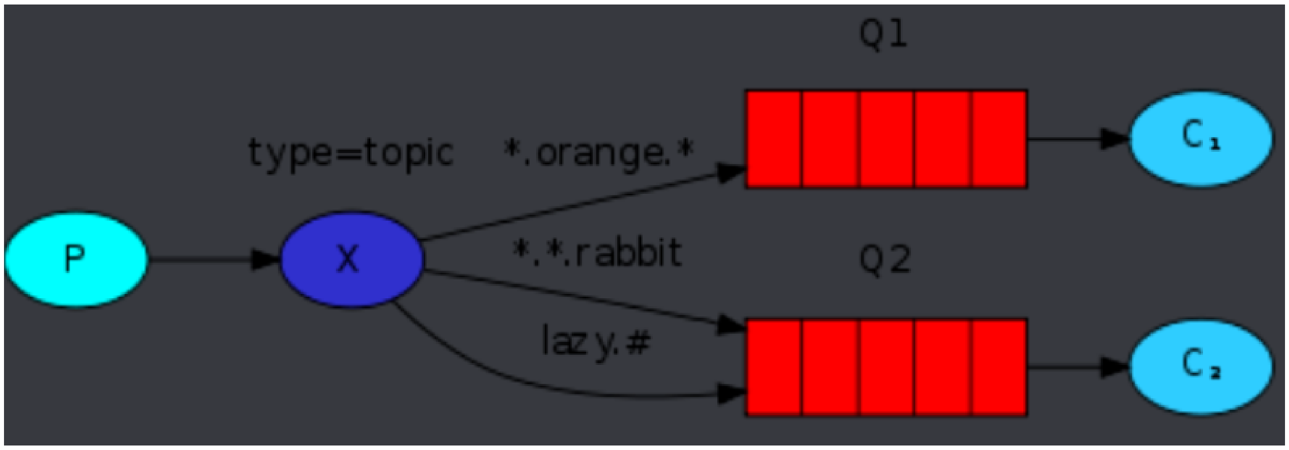
- 简单模式、工作队列模式、发布订阅模式、路由模式、通配符模式











三、SpringBoot集成RabbitMQ

> 推荐阅读：https://cloud.tencent.com/developer/article/1947188

1. Docker部署

> https://hub.docker.com/_/rabbitmq

```
docker pull rabbitmq:3.11.5-management
```

```
docker run --cpus=3 --memory=2GB -d --hostname my-rabbit-host-1 --name my-rabbit-1 -e
RABBITMQ_DEFAULT_USER=admin -e RABBITMQ_DEFAULT_PASS=Rabbitmq12345 -p 15672:15672 -p 5672:5672
rabbitmq:3.11.5-management
docker run --cpus=3 --memory=2GB -d --hostname my-rabbit-host-2 --name my-rabbit-2 -e
RABBITMQ_DEFAULT_USER=admin -e RABBITMQ_DEFAULT_PASS=Rabbitmq12345 -p 15672:15672 -p 5672:5672
rabbitmq:3.11.5-management
docker run --cpus=3 --memory=2GB -d --hostname my-rabbit-host-3 --name my-rabbit-3 -e
RABBITMQ_DEFAULT_USER=admin -e RABBITMQ_DEFAULT_PASS=Rabbitmq12345 -p 15672:15672 -p 5672:5672
rabbitmq:3.11.5-management
```

- -h --hostname

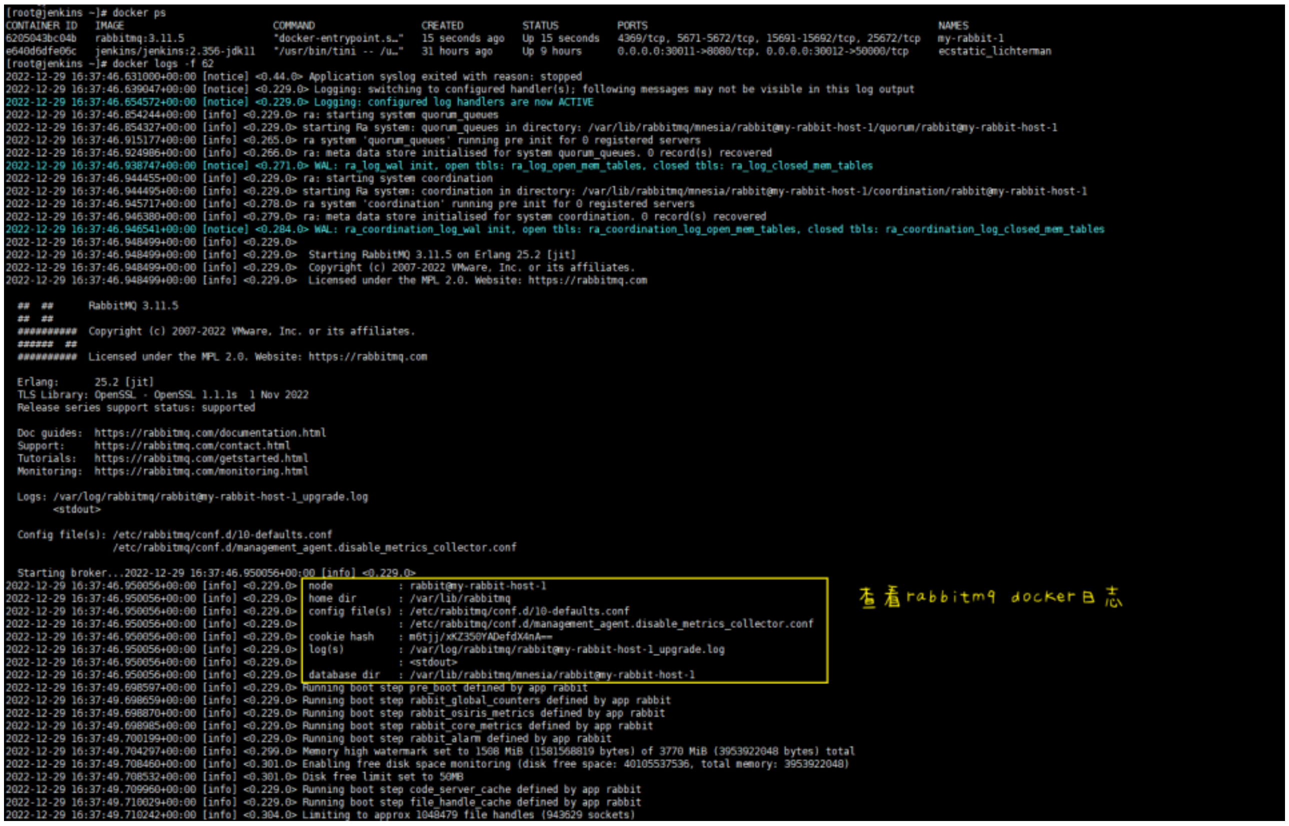
```
--cpus=3
--memory=2GB
```

这将启动侦听默认端口 5672 的 RabbitMQ 容器。如果你给它一分钟，然后做，你会在输出中看到一个类似于: `docker logs some-rabbit`

```
=INFO REPORT==== 6-Jul-2015::20:47:02 ===
node           : rabbit@my-rabbit
home dir       : /var/lib/rabbitmq
config file(s) : /etc/rabbitmq/rabbitmq.config
cookie hash    : UoNOcDhfxW9uoZ92wh6BjA==
log            : tty
sasl log       : tty
database dir   : /var/lib/rabbitmq/mnesia/rabbit@my-rabbit
```

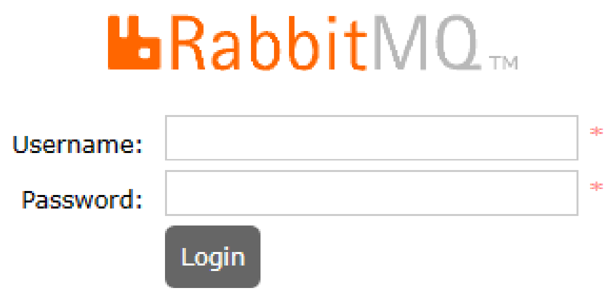
请注意那里，特别是它将我的"节点名称"附加到文件存储的末尾。默认情况下，此映像使所有卷全部生效。`database dir /var/lib/rabbitmq`

查看rabbitmq docker日志

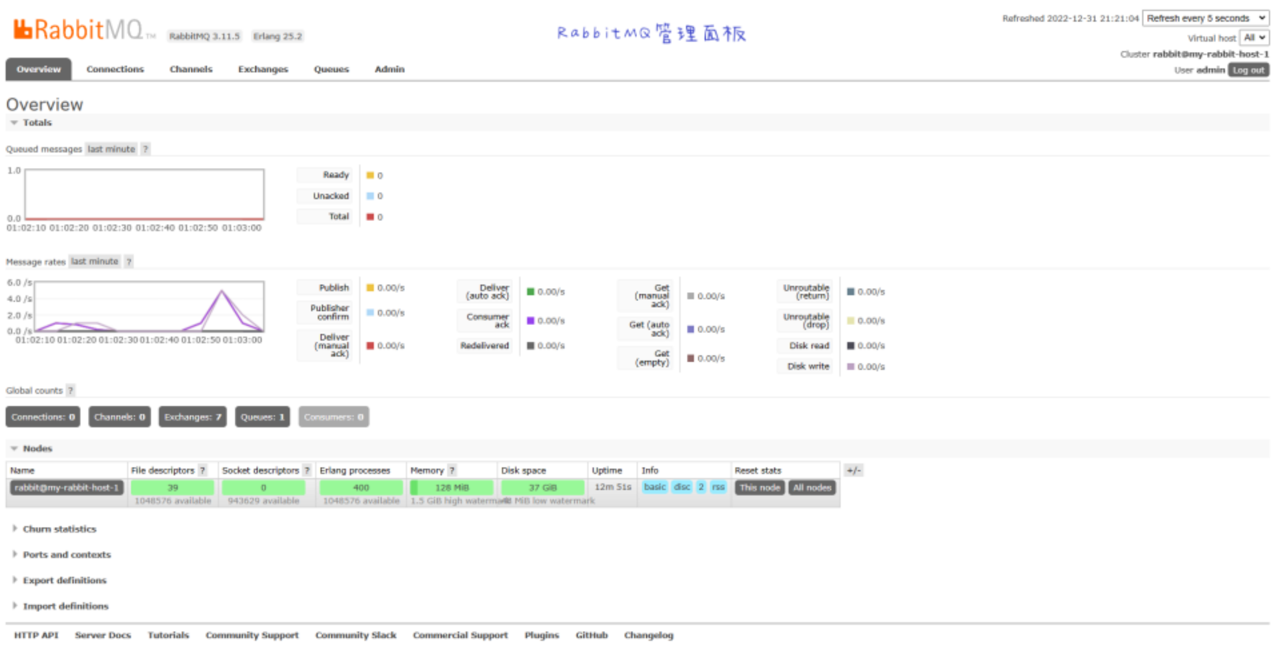


```
[root@jenkins ~]# docker ps
CONTAINER ID   IMAGE                        COMMAND                  CREATED          STATUS          PORTS                                                        NAMES
6205043bc04b   rabbitmq:3.11.5              "docker-entrypoint.s…"   15 seconds ago   Up 15 seconds   4369/tcp, 5671-5672/tcp, 15691-15692/tcp, 25672/tcp          my-rabbit-1
e640d6dfe06c   jenkins/jenkins:2.356-jdk11  "/usr/bin/tini -- /u…"   31 hours ago     Up 9 hours      0.0.0.0:30011->8080/tcp, 0.0.0.0:30012->50000/tcp            ecstatic_lichterman
[root@jenkins ~]# docker logs -f 62
2022-12-29 16:37:46.631000+00:00 [notice] <0.44.0> Application syslog exited with reason: stopped
2022-12-29 16:37:46.639047+00:00 [notice] <0.229.0> Logging: switching to configured handler(s); following messages may not be visible in this log output
2022-12-29 16:37:46.654572+00:00 [notice] <0.229.0> Logging: configured log handlers are now ACTIVE
2022-12-29 16:37:46.854244+00:00 [info] <0.229.0> ra: starting system quorum_queues
2022-12-29 16:37:46.854327+00:00 [info] <0.229.0> starting Ra system: quorum_queues in directory: /var/lib/rabbitmq/mnesia/rabbit@my-rabbit-host-1/quorum/rabbit@my-rabbit-host-1
2022-12-29 16:37:46.915177+00:00 [info] <0.265.0> ra system 'quorum_queues' running pre init for 0 registered servers
2022-12-29 16:37:46.924986+00:00 [info] <0.266.0> ra: meta data store initialised for system quorum_queues. 0 record(s) recovered
2022-12-29 16:37:46.938747+00:00 [notice] <0.271.0> WAL: ra_log_wal init, open tbls: ra_log_open_mem_tables, closed tbls: ra_log_closed_mem_tables
2022-12-29 16:37:46.944455+00:00 [info] <0.229.0> ra: starting system coordination
2022-12-29 16:37:46.944495+00:00 [info] <0.229.0> starting Ra system: coordination in directory: /var/lib/rabbitmq/mnesia/rabbit@my-rabbit-host-1/coordination/rabbit@my-rabbit-host-1
2022-12-29 16:37:46.945717+00:00 [info] <0.278.0> ra system 'coordination' running pre init for 0 registered servers
2022-12-29 16:37:46.946380+00:00 [info] <0.279.0> ra: meta data store initialised for system coordination. 0 record(s) recovered
2022-12-29 16:37:46.946541+00:00 [notice] <0.284.0> WAL: ra_coordination_log_wal init, open tbls: ra_coordination_log_open_mem_tables, closed tbls: ra_coordination_log_closed_mem_tables
2022-12-29 16:37:46.948499+00:00 [info] <0.229.0>
2022-12-29 16:37:46.948499+00:00 [info] <0.229.0>  Starting RabbitMQ 3.11.5 on Erlang 25.2 [jit]
2022-12-29 16:37:46.948499+00:00 [info] <0.229.0>  Copyright (c) 2007-2022 VMware, Inc. or its affiliates.
2022-12-29 16:37:46.948499+00:00 [info] <0.229.0>  Licensed under the MPL 2.0. Website: https://rabbitmq.com

  ##  ##      RabbitMQ 3.11.5
  ##  ##
  ##########  Copyright (c) 2007-2022 VMware, Inc. or its affiliates.
  ######  ##
  ##########  Licensed under the MPL 2.0. Website: https://rabbitmq.com

  Erlang:      25.2 [jit]
  TLS Library: OpenSSL - OpenSSL 1.1.1s  1 Nov 2022
  Release series support status: supported

  Doc guides:  https://rabbitmq.com/documentation.html
  Support:     https://rabbitmq.com/contact.html
  Tutorials:   https://rabbitmq.com/getstarted.html
  Monitoring:  https://rabbitmq.com/monitoring.html

  Logs: /var/log/rabbitmq/rabbit@my-rabbit-host-1_upgrade.log
        <stdout>

  Config file(s): /etc/rabbitmq/conf.d/10-defaults.conf
                  /etc/rabbitmq/conf.d/management_agent.disable_metrics_collector.conf

  Starting broker...2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>
2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>  node           : rabbit@my-rabbit-host-1
2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>  home dir       : /var/lib/rabbitmq
2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>  config file(s) : /etc/rabbitmq/conf.d/10-defaults.conf
2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>                 : /etc/rabbitmq/conf.d/management_agent.disable_metrics_collector.conf
2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>  cookie hash    : m6tjj/xKZ350YADefdX4nA==
2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>  log(s)         : /var/log/rabbitmq/rabbit@my-rabbit-host-1_upgrade.log
2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>                 : <stdout>
2022-12-29 16:37:46.950056+00:00 [info] <0.229.0>  database dir   : /var/lib/rabbitmq/mnesia/rabbit@my-rabbit-host-1
2022-12-29 16:37:49.698597+00:00 [info] <0.229.0> Running boot step pre_boot defined by app rabbit
2022-12-29 16:37:49.698659+00:00 [info] <0.229.0> Running boot step rabbit_global_counters defined by app rabbit
2022-12-29 16:37:49.698870+00:00 [info] <0.229.0> Running boot step rabbit_osiris_metrics defined by app rabbit
2022-12-29 16:37:49.698985+00:00 [info] <0.229.0> Running boot step rabbit_core_metrics defined by app rabbit
2022-12-29 16:37:49.700199+00:00 [info] <0.229.0> Running boot step rabbit_alarm defined by app rabbit
2022-12-29 16:37:49.704297+00:00 [info] <0.299.0> Memory high watermark set to 1508 MiB (1581568819 bytes) of 3770 MiB (3953922048 bytes) total
2022-12-29 16:37:49.708460+00:00 [info] <0.301.0> Enabling free disk space monitoring (disk free space: 40105537536, total memory: 3953922048)
2022-12-29 16:37:49.708532+00:00 [info] <0.301.0> Disk free limit set to 50MB
2022-12-29 16:37:49.709960+00:00 [info] <0.229.0> Running boot step code_server_cache defined by app rabbit
2022-12-29 16:37:49.710029+00:00 [info] <0.229.0> Running boot step file_handle_cache defined by app rabbit
2022-12-29 16:37:49.710242+00:00 [info] <0.304.0> Limiting to approx 1048479 file handles (943629 sockets)
```

登录管理面板



http://192.168.204.113:15672/



Refreshed 2022-12-31 21:21:04 Refresh every 5 seconds
Virtual host All
Cluster rabbit@my-rabbit-host-1
User admin Log out

Overview

Totals

Queued messages last minute ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last minute ?

Publish	🟨 0.00/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	■ 0.00/s	Unroutable (return)	■ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 0.00/s	Get (auto ack)	🟪 0.00/s	Unroutable (drop)	🟨 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	■ 0.00/s	Get (empty)	🟫 0.00/s	Disk read	■ 0.00/s
						Disk write	🟪 0.00/s

Global counts ?

Connections: 0 Channels: 0 Exchanges: 7 Queues: 1 Consumers: 0

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@my-rabbit-host-1	39 1048576 available	0 943629 available	400 1048576 available	128 MiB 1.5 GiB high waterma	37 GiB 48 MiB low watermark	12m 51s	basic disc 2 rss	This node All nodes

Churn statistics

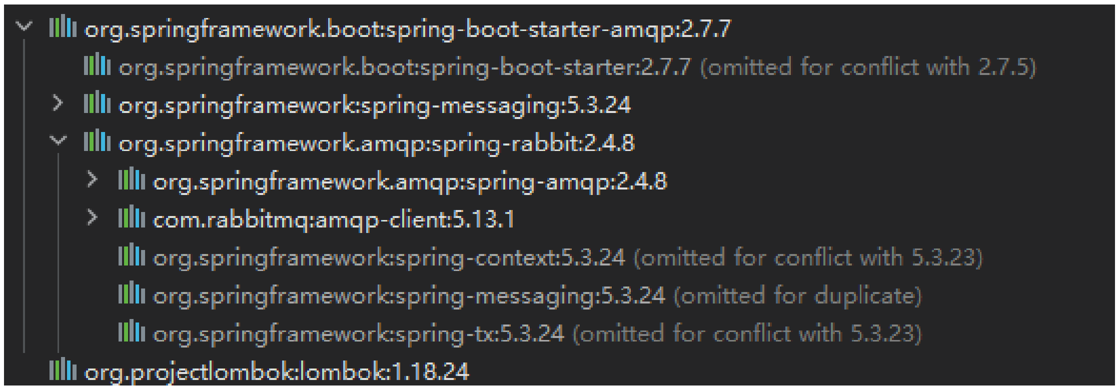
Ports and contexts

Export definitions

Import definitions

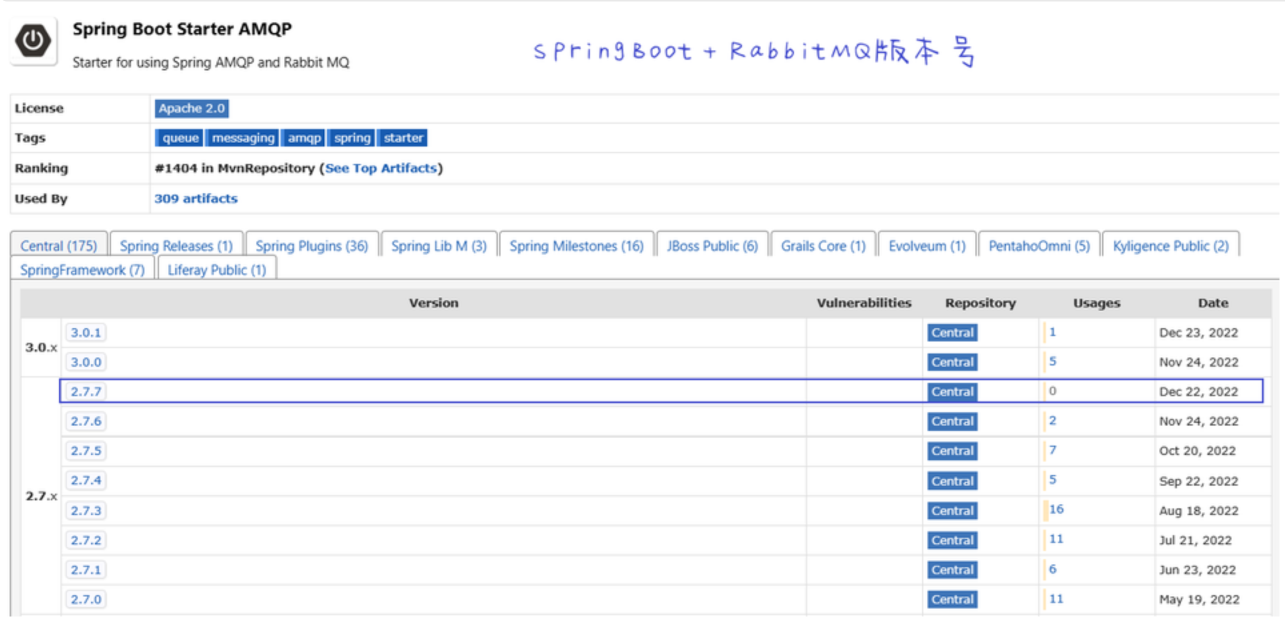
HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

2. 引入依赖



```
org.springframework.boot:spring-boot-starter-amqp:2.7.7
    org.springframework.boot:spring-boot-starter:2.7.7 (omitted for conflict with 2.7.5)
  > org.springframework:spring-messaging:5.3.24
    org.springframework.amqp:spring-rabbit:2.4.8
      > org.springframework.amqp:spring-amqp:2.4.8
      > com.rabbitmq:amqp-client:5.13.1
        org.springframework:spring-context:5.3.24 (omitted for conflict with 5.3.23)
        org.springframework:spring-messaging:5.3.24 (omitted for duplicate)
        org.springframework:spring-tx:5.3.24 (omitted for conflict with 5.3.23)
org.projectlombok:lombok:1.18.24
```

```xml
<!-- 消息队列 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
    <version>2.7.7</version>
</dependency>
```



Spring Boot Starter AMQP

Starter for using Spring AMQP and Rabbit MQ

SpringBoot + RabbitMQ版本号

License	Apache 2.0
Tags	queue messaging amqp spring starter
Ranking	#1404 in MvnRepository (See Top Artifacts)
Used By	309 artifacts

Central (175) | Spring Releases (1) | Spring Plugins (36) | Spring Lib M (3) | Spring Milestones (16) | JBoss Public (6) | Grails Core (1) | Evolveum (1) | PentahoOmni (5) | Kyligence Public (2) | SpringFramework (7) | Liferay Public (1)

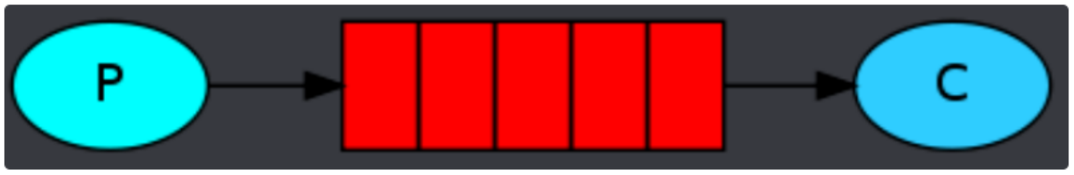
	Version	Vulnerabilities	Repository	Usages	Date
3.0.x	3.0.1		Central	1	Dec 23, 2022
	3.0.0		Central	5	Nov 24, 2022
2.7.x	2.7.7		Central	0	Dec 22, 2022
	2.7.6		Central	2	Nov 24, 2022
	2.7.5		Central	7	Oct 20, 2022
	2.7.4		Central	5	Sep 22, 2022
	2.7.3		Central	16	Aug 18, 2022
	2.7.2		Central	11	Jul 21, 2022
	2.7.1		Central	6	Jun 23, 2022
	2.7.0		Central	11	May 19, 2022

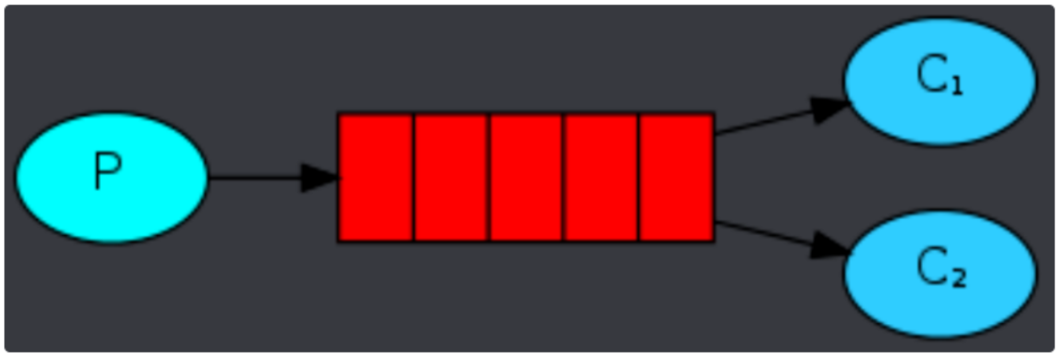
3. 配置文件（单机）

```
spring:
  # RabbitMQ服务地址
  rabbitmq:
    host: 192.168.204.113
    port: 5672
    username: admin
    password: Rabbitmq12345
```

4. Producer+Consumer

> RabbitTemplate#convertAndSend(String routingKey, Object message): void





```
import lombok.RequiredArgsConstructor;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Component;
```

```
@Component
@RequiredArgsConstructor
public class Producer {

    private final static String QUEUE_NAME = "notice_queue";
    private final RabbitTemplate rabbitTemplate;

    public void produce() {
        String message = "消息内容123456";
        System.out.println("生产者说： " + message);
        // String routingKey + Object message : void
        rabbitTemplate.convertAndSend(QUEUE_NAME, message);
    }
}
```

```
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
public class Consumer {
    private final static String QUEUE_NAME = "notice_queue";

    @RabbitHandler
    @RabbitListener(queuesToDeclare = @Queue(QUEUE_NAME))
    public void process(String message) {
        System.out.println("消费者A收到通知： " + message);
    }
}
```

```
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
public class Consumer1 {
    private final static String QUEUE_NAME = "notice_queue";

    @RabbitHandler
    @RabbitListener(queuesToDeclare = @Queue(QUEUE_NAME))
    public void process(String message) {
        System.out.println("消费者B收到通知： " + message);
    }
}
```

5. 消费者限制速度（生产者速度大大高于消费者速度，消费者不能不顾一切利用资源进行消费，此时必须手动签收）

- Springboot环境下RabbitMQ的批量消费(consumer-batch-enabled)_aludashic9的博客-CSDN博客_rabbitmq批量消费
- Spring+RabbitMq实现数据批量接收，批量操作_讲真话的猫的博客-CSDN博客_rabbitmq一次接收多条消息
- 深入理解RabbitMQ中的prefetch_count参数_LiZhen798的博客-CSDN博客_prefetch_count

5.1 配置文件

```
spring:
  # datasource:
  #    type: com.alibaba.druid.pool.DruidDataSource
  #    driver-class-name: com.mysql.cj.jdbc.Driver
  #    url: jdbc:mysql://192.168.204.112:23306/dev_test?
autoReconnect=true&useServerPreparedStmts=true&cachePrepStmts=true&rewriteBatchedStatements=true&allowMultiQu
```

5.2 配置类

```
import lombok.RequiredArgsConstructor;
import org.springframework.amqp.core.AcknowledgeMode;
import org.springframework.amqp.rabbit.config.SimpleRabbitListenerContainerFactory;
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
@RequiredArgsConstructor
public class RabbitMQConfig {
    private final CachingConnectionFactory connectionFactory;

    @Value("${spring.rabbitmq.listener.simple.prefetch}")
    int prefetchCount=100;
    @Value("${spring.rabbitmq.listener.simple.batch-size}")
    int batchSize=100;

    @Bean
    public SimpleRabbitListenerContainerFactory mqConsumerlistenerContainer() {
        SimpleRabbitListenerContainerFactory factory = new SimpleRabbitListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory);

        // consumer限制消费速率 (下方注释说明使用方式)
        // @RabbitHandler
        // @RabbitListener(queuesToDeclare = @Queue(QUEUE_NAME), containerFactory =
"mqConsumerlistenerContainer")

        // 手动签收 + 开启消费者批量消费数据
        factory.setAcknowledgeMode(AcknowledgeMode.MANUAL);
        factory.setConsumerBatchEnabled(true);
        factory.setBatchListener(true);
        // 设置批处理大小
        factory.setPrefetchCount(prefetchCount);
        factory.setBatchSize(batchSize);
        // 10s 没有数据写入队列时，消费端开始消费批数据
        factory.setReceiveTimeout(1000L * 10);
        return factory;
    }
}
```

5.3 消费者A（消费者实现方式一）

```
import com.rabbitmq.client.Channel;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.rabbit.listener.api.ChannelAwareMessageListener;
import org.springframework.stereotype.Component;

import java.io.IOException;
import java.util.List;

@Component
public class Consumer implements ChannelAwareMessageListener {
    private final String FANOUT_SMS_QUEUE = "fanout_sms_queue";
    int count = 0;

    @Override
    public void onMessage(Message message, Channel channel) throws Exception {

    }

    @RabbitListener(queues = FANOUT_SMS_QUEUE, containerFactory = "mqConsumerlistenerContainer")
    @Override
    public void onMessageBatch(List<Message> messages, Channel channel) {
        System.out.println(messages.size());
        System.out.println(Thread.currentThread().getName() + ">>> " + (++count));
        for (Message message : messages) {
            // System.out.println(">>> 短信消费者获取生产者消息:" + new String(message.getBody(), StandardCharsets.
UTF_8));
            // 手动签收
            try {
                channel.basicAck(message.getMessageProperties().getDeliveryTag(), true);
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        System.out.println("短信消费者执行结束....");
    }
}
```

5.4 消费者B（消费者实现方式二）

```
import com.rabbitmq.client.Channel;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets;
import java.util.List;

@Component
public class Consumer1 {
    private final String FANOUT_EMAIL_QUEUE = "fanout_email_queue";

    @RabbitListener(queuesToDeclare = @Queue(FANOUT_EMAIL_QUEUE), containerFactory =
"mqConsumerlistenerContainer")
    public void process(List<Message> messages, Channel channel) throws Exception {

        for (Message message : messages) {
            System.out.println(">>>> 邮件消费者获取生产者消息:" + new String(message.getBody(), StandardCharsets.
UTF_8));
            // 手动签收
            channel.basicAck(message.getMessageProperties().getDeliveryTag(), true);
        }

        System.out.println("邮件消费者执行结束....");
    }
}
```

5.5 生产者

```
import com.alibaba.fastjson2.JSONObject;
import lombok.RequiredArgsConstructor;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageBuilder;
import org.springframework.amqp.core.MessageProperties;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Component;

import java.util.UUID;

@Component
@RequiredArgsConstructor
public class Producer {
    private final RabbitTemplate rabbitTemplate;

    public void produce(String queueName) {
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("email", "22222@qq.com");
        jsonObject.put("phoneNumber", "12222222222");
        jsonObject.put("timestamp", System.currentTimeMillis());
        String jsonString = jsonObject.toJSONString();

        // 设置消息唯一id 保证每次重试消息id唯一
        Message message = MessageBuilder.withBody(jsonString.getBytes())
                .setContentType(MessageProperties.CONTENT_TYPE_JSON).setContentEncoding("utf-8")
                .setMessageId(UUID.randomUUID() + "").build();
        // rabbitTemplate.convertAndSend(queueName, message);
        rabbitTemplate.convertAndSend(queueName, "", message);
    }
}
```

5.6 消息发布订阅模型配置类

```
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.FanoutExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;

//Fanout 类型 发布订阅模式
@Component
public class FanoutConfig {

    // 邮件队列
    private final String FANOUT_EMAIL_QUEUE = ”fanout_email_queue”;

    // 短信队列
    private final String FANOUT_SMS_QUEUE = ”fanout_sms_queue”;
    // fanout 交换机
    private final String EXCHANGE_NAME = ”fanoutExchange”;

    // 1.定义邮件队列
    @Bean
    public Queue fanOutEamilQueue() {
        return new Queue(FANOUT_EMAIL_QUEUE);
    }

    // 2.定义短信队列
    @Bean
    public Queue fanOutSmsQueue() {
        return new Queue(FANOUT_SMS_QUEUE);
    }

    // 2.定义交换机
    @Bean
    FanoutExchange fanoutExchange() {
        return new FanoutExchange(EXCHANGE_NAME);
    }

    // 3.队列与交换机绑定邮件队列
    @Bean
    Binding bindingExchangeEmail(Queue fanOutEamilQueue, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(fanOutEamilQueue).to(fanoutExchange);
    }

    // 4.队列与交换机绑定短信队列
    @Bean
    Binding bindingExchangeSms(Queue fanOutSmsQueue, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(fanOutSmsQueue).to(fanoutExchange);
    }
}
```

5.7 测试类

```
import com.xii.mp.mq.Producer;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.concurrent.TimeUnit;

@SpringBootTest
public class AppTest1 {

    private final String EXCHANGE_NAME = ”fanoutExchange”;

    @Autowired
    private Producer producer;

    @Test
    public void test01() {
        for (int i = 0; i < 100; i++) {
            producer.produce(EXCHANGE_NAME);
            System.out.println("hello,rabbitmq-" + i + ": 消息发送成功！");
        }

        try {
            TimeUnit.SECONDS.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

四、Consumer手动签收消息

- 如果没有签收消息，新启动的客户端会从头开始消费消息。

> 推荐阅读：https://blog.51cto.com/u_15461374/5938036

```
<!-- 消息队列+fastjson2 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
    <version>2.7.7</version>
</dependency>
<dependency>
    <groupId>com.alibaba.fastjson2</groupId>
    <artifactId>fastjson2</artifactId>
    <version>2.0.21</version>
</dependency>
```

```
spring:
  # datasource:
  #   type: com.alibaba.druid.pool.DruidDataSource
  #   driver-class-name: com.mysql.cj.jdbc.Driver
  #   url: jdbc:mysql://192.168.204.112:23306/dev_test?
autoReconnect=true&useServerPreparedStmts=true&cachePrepStmts=true&rewriteBatchedStatements=true&allowMultiQu
```

```
import com.alibaba.fastjson2.JSONObject;
import lombok.RequiredArgsConstructor;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageBuilder;
import org.springframework.amqp.core.MessageProperties;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Component;

import java.util.UUID;

@Component
@RequiredArgsConstructor
public class Producer {
    private final RabbitTemplate rabbitTemplate;
    public void produce(String queueName) {
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("email", "22222@qq.com");
        jsonObject.put("timestamp", System.currentTimeMillis());
        String jsonString = jsonObject.toJSONString();
        System.out.println("jsonString:" + jsonString);
        // 设置消息唯一id 保证每次重试消息id唯一
        Message message = MessageBuilder.withBody(jsonString.getBytes())
                .setContentType(MessageProperties.CONTENT_TYPE_JSON).setContentEncoding("utf-8")
                .setMessageId(UUID.randomUUID() + "").build();
        rabbitTemplate.convertAndSend(queueName, message);

    }
}
```

```
import com.alibaba.fastjson2.JSONObject;
import com.rabbitmq.client.Channel;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.support.AmqpHeaders;
import org.springframework.messaging.handler.annotation.Headers;
import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets;
import java.util.Map;

@Component
public class Consumer1 {
    private final String FANOUT_EMAIL_QUEUE = "fanout_email_queue";

    @RabbitHandler
    @RabbitListener(queuesToDeclare = @Queue(FANOUT_EMAIL_QUEUE))
    public void process(Message message, @Headers Map<String, Object> headers, Channel channel) throws Exception
{
        // 获取消息Id
        String messageId = message.getMessageProperties().getMessageId();
        String msg = new String(message.getBody(), StandardCharsets.UTF_8);
        System.out.println("邮件消费者获取生产者消息" + "messageId:" + messageId + ",消息内容:" + msg);
        JSONObject jsonObject = JSONObject.parseObject(msg);
        // 获取email参数
        String email = jsonObject.getString("email");

        // 请求地址,发送邮件
        // String emailUrl = "http://127.0.0.1:8083/sendEmail?email=" + email;
        // JSONObject result = HttpClientUtils.httpGet(emailUrl);
        // if (result == null) {
        //     // 因为网络原因,造成无法访问,继续重试
        //     throw new Exception("调用接口失败!");
        // }

        // 手动签收
        // 参数二：是否签收deliverTag之外所有小于deliverTag的消息
        channel.basicAck((Long) headers.get(AmqpHeaders.DELIVERY_TAG), false);
        System.out.println("执行结束....");
    }
}
```

```
import com.xii.mp.mq.Producer;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.concurrent.TimeUnit;

@SpringBootTest
public class AppTest {
    private final String FANOUT_EMAIL_QUEUE = "fanout_email_queue";

    @Autowired
    private Producer producer;

    @Test
    public void test01() {
        for (int i = 0; i < 2; i++) {
            producer.produce(FANOUT_EMAIL_QUEUE);
            producer.produce(FANOUT_EMAIL_QUEUE);
            System.out.println("=".repeat(66)+i);
        }


        try { TimeUnit.SECONDS.sleep(1000); } catch (InterruptedException e) { throw new RuntimeException(e); }
    }
}
```

五、发布订阅模型

> 一个消息被多个消费者消费
>
> - 推荐阅读：https://cloud.tencent.com/developer/article/2051103

1. 创建两个Queue绑定到一个Exchange中

```
<!-- 消息队列+fastjson2 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
    <version>2.7.7</version>
</dependency>
<dependency>
    <groupId>com.alibaba.fastjson2</groupId>
    <artifactId>fastjson2</artifactId>
    <version>2.0.21</version>
</dependency>

<!-- lombok -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.24</version>
</dependency>
```

```
spring:
  # datasource:
  #    type: com.alibaba.druid.pool.DruidDataSource
  #    driver-class-name: com.mysql.cj.jdbc.Driver
  #    url: jdbc:mysql://192.168.204.112:23306/dev_test?
autoReconnect=true&useServerPreparedStmts=true&cachePrepStmts=true&rewriteBatchedStatements=true&allowMultiQu
```

```
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.FanoutExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;

//Fanout 类型 发布订阅模式
@Component
public class FanoutConfig {

    // 邮件队列
    private final String FANOUT_EMAIL_QUEUE = "fanout_email_queue";

    // 短信队列
    private final String FANOUT_SMS_QUEUE = "fanout_sms_queue";
    // fanout 交换机
    private final String EXCHANGE_NAME = "fanoutExchange";

    // 1.定义邮件队列
    @Bean
    public Queue fanOutEamilQueue() {
        return new Queue(FANOUT_EMAIL_QUEUE);
    }

    // 2.定义短信队列
    @Bean
    public Queue fanOutSmsQueue() {
        return new Queue(FANOUT_SMS_QUEUE);
    }

    // 2.定义交换机
    @Bean
    FanoutExchange fanoutExchange() {
        return new FanoutExchange(EXCHANGE_NAME);
    }

    // 3.队列与交换机绑定邮件队列
    @Bean
    Binding bindingExchangeEmail(Queue fanOutEamilQueue, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(fanOutEamilQueue).to(fanoutExchange);
    }

    // 4.队列与交换机绑定短信队列
    @Bean
    Binding bindingExchangeSms(Queue fanOutSmsQueue, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(fanOutSmsQueue).to(fanoutExchange);
    }
}
```

2. 生产者

```
import com.alibaba.fastjson2.JSONObject;
import lombok.RequiredArgsConstructor;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageBuilder;
import org.springframework.amqp.core.MessageProperties;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Component;

import java.util.UUID;

@Component
@RequiredArgsConstructor
public class Producer {
    private final RabbitTemplate rabbitTemplate;

    public void produce(String queueName) {
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("email", "22222@qq.com");
        jsonObject.put("phoneNumber", "12222222222");
        jsonObject.put("timestamp", System.currentTimeMillis());
        String jsonString = jsonObject.toJSONString();
        System.out.println("jsonString:" + jsonString);
        // 设置消息唯一id 保证每次重试消息id唯一
        Message message = MessageBuilder.withBody(jsonString.getBytes())
                .setContentType(MessageProperties.CONTENT_TYPE_JSON).setContentEncoding("utf-8")
                .setMessageId(UUID.randomUUID() + "").build();
//        rabbitTemplate.convertAndSend(queueName, message);
        rabbitTemplate.convertAndSend(queueName, "", message);

    }
}
```

3. 邮件消费者+短信消费者

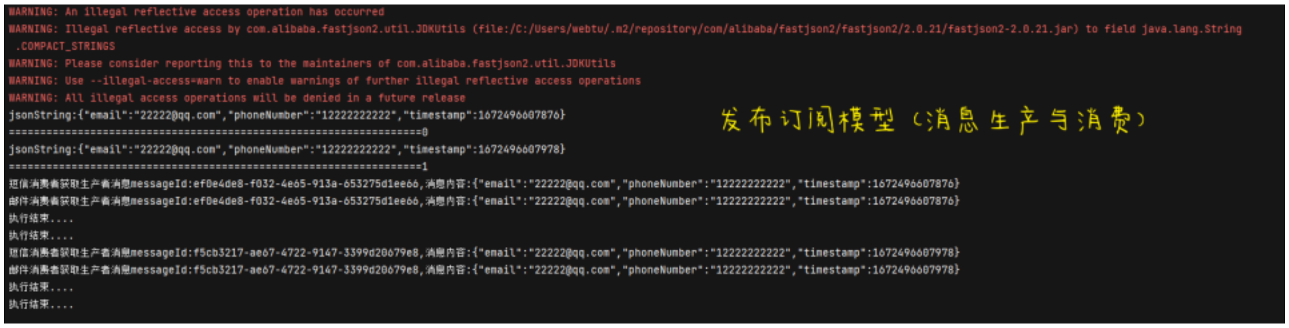
```
import com.alibaba.fastjson2.JSONObject;
import com.rabbitmq.client.Channel;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.support.AmqpHeaders;
import org.springframework.messaging.handler.annotation.Headers;
import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets;
import java.util.Map;

@Component
public class Consumer1 {
    private final String FANOUT_EMAIL_QUEUE = "fanout_email_queue";
    @RabbitHandler
    @RabbitListener(queuesToDeclare = @Queue(FANOUT_EMAIL_QUEUE))
    public void process(Message message, @Headers Map<String, Object> headers, Channel channel) throws Exception
{
        System.out.println("Consumer1.process");
        System.out.println(channel);
        System.out.println(channel.getConnection());
        System.out.println(headers);

        // 获取消息Id
        String messageId = message.getMessageProperties().getMessageId();
        String msg = new String(message.getBody(), StandardCharsets.UTF_8);
        System.out.println("邮件消费者获取生产者消息" + "messageId:" + messageId + ",消息内容:" + msg);
        JSONObject jsonObject = JSONObject.parseObject(msg);
        // 获取email参数
        String email = jsonObject.getString("email");

        // 请求地址,发送邮件
        // String emailUrl = "http://127.0.0.1:8083/sendEmail?email=" + email;
        // JSONObject result = HttpClientUtils.httpGet(emailUrl);
        // if (result == null) {
        //     // 因为网络原因,造成无法访问,继续重试
        //     throw new Exception("调用接口失败!");
        // }

        // 手动签收
        channel.basicAck((Long) headers.get(AmqpHeaders.DELIVERY_TAG), false);
        System.out.println("执行结束....");
    }
}
```

```
import com.alibaba.fastjson2.JSONObject;
import com.rabbitmq.client.Channel;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.support.AmqpHeaders;
import org.springframework.messaging.handler.annotation.Headers;
import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets;
import java.util.Map;

@Component
public class Consumer {
    private final String FANOUT_SMS_QUEUE = "fanout_sms_queue";

    @RabbitHandler
    @RabbitListener(queuesToDeclare = @Queue(FANOUT_SMS_QUEUE))
    public void process(Message message, @Headers Map<String, Object> headers, Channel channel) throws Exception
{
        System.out.println("Consumer.process");
        System.out.println(channel);
        System.out.println(channel.getConnection());
        System.out.println(headers);
        // 获取消息Id
        String messageId = message.getMessageProperties().getMessageId();
        String msg = new String(message.getBody(), StandardCharsets.UTF_8);
        System.out.println("短信消费者获取生产者消息" + "messageId:" + messageId + ",消息内容:" + msg);
        JSONObject jsonObject = JSONObject.parseObject(msg);
        // 获取phoneNumber参数
        String phoneNumber = jsonObject.getString("phoneNumber");

        // 请求地址,发送短信
        // String smsUrl = "http://127.0.0.1:8083/sendSMS?phoneNumber=" + phoneNumber;
        // JSONObject result = HttpClientUtils.httpGet(smsUrl);
        // if (result == null) {
        //     // 因为网络原因,造成无法访问,继续重试
        //     throw new Exception("调用接口失败!");
        // }

        // 手动签收
        channel.basicAck((Long) headers.get(AmqpHeaders.DELIVERY_TAG), false);
        System.out.println("执行结束....");
    }
}
```

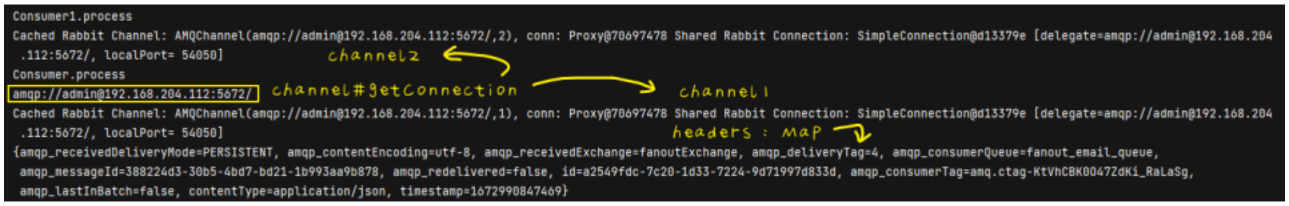
4. 测试生产数据（生产两条数据）

```
import com.xii.mp.mq.Producer;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.concurrent.TimeUnit;

@SpringBootTest
public class AppTest {
    // 邮件队列
    private final String FANOUT_EMAIL_QUEUE = "fanout_email_queue";

    // 短信队列
    private final String FANOUT_SMS_QUEUE = "fanout_sms_queue";
    // fanout 交换机
    private final String EXCHANGE_NAME = "fanoutExchange";

    @Autowired
    private Producer producer;

    @Test
    public void test01() {
        for (int i = 0; i < 2; i++) {
            producer.produce(EXCHANGE_NAME);
            System.out.println("=".repeat(66)+i);
        }



        try { TimeUnit.SECONDS.sleep(1000); } catch (InterruptedException e) { throw new RuntimeException(e); }
    }
}
```

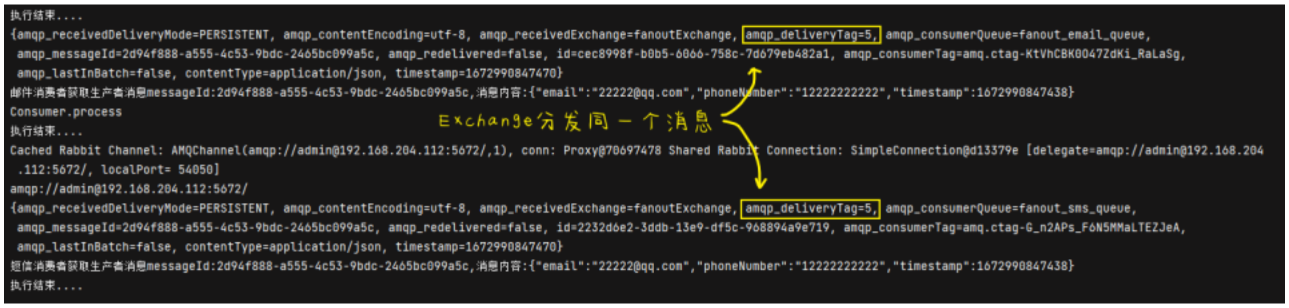
5. 测试结果



```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.alibaba.fastjson2.util.JDKUtils (file:/C:/Users/webtu/.m2/repository/com/alibaba/fastjson2/fastjson2/2.0.21/fastjson2-2.0.21.jar) to field java.lang.String
 .COMPACT_STRINGS
WARNING: Please consider reporting this to the maintainers of com.alibaba.fastjson2.util.JDKUtils
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
jsonString:{"email":"22222@qq.com","phoneNumber":"12222222222","timestamp":1672496607876}
================================================================0
jsonString:{"email":"22222@qq.com","phoneNumber":"12222222222","timestamp":1672496607978}
================================================================1
短信消费者获取生产者消息messageId:ef0e4de8-f032-4e65-913a-653275d1ee66,消息内容:{"email":"22222@qq.com","phoneNumber":"12222222222","timestamp":1672496607876}
邮件消费者获取生产者消息messageId:ef0e4de8-f032-4e65-913a-653275d1ee66,消息内容:{"email":"22222@qq.com","phoneNumber":"12222222222","timestamp":1672496607876}
执行结束....
执行结束....
短信消费者获取生产者消息messageId:f5cb3217-ae67-4722-9147-3399d20679e8,消息内容:{"email":"22222@qq.com","phoneNumber":"12222222222","timestamp":1672496607978}
邮件消费者获取生产者消息messageId:f5cb3217-ae67-4722-9147-3399d20679e8,消息内容:{"email":"22222@qq.com","phoneNumber":"12222222222","timestamp":1672496607978}
执行结束....
执行结束....
```



```
Consumer1.process
Cached Rabbit Channel: AMQChannel(amqp://admin@192.168.204.112:5672/,2), conn: Proxy@70697478 Shared Rabbit Connection: SimpleConnection@d13379e [delegate=amqp://admin@192.168.204
 .112:5672/, localPort= 54050]
Consumer.process
amqp://admin@192.168.204.112:5672/
Cached Rabbit Channel: AMQChannel(amqp://admin@192.168.204.112:5672/,1), conn: Proxy@70697478 Shared Rabbit Connection: SimpleConnection@d13379e [delegate=amqp://admin@192.168.204
 .112:5672/, localPort= 54050]
{amqp_receivedDeliveryMode=PERSISTENT, amqp_contentEncoding=utf-8, amqp_receivedExchange=fanoutExchange, amqp_deliveryTag=4, amqp_consumerQueue=fanout_email_queue,
 amqp_messageId=388224d3-30b5-4bd7-bd21-1b993aa9b878, amqp_redelivered=false, id=a2549fdc-7c20-1d33-7224-9d71997d833d, amqp_consumerTag=amq.ctag-KtVhCBK0047ZdKi_RaLaSg,
 amqp_lastInBatch=false, contentType=application/json, timestamp=1672990847469}
```



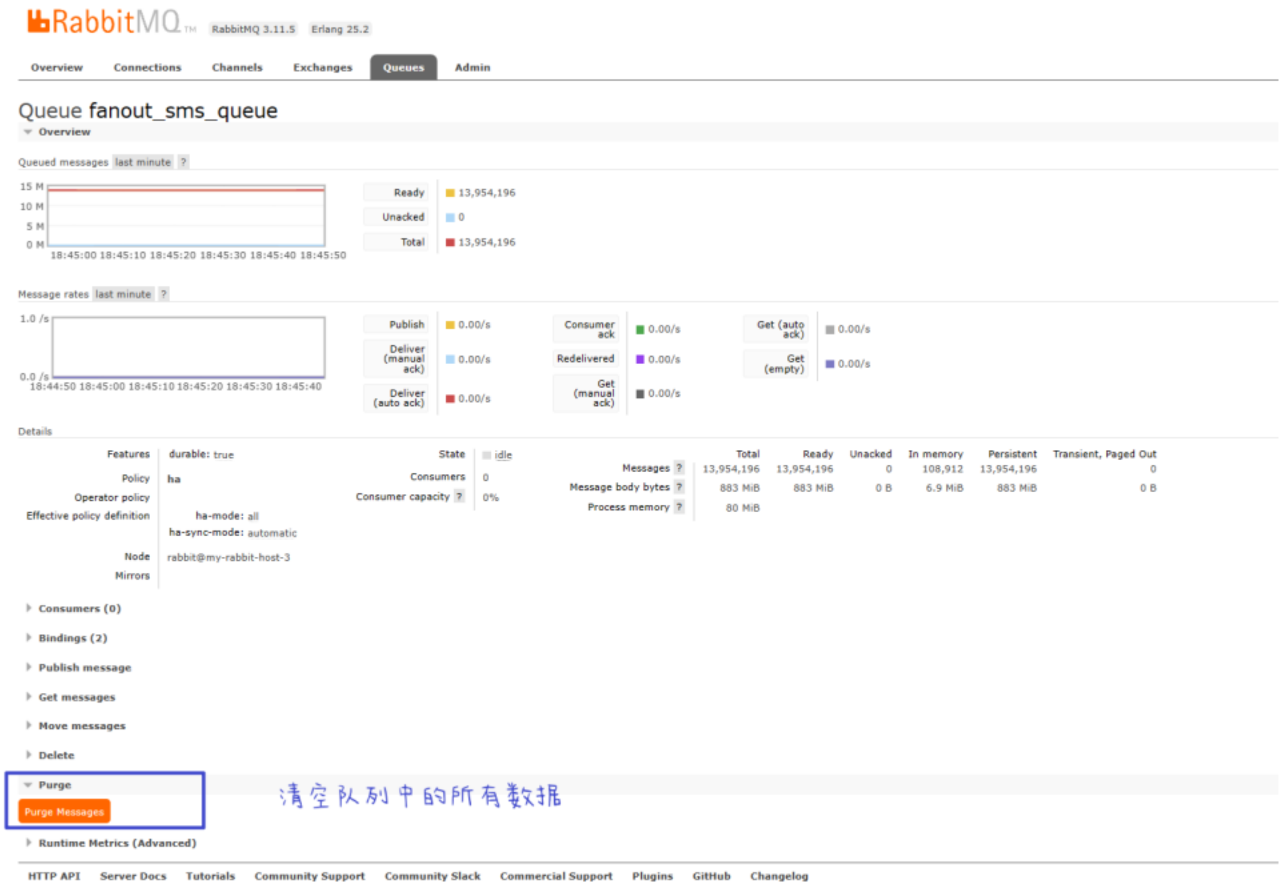
```
执行结束....
{amqp_receivedDeliveryMode=PERSISTENT, amqp_contentEncoding=utf-8, amqp_receivedExchange=fanoutExchange, amqp_deliveryTag=5, amqp_consumerQueue=fanout_email_queue,
 amqp_messageId=2d94f888-a555-4c53-9bdc-2465bc099a5c, amqp_redelivered=false, id=cec8998f-b0b5-6066-758c-7d679eb482a1, amqp_consumerTag=amq.ctag-KtVhCBK0047ZdKi_RaLaSg,
 amqp_lastInBatch=false, contentType=application/json, timestamp=1672990847470}
邮件消费者获取生产者消息messageId:2d94f888-a555-4c53-9bdc-2465bc099a5c,消息内容:{"email":"22222@qq.com","phoneNumber":"12222222222","timestamp":1672990847438}
Consumer.process
执行结束....
Cached Rabbit Channel: AMQChannel(amqp://admin@192.168.204.112:5672/,1), conn: Proxy@70697478 Shared Rabbit Connection: SimpleConnection@d13379e [delegate=amqp://admin@192.168.204
 .112:5672/, localPort= 54050]
amqp://admin@192.168.204.112:5672/
{amqp_receivedDeliveryMode=PERSISTENT, amqp_contentEncoding=utf-8, amqp_receivedExchange=fanoutExchange, amqp_deliveryTag=5, amqp_consumerQueue=fanout_sms_queue,
 amqp_messageId=2d94f888-a555-4c53-9bdc-2465bc099a5c, amqp_redelivered=false, id=2232d6e2-3ddb-13e9-df5c-968894a9e719, amqp_consumerTag=amq.ctag-G_n2APs_F6N5MMaLTEZJeA,
 amqp_lastInBatch=false, contentType=application/json, timestamp=1672990847470}
短信消费者获取生产者消息messageId:2d94f888-a555-4c53-9bdc-2465bc099a5c,消息内容:{"email":"22222@qq.com","phoneNumber":"12222222222","timestamp":1672990847438}
执行结束....
```

六、清空队列中的数据

1. 方式一

> rabbitmqctl –q purge_queue queue_name

2. 方式二



Queue fanout_sms_queue

Overview

Queued messages last minute ?

Ready	🟨 13,954,196
Unacked	🟦 0
Total	🟥 13,954,196

Message rates last minute ?

Publish	🟨 0.00/s	Consumer ack	🟩 0.00/s	Get (auto ack)	⬜ 0.00/s
Deliver (manual ack)	🟦 0.00/s	Redelivered	🟪 0.00/s	Get (empty)	🟪 0.00/s
Deliver (auto ack)	🟥 0.00/s	Get (manual ack)	⬛ 0.00/s		

Details

Features	durable: true
Policy	ha
Operator policy	
Effective policy definition	ha-mode: all ha-sync-mode: automatic
Node	rabbit@my-rabbit-host-3
Mirrors	

State	idle
Consumers	0
Consumer capacity ?	0%

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	13,954,196	13,954,196	0	108,912	13,954,196	0
Message body bytes ?	883 MiB	883 MiB	0 B	6.9 MiB	883 MiB	0 B
Process memory ?	80 MiB					

清空队列中的所有数据

七、多记录拼接到一个Message中

- MQ在海量数据导入数据库中的集成方案

> 4核4G RabbitMQ
>
> - Max写入：10000条/s
> - Max读取：5000条/s
>
> 我们可以尽可能增加每一个Message携带的信息量，避免使得读取/写入上限速率成为瓶颈因素
>
> - 将10~50条数据记录到一个Message中

1. 生产者

```
import com.alibaba.fastjson2.JSONObject;
import com.xii.mp.domain.entity.User;
import lombok.RequiredArgsConstructor;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageBuilder;
import org.springframework.amqp.core.MessageProperties;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import java.util.concurrent.ThreadLocalRandom;

@Component
@RequiredArgsConstructor
public class Producer {
    private final RabbitTemplate rabbitTemplate;

    public void produce(String queueName) {

        List<User> users = new ArrayList<>();
        for (int i = 0; i < 10_0000; i++) {
            users.clear();
            for (int ii = 0; ii < 60; ii++) {
                int age = ThreadLocalRandom.current().nextInt(1, 130);
                users.add(new User(UUID.randomUUID().toString().replaceAll("-", ""), "AAA-" + age, age));
            }

            // 设置消息唯一id 保证每次重试消息id唯一
            Message message = MessageBuilder.withBody(JSONObject.toJSONString(users).getBytes())
                    .setContentType(MessageProperties.CONTENT_TYPE_JSON).setContentEncoding("utf-8")
                    .setMessageId(UUID.randomUUID() + "").build();
            rabbitTemplate.convertAndSend(queueName, message);
//            users.forEach(user -> {
//
//                // 设置消息唯一id 保证每次重试消息id唯一
//                Message message = MessageBuilder.withBody(JSONObject.toJSONString(user).getBytes())
//                        .setContentType(MessageProperties.CONTENT_TYPE_JSON).setContentEncoding("utf-8")
//                        .setMessageId(UUID.randomUUID() + "").build();
//                rabbitTemplate.convertAndSend(queueName, message);
//            });
        }
        rabbitTemplate.convertAndSend(queueName, "@@END@@");
    }
}
```

2. 消费者

- 批量读取MQ数据
- 解析批量Message中每个Message中的记录数据
- 结束标记解析
- 事务的开始与提交/回滚时间点

```
import com.alibaba.fastjson2.JSONObject;
import com.rabbitmq.client.Channel;
import com.xii.mp.domain.entity.User;
import com.xii.mp.service.UserService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.rabbit.listener.api.ChannelAwareMessageListener;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@Component
@Slf4j
public class Consumer implements ChannelAwareMessageListener {

    private final String FANOUT_SMS_QUEUE = "fanout_sms_queue";
    @Autowired
    UserService userService;
    int transactionStart = 0;
    List<Boolean> rightTrans = new ArrayList<>();
    int endTag = 0;
    @Autowired
    private PlatformTransactionManager manager;
    TransactionDefinition definition = null;
    TransactionStatus insertTrans = null;

    @Override
    public void onMessage(Message message, Channel channel) throws Exception {

    }

    @RabbitListener(queues = FANOUT_SMS_QUEUE, containerFactory = "mqConsumerlistenerContainer")
    @Override
    public void onMessageBatch(List<Message> messages, Channel channel) {
        if (transactionStart == 0) {
            String now1 = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss").format(LocalDateTime.now());
            System.out.println("now1 = " + now1);
            // 开启事务
            definition = new DefaultTransactionDefinition();
            insertTrans = manager.getTransaction(definition);
        }
        // 获得消息编号，用于手动ack
        transactionStart += messages.size();

        // 判断当前批次消息是否包含结束信息
        // 注意：请确保一次将抽取的数据全部消费完毕，否则结束标记可能被下一批次获取从而导致解析异常
        if ("@@END@@".equals(new String(messages.get(messages.size()-1).getBody(), StandardCharsets.UTF_8))) {
            endTag = 1;
            messages.remove(messages.size() - 1);
        }
        List<User> users = new ArrayList<>();
        messages.stream().map(x -> JSONObject.parseObject(new String(x.getBody(), StandardCharsets.UTF_8), List.
class))
                .collect(Collectors.toList()).forEach(users::addAll);

        // 判断事务是否执行成功,users.size()==0表示最后一批只有一条结束数据
        if(users.size()!=0){
            try {
                // 模拟异常：userService.saveBatch(users) && userService.saveBatch(users)
                if (userService.saveBatch(users)) {
                    rightTrans.add(true);
                    try {
                        channel.basicAck(transactionStart, true);
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                } else {
                    rightTrans.add(false);
                    try {
                        channel.basicAck(messages.size(), true);
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                }
            } catch (RuntimeException e) {
                rightTrans.add(false);
                log.error(e.getMessage());
                try {
                    channel.basicAck(transactionStart, true);
                } catch (IOException ee) {
                    throw new RuntimeException(e);
                }
            }
        }else{
            try {
                channel.basicAck(messages.size(), true);
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }

        // 消费完毕，如果存在事务异常则rollback，反之commit
        if (rightTrans.contains(false) && endTag == 1) {
            System.out.println("ROLLBACK");
            manager.rollback(insertTrans);
        }
        if (!rightTrans.contains(false) && endTag == 1) {
            System.out.println("COMMIT");
            manager.commit(insertTrans);
        }

        String now2 = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss").format(LocalDateTime.now());
        System.out.println("now2 = " + now2);
        System.out.println("消费者执行结束....: " + transactionStart);
    }
}
```

3. 测试程序

```
import com.xii.mp.mq.Producer;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.concurrent.TimeUnit;
```

```
@SpringBootTest
public class AppTest1 {

    private final String EXCHANGE_NAME = ”fanoutExchange”;

    // 短信队列
    private final String FANOUT_SMS_QUEUE = ”fanout_sms_queue”;

    @Autowired
    private Producer producer;


    @Test
    public void test01() {
        producer.produce(FANOUT_SMS_QUEUE);

        try {
            TimeUnit.SECONDS.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

八、编程式消费队列中数据

1. 生产者

```
import com.alibaba.fastjson2.JSONObject;
import com.xii.mp.domain.entity.User;
import lombok.RequiredArgsConstructor;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageBuilder;
import org.springframework.amqp.core.MessageProperties;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import java.util.concurrent.ThreadLocalRandom;
import java.util.concurrent.TimeUnit;

@Component
@RequiredArgsConstructor
public class Producer {
    private final RabbitTemplate rabbitTemplate;

    public void produce(String queueName) {

        List<User> users = new ArrayList<>();
        for (int i = 0; i < 15; i++) {
            users.clear();
            for (int ii = 0; ii < 6; ii++) {
                int age = ThreadLocalRandom.current().nextInt(1, 130);
                users.add(new User(UUID.randomUUID().toString().replaceAll("-", ""), "AAA-" + age, age));
            }

            // 设置消息唯一id 保证每次重试消息id唯一
            Message message = MessageBuilder.withBody(JSONObject.toJSONString(users).getBytes())
                    .setContentType(MessageProperties.CONTENT_TYPE_JSON).setContentEncoding("utf-8")
                    .setMessageId(UUID.randomUUID() + "").build();
            rabbitTemplate.convertAndSend(queueName, message);
            try {
                TimeUnit.MICROSECONDS.sleep(200);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }

//            users.forEach(user -> {
//
//                // 设置消息唯一id 保证每次重试消息id唯一
//                Message message = MessageBuilder.withBody(JSONObject.toJSONString(user).getBytes())
//                        .setContentType(MessageProperties.CONTENT_TYPE_JSON).setContentEncoding("utf-8")
//                        .setMessageId(UUID.randomUUID() + "").build();
//                rabbitTemplate.convertAndSend(queueName, message);
//            });
        }
        // rabbitTemplate.convertAndSend(queueName, "@@END@@");
    }
}
```

2. 消费者

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.GetResponse;
import com.xii.mp.mq.Producer;
import org.junit.jupiter.api.Test;
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;
import org.springframework.amqp.rabbit.connection.Connection;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;

@SpringBootTest
public class AppTest1 {

    private final String EXCHANGE_NAME = "fanoutExchange";

    // 短信队列
    private final String FANOUT_SMS_QUEUE = "fanout_sms_queue";
    int count = 0;
    @Autowired
    private Producer producer;
    @Autowired
    private CachingConnectionFactory connectionFactory;
    @Test
    public void test01() {
        new Thread(() -> {
            producer.produce(FANOUT_SMS_QUEUE);
        }, "th1").start();
        System.out.println("----------------".repeat(5));


        List<String> batchData = new ArrayList();

        try (
                Connection conn = connectionFactory.createConnection();
                Channel channel = conn.createChannel(false);
        ) {
            long time1 = System.currentTimeMillis();
            while (true) {
                long time2 = System.currentTimeMillis();
                GetResponse getResponse = channel.basicGet(FANOUT_SMS_QUEUE, false);
                if (getResponse != null) {
                    time1 = System.currentTimeMillis();
                    batchData.add(new String(getResponse.getBody(), StandardCharsets.UTF_8));
                    // long deliveryTag = getResponse.getEnvelope().getDeliveryTag();
                    // channel.basicAck(deliveryTag, false);
                }
                if (batchData.size() == 10) {
                    System.out.println("消费batchData: " + batchData.size());
                    batchData.clear();
                    channel.basicAck(count, true);
                }
                // 10s队列中没有新增数据，消费batchData中的已有数据
                if (time2 - time1 > 10_000) {
                    if (batchData.size() > 0) {
                        System.out.println("消费batchData: " + batchData.size());
                        batchData.clear();
                        channel.basicAck(count, true);
                    }
                    break;
                }
            }
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }


        System.out.println("===========".repeat(5));
    }
}
```