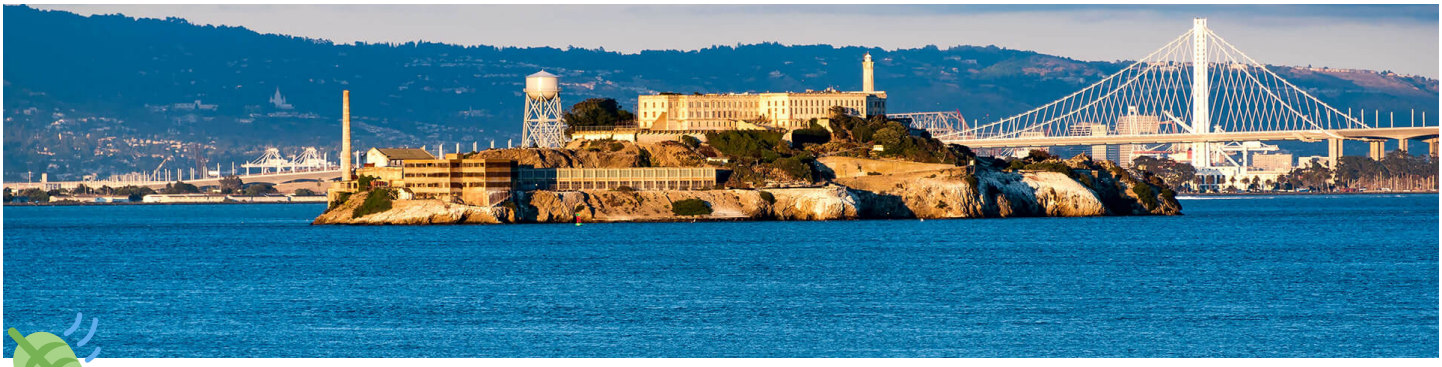


消息队列 | RabbitMQ

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。在分布式系统中，可以通过它来进行不同服务之间的消息传递。

用途

应用解耦：订单系统下单后，将订单信息存在 mq 中，库存系统再从 mq 消费订单信息，可避免库存系统宕机导致订单丢失。

异步消费：订单系统下单后，将订单信息存在 mq 中，此时可以立即返回下单成功，然后再完成下单的剩余操作。

流量削峰：海量订单到来后，将订单信息存在 mq 中，（MQ 可以承受海量并发，数据库不能承受），此时可以缓慢消费 mq 的消息来减少数据库的并发。

消息可靠性

消息到 mq：使用事务机制或 Confirm 机制。

RabbitMQ 自身：持久化、集群、普通模式、镜像模式。

从 mq 消费：basicAck 机制、死信队列、消息补偿机制。

队列类型

死信队列：一个专门用来存放被拒消息、超时消息、队列溢出的消息的队列。

延迟队列：队列中的消息不会立即让消费者进行消息，而是等待一段时间。

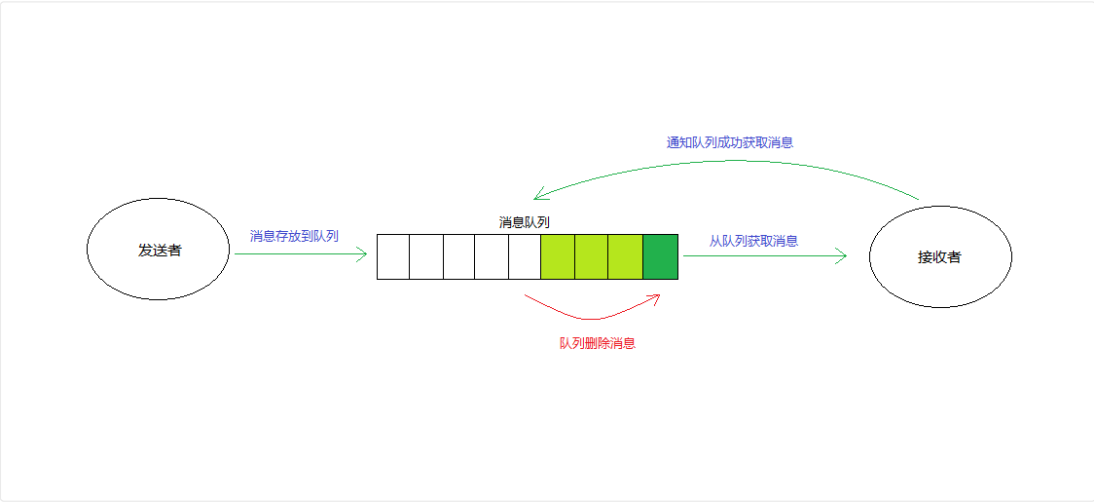
优先级队列：优先级高的队列会先被消费。消费速度>生产速度且 Broker 没有堆积的情况下，优先级失去意义。

理论知识

使用 RabbitMQ 需要下载服务并在机器上运行，启动后可进入控制台：http://127.0.0.1:15672 用户密码均为：guest

点对点模式

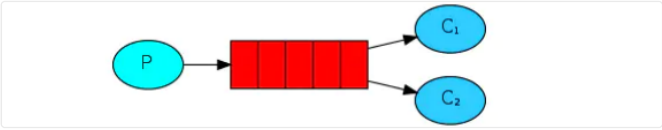
RabbitMQ 发送多条消息时，会将消息放在队列中，接收方从该队列获取消息并通知队列自己已经接收（发送 ack），随后队列将该消息删除。



接收者通知队列自身接收成功，可以避免接收者获取消息时宕机，而队列删除消息从而导致消息无法被“消费”的情况出现。

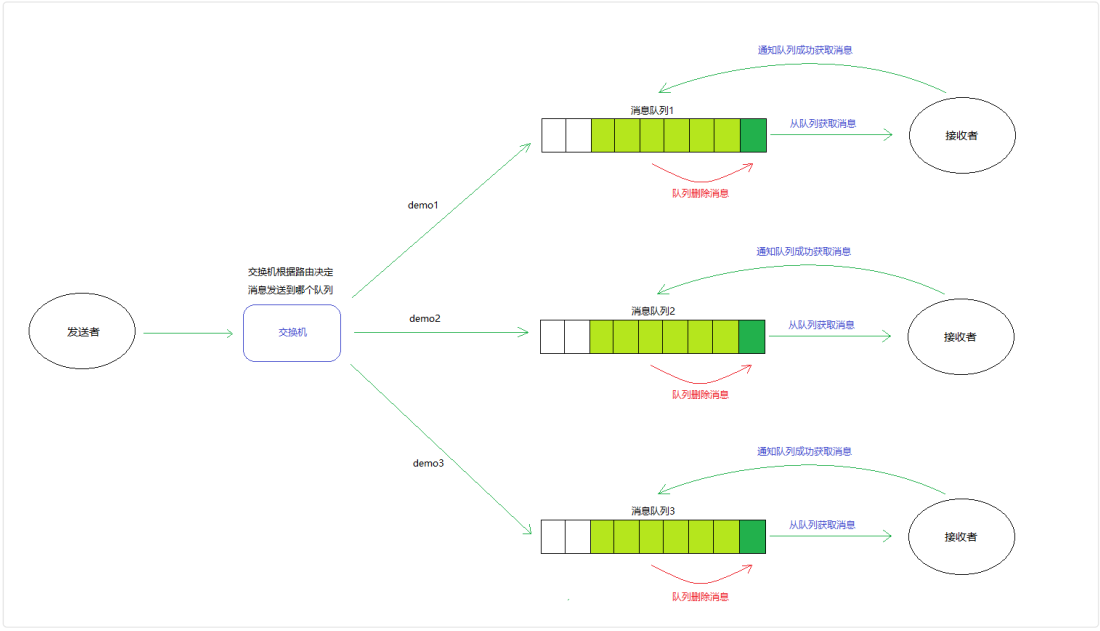
工作模式

多个消费者共同消费一个队列，先拿到消息的消费者进行消费，会产生竞争。并非情况下可能会产生多个消费者消费了同一个消息的情况。



发布-订阅者模式

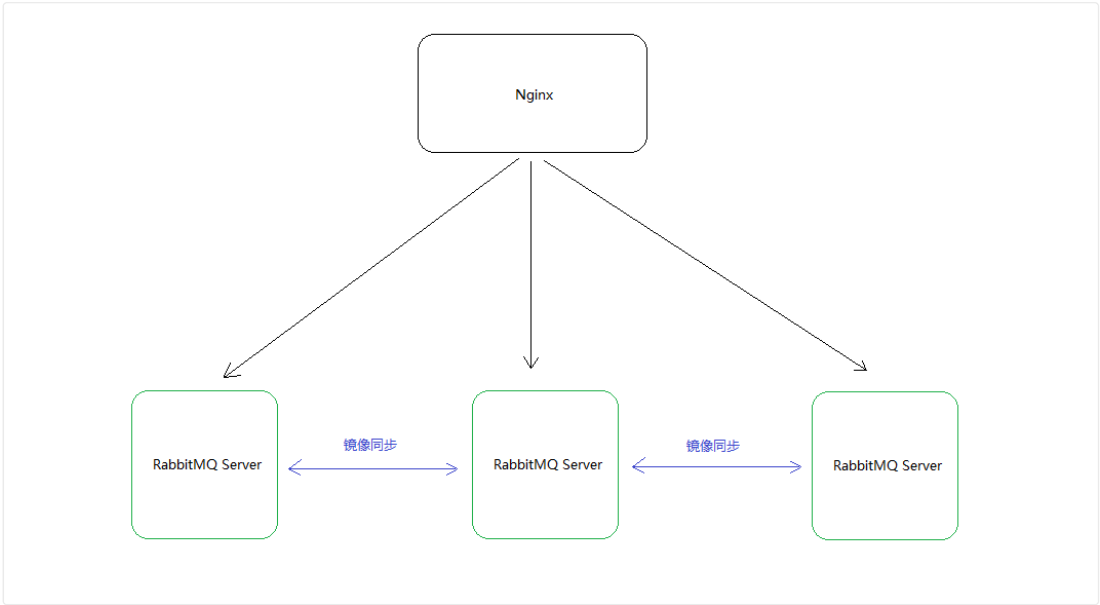
即将一个消息同时发送给多个接收者。也可以利用消息分组来将消息发送给部分服务（下图中，并非表示一个接收者一个队列，每个接收者都可以接收多个队列的消息）。



交换机根据路由（demo1、demo2、demo3）来决定将消息发送给哪个队列，除此之外，可以根据表达式进行路由匹配（主题模式）。

RabbitMQ 高可用集群

单台 RabbitMQ 服务到达瓶颈后，需要开启多个 RabbitMQ server 来分摊压力。RabbitMQ 本身并不支持负载均衡，不过可以通过 nginx 来实现。



RabbitMQ 集群各节点之间共享队列，但各队列之间默认数据不同步，仅收到消息的队列有消息，所以需要开启镜像模式，才能同步并实现高可用。

代码实现

使用 RabbitMQ 需要先搭建并启动服务。

消息发送

引入依赖

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

配置：

```yaml
spring:
 rabbitmq:
   host: localhost
   port: 5672
   username: rabbit # 账号rabbit是在控制台新加的，默认账号不支持远程连接
   password: rabbit
   virtual-host: myhost  #虚拟机名，如果没添加，直接用 / 即可
   listener:
    simple:
      acknowledge-mode: manual  # 设置手动回复ack
```

配置交换机：

```java
@Configuration
public class RabbitConfig {

    /**
     * new Queue():
     *  参数一：队列名称
     *  参数二：是否持久化队列，默认true
     *  参数三：是否为当前连接专属，连接关闭后销毁，默认false
     *  参数四：是否自动删除，无服务使用时自动删除，默认false
     *
     */
    @Bean // 队列
    public Queue theQueue(){
        return new Queue("theQueue",true);
    }

    @Bean // 交换机
    public DirectExchange theDirectExchange(){
        return new DirectExchange("theDirectExchange",true,false);//名字、持久化、自动删除
    }

    @Bean // 绑定队列、交换机，匹配值为key
    public Binding bindingDirect() { //监听key的服务会收到消息
        return BindingBuilder.bind(theQueue()).to(theDirectExchange()).with("key");
    }

    @Bean // 绑定队列、交换机，匹配值为key开头
    public Binding bindingDirectAll() { //监听Key开头的服务会收到消息
        return BindingBuilder.bind(theQueue()).to(theDirectExchange()).with("key.#");
    }
}
```

消息发送：向key匹配的队列发送消息

```java
@Service
public class RabbitmqServiceImpl {

  @Autowired
  private RabbitTemplate rabbitTemplate;

  @Resource
  private RabbitmqCallback rabbitmqCallback;

  // 不含回调
  public void sendMessage(){ // 交换机、绑定键、消息
     rabbitTemplate.convertAndSend("theDirectExchange","key","hello world!");
  }

  // 含回调
```

```
    @PostMapping("/send2")
    public void sendMessage(String exchange, String key, String message, CorrelationData correlationData) {
        rabbitTemplate.setConfirmCallback(rabbitmqCallback);//设置回调
        rabbitTemplate.convertAndSend(exchange,key,message,correlationData);
    }

}
```

消息发送回调

Java

```
@Slf4j
@Component
public class RabbitmqCallback implements RabbitTemplate.ConfirmCallback {


    @Autowired
    private MessageRecordService messageRecordService;

    @Autowired
    private RabbitmqService rabbitmqService;

    /**
     * 消息发送回调，进行消息确认
     * @param correlationData 相关信息
     * @param isAck 是否收到ack
     * @param errorMessage 错误信息
     */
    @Override
    public void confirm(CorrelationData correlationData, boolean isAck, String errorMessage) {

        assert correlationData != null;
        String messageId=correlationData.getId();
        MessageRecord messageRecord = messageRecordService.findMessageRecord(messageId);
        log.info("消息发送状态: {}",isAck);
        //未收到ack，进行重发；收到时将消息状态改为发送成功
        if(!isAck){
            rabbitmqService.sendMessage(RabbitmqName.ORDER_EXCHANGE, RabbitmqName.ORDER_KEY, JSON.toJSONString(messageRecord),correlati
        }else{
            // messageRecordService.updateMessageSendSuccess(messageId);
        }

    }

}
```

消息接收

引入依赖：(和发送一样)

XML

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

配置：（和发送类似）

YAML

```
spring:
 rabbitmq:
   host: localhost
   port: 5673
   username: rabbit
   password: rabbit
   virtual-host: myhost  #虚拟机名，如果没添加，直接用 / 即可
```

监听队列+消息接收

Java

```
@Component
@RabbitListener(queues = "theQueue") // 监听的队列
public class RabbitListen {

```

```
    @RabbitHandler
    public void getMessage(String msg, Channel channel, Message message){
        System.out.println("消费者1收到消息：====="+msg);
        //查询数据库订单，如果订单不存在，进行补单
        sendAck(channel,message);
    }


    /*
      确认一条消息：发送ACK确认，通知队列删除消息
      channel.basicAck(deliveryTag, false);
      deliveryTag:该消息的index
      multiple：是否批量.true:将一次性ack所有小于deliveryTag的消息
     */
    public void sendAck(Channel channel, Message message){
        try {
            channel.basicNack(message.getMessageProperties().getDeliveryTag(),false,false);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}


```

服务搭建

Erlang

Erlang必须安装，并且版本要和RibbitMQ版本一致，可以参考: Erlang-ribbitmq版本对照 。

```Ada
1.下载地址Erlang,并进行安装

2.安装完成配置环境变量：path指向Erlang的bin目录。

3.cmd 输入 erl 出现版本信息，此时安装成功！
```

RabbitMQ

安装

```Ada
下载后点击安装，一直下一步即可。
```

管理server

安装后，开始菜单-最近添加中将出现几个选项

```Ada
RibbitMQ Service - install  # 安装

RibbitMQ Service - remove   # 移除

RibbitMQ Service - start    # 启动

RibbitMQ Service - stop     # 停止
```

UI管理工具

```Ada
1.RabbitMQ安装目录下sbin目录，进入cmd，执行命令：rabbitmq-plugins enable rabbitmq_management

2.双击：sbin / rabbitmq-server.bat

3.访问：http://127.0.0.1:15672 ，用户密码均为：guest
```