课程目标

1、 掌握 RabbitMQ 的可靠性消息的发送

2、 掌握 RabbitMQ 集群的原理与高可用架构的搭建

3、 学习 RabbitMQ 的实践经验

内容定位

掌握了 RabbitMQ 的基本使用之后，进一步学习如何搭建高可用的集群以及如何保证消息投递的可靠性，以及常见面试题的分析。

上节课的回顾

1、MQ 的本质，MQ 的作用

2、RabbitMQ 的特性，工作模型，交换机详解

3、Java API 编程，UI 管理界面

4、进阶知识：TTL、死信队列、延迟队列，服务端流控和消费端限流

5、Spring AMQP 核心组件

6、Spring Boot 集成 RabbitMQ，项目实战

1. RabbitMQ 可靠性投递与高可用架构

1.1. 可靠性投递

上节课我们说在代码里面一定是先操作数据库再发送消息。避免因为数据库回滚导致的数据不一致。但是如果先操作数据，后发送消息，发送消息出了问题，那不是一样会出现业务数据的不一致?

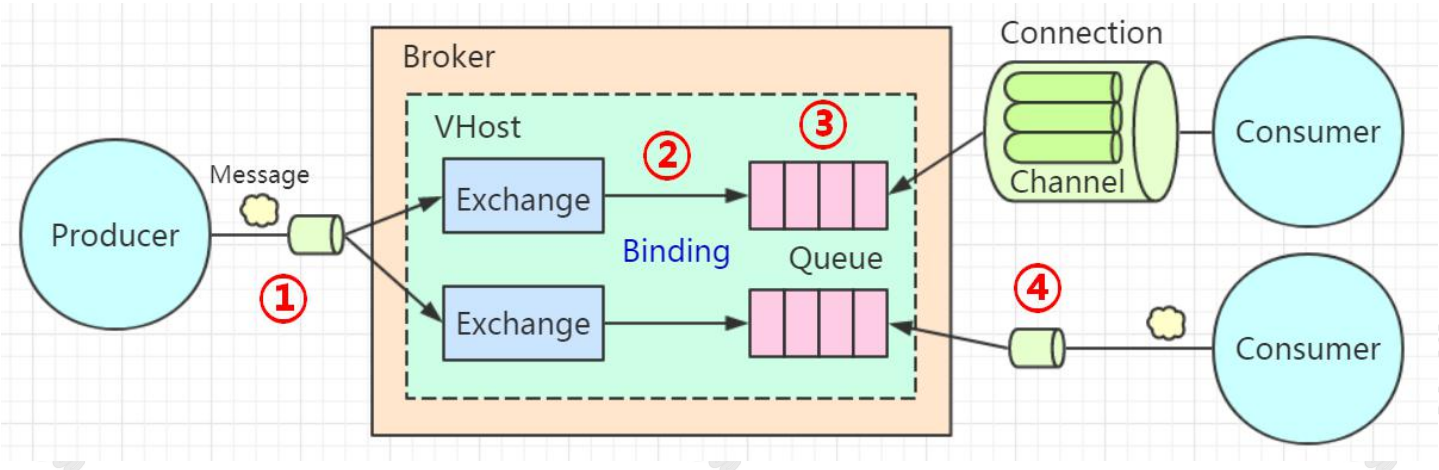
本节课我们来分析 RabbitMQ 的可靠性投递，也就是在使用 RabbitMQ 实现异步通信的时候，消息丢了怎么办，消息重复消费怎么办?

在 RabbitMQ 里面提供了很多保证消息可靠投递的机制，这个也是 RabbitMQ 的一个特性。

我们在讲可靠性投递的时候，必须要明确一个问题，因为效率与可靠性是无法兼得的，如果要保证每一个环节都成功，势必会对消息的收发效率造成影响。所以如果是一些业务实时一致性要求不是特别高的场合，可以牺牲一些可靠性来换取效率。

比如发送通知或者记录日志的这种场景，如果用户没有收到通知，不会造成业务影响，只要再次发送就可以了。

我们再来回顾一下上节课讲的 RabbitMQ 的工作模型。



在我们使用 RabbitMQ 收发消息的时候，有几个主要环节:

① 代表消息从生产者发送到 Broker

生产者把消息发到 Broker 之后，怎么知道自己的消息有没有被 Broker 成功接收?

② 代表消息从 Exchange 路由到 Queue

Exchange 是一个绑定列表，如果消息没有办法路由到正确的队列，会发生什么事情？应该怎么处理?

③ 代表消息在 Queue 中存储

队列是一个独立运行的服务，有自己的数据库（Mnesia），它是真正用来存储消息的。如果还没有消费者来消费，那么消息要一直存储在队列里面。如果队列出了问题，消息肯定会丢失。怎么保证消息在队列稳定地存储呢?

④ 代表消费者订阅 Queue 并消费消息

队列的特性是什么？FIFO。队列里面的消息是一条一条的投递的，也就是说，只有上一条消息被消费者接收以后，才能把这一条消息从数据库删掉，继续投递下一条消息。那么问题来了，Broker 怎么知道消费者已经接收了消息呢?

1.1.1. 消息发送到 RabbitMQ 服务器

第一个环节是生产者发送消息到 Broker。可能因为网络或者 Broker 的问题导致消息发送失败，生产者不能确定 Broker 有没有正确的接收。

在 RabbitMQ 里面提供了两种机制**服务端确认机制**，也就是在生产者发送消息给 RabbitMQ 的服务端的时候，服务端会通过某种方式返回一个应答，只要生产者收到了这个应答，就知道消息发送成功了。

第一种是 Transaction（事务）模式，第二种 Confirm（确认）模式。

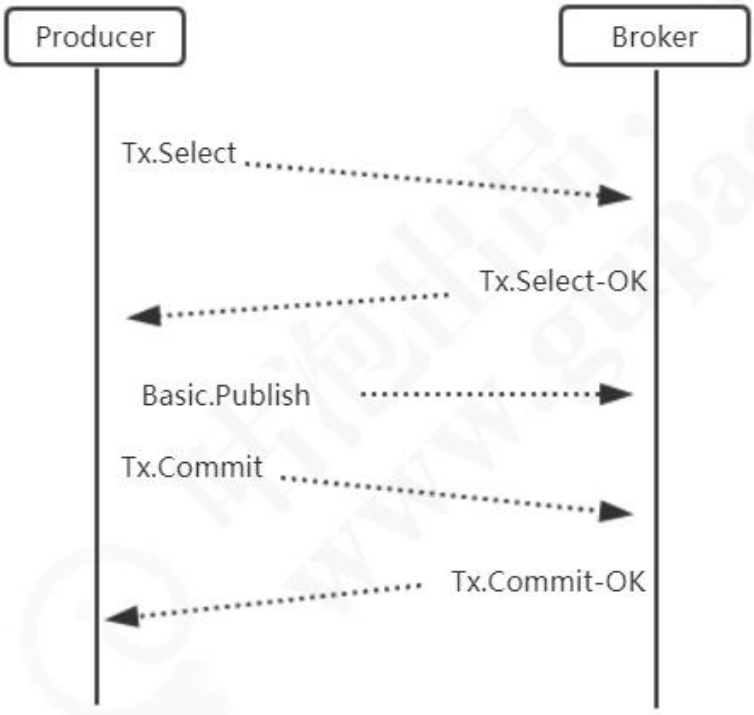
Transaction（事务）模式

事务模式怎么使用呢?

我们通过一个 channel.txSelect()的方法把信道设置成事务模式，然后就可以发布消息给 RabbitMQ 了，如果 channel.txCommit();的方法调用成功，就说明事务提交成功，则消息一定到达了 RabbitMQ 中。

如果在事务提交执行之前由于 RabbitMQ 异常崩溃或者其他原因抛出异常，这个时候我们便可以将其捕获，进而通过执行 channel.txRollback()方法来实现事务回滚。

代码：gupaoedu-vip-rabbitmq-javaapi：com.gupaoedu.transaction



AMQP 协议抓包示意

在事务模式里面，只有收到了服务端的 Commit-OK 的指令，才能提交成功。所以可以解决生产者和服务端确认的问题。但是事务模式有一个缺点，它是阻塞的，一条消息没有发送完毕，不能发送下一条消息，它会榨干 RabbitMQ 服务器的性能。所以不建议大家在生产环境使用。

Spring Boot 中的设置：

```
rabbitTemplate.setChannelTransacted(true);
```

那么有没有其他可以保证消息被 Broker 接收，但是又不大量消耗性能的方式呢？这个就是第二种模式，叫做确认（Confirm）模式。

Confirm（确认）模式

确认模式有三种，一种是普通确认模式。

在生产者这边通过调用 channel.confirmSelect()方法将信道设置为 Confirm 模式，然后发送消息。一旦消息被投递到所有匹配的队列之后，RabbitMQ 就会发送一个确认（Basic.Ack）给生产者，也就是调用 channel.waitForConfirms()返回 true，这样生产者就知道消息被服务端接收了。

这种发送 1 条确认 1 条的方式消息还不是太高，所以我们还有一种批量确认的方式。批量确认，就是在开启 Confirm 模式后，先发送一批消息。只要 channel.waitForConfirmsOrDie();方法没有抛出异常，就代表消息都被服务端接收了。

批量确认的方式比单条确认的方式效率要高，但是也有两个问题，第一个就是批量的数量的确定。对于不同的业务，到底发送多少条消息确认一次？数量太少，效率提升不上去。数量多的话，又会带来另一个问题，比如我们发 1000 条消息才确认一次，如果前面 999 条消息都被服务端接收了，如果第 1000 条消息被拒绝了，那么前面所有的消息都要重发。

有没有一种方式，可以一边发送一边确认的呢？这个就是异步确认模式。

异步确认模式需要添加一个 ConfirmListener，并且用一个 SortedSet 来维护没有被确认的消息。

Confirm 模式是在 Channel 上开启的，因为 RabbitTemplate 对 Channel 进行了封装，叫做 ConfimrCallback。

```
rabbitTemplate.setConfirmCallback(new RabbitTemplate.ConfirmCallback() {
    @Override
    public void confirm(CorrelationData correlationData, boolean ack, String cause) {
        if (!ack) {
            System.out.println("发送消息失败：" + cause);
            throw new RuntimeException("发送异常：" + cause);
        }
    }
});
```

参考：

gupaoedu-vip-rabbitmq-javaapi：com.gupaoedu.confirm

gupaoedu-vip-springboot-amqp：com.gupaoedu.amqp.template.TemplateConfig

1.1.2. 消息从交换机路由到队列

第二个环节就是消息从交换机路由到队列。在什么情况下，消息会无法路由到正确的队列？可能因为路由键错误，或者队列不存在。

我们有两种方式处理无法路由的消息，一种就是让服务端重发给生产者，一种是让交换机路由到另一个备份的交换机。

消息回发的方式：使用 mandatory 参数和 ReturnListener（在 Spring AMQP 中是 ReturnCallback）。

```
rabbitTemplate.setMandatory(true);
rabbitTemplate.setReturnCallback(new RabbitTemplate.ReturnCallback(){
```

```
        public void returnedMessage(Message message,
                                    int replyCode,
                                    String replyText,
                                    String exchange,
                                    String routingKey){
            System.out.println("回发的消息：");
            System.out.println("replyCode: "+replyCode);
            System.out.println("replyText: "+replyText);
            System.out.println("exchange: "+exchange);
            System.out.println("routingKey: "+routingKey);
        }
    });
```

消息路由到备份交换机的方式：在创建交换机的时候，从属性中指定备份交换机。

```
Map<String,Object> arguments = new HashMap<String,Object>();
arguments.put("alternate-exchange","ALTERNATE_EXCHANGE"); // 指定交换机的备份交换机

channel.exchangeDeclare("TEST_EXCHANGE","topic", false, false, false, arguments);
```

（注意区别，队列可以指定死信交换机；交换机可以指定备份交换机）

参考：

gupaoedu-vip-rabbitmq-javaapi：com.gupaoedu.returnlistener

gupaoedu-vip-springboot-amqp：com.gupaoedu.amqp.template.TemplateConfig

1.1.3. 消息在队列存储

第三个环节是消息在队列存储，如果没有消费者的话，队列一直存在在数据库中。

如果 RabbitMQ 的服务或者硬件发生故障，比如系统宕机、重启、关闭等等，可能会导致内存中的消息丢失，所以我们要把消息本身和元数据（队列、交换机、绑定）都保存到磁盘。

解决方案：

1、队列持久化

RabbitConfig.java

```
@Bean("GpQueue")
public Queue GpQueue() {
    // queueName, durable, exclusive, autoDelete, Properties
    return new Queue("GP_TEST_QUEUE", true, false, false, new HashMap<>());
}
```

2、交换机持久化

```
@Bean("GpExchange")
public DirectExchange exchange() {
    // exchangeName, durable, exclusive, autoDelete, Properties
    return new DirectExchange("GP_TEST_EXCHANGE", true, false, new HashMap<>());
}
```

3、消息持久化

gupaoedu-vip-springboot-amqp 工程：

com.gupaoedu.reliable.delivery.producer.ProducerApp

```
MessageProperties messageProperties = new MessageProperties();
messageProperties.setDeliveryMode(MessageDeliveryMode.PERSISTENT);
Message message = new Message("持久化消息".getBytes(), messageProperties);


rabbitTemplate.send("GP_TEST_EXCHANGE", "gupao.test", message);
```

4、集群

如果只有一个 RabbitMQ 的节点，即使交换机、队列、消息做了持久化，如果服务崩溃或者硬件发生故障，RabbitMQ 的服务一样是不可用的，所以为了提高 MQ 服务的可用性，保障消息的传输，我们需要有多个 RabbitMQ 的节点，在下一节会详细讲到。

1.1.4. 消息投递到消费者

如果消费者收到消息后没来得及处理即发生异常，或者处理过程中发生异常，会导致④失败。服务端应该以某种方式得知消费者对消息的接收情况，并决定是否重新投递这条消息给其他消费者。

RabbitMQ 提供了消费者的消息确认机制（message acknowledgement），消费者可以自动或者手动地发送 ACK 给服务端。

没有收到 ACK 的消息，消费者断开连接后，RabbitMQ 会把这条消息发送给其他消费者。如果没有其他消费者，消费者重启后会重新消费这条消息，重复执行业务逻辑。

消费者在订阅队列时，可以指定 autoAck 参数，当 autoAck 等于 false 时，RabbitMQ 会等待消费者显式地回复确认信号后才从队列中移去消息。

如何设置手动 ACK?

SimpleRabbitListenerContainer 或者 SimpleRabbitListenerContainerFactory

```
factory.setAcknowledgeMode(AcknowledgeMode.MANUAL);
```

application.properties

```
spring.rabbitmq.listener.direct.acknowledge-mode=manual
spring.rabbitmq.listener.simple.acknowledge-mode=manual
```

注意这三个值的区别：

NONE：自动 ACK

MANUAL： 手动 ACK

AUTO：如果方法未抛出异常，则发送 ack。

当抛出 AmqpRejectAndDontRequeueException 异常的时候，则消息会被拒绝，且不重新入队。当抛出 ImmediateAcknowledgeAmqpException 异常，则消费者会发送 ACK。其他的异常，则消息会被拒绝，且 requeue = true 会重新入队。

在 Spring Boot 中，消费者又怎么调用 ACK，或者说怎么获得 Channel 参数呢？

引入 com.rabbitmq.client.Channel。

参考：gupaoedu-vip-springboot-project ： com.gupaoedu.consumer.SecondConsumer

```
public class SecondConsumer {
    @RabbitHandler
    public void process(String msgContent,Channel channel, Message message) throws IOException {
        System.out.println("Second Queue received msg : " + msgContent );
        channel.basicAck(message.getMessageProperties().getDeliveryTag(), false);
    }
}
```

如果消息无法处理或者消费失败，也有两种拒绝的方式，Basic.Reject()拒绝单条，Basic.Nack()批量拒绝。如果 requeue 参数设置为 true，可以把这条消息重新存入队列，以便发给下一个消费者（当然，只有一个消费者的时候，这种方式可能会出现无限循环重复消费的情况。可以投递到新的队列中，或者只打印异常日志）。

思考：服务端收到了 ACK 或者 NACK，生产者会知道吗？即使消费者没有接收到消息，或者消费时出现异常，生产者也是完全不知情的。

例如，我们寄出去一个快递，是怎么知道收件人有没有收到的？因为有物流跟踪和签收反馈，所以寄件人可以知道。

在没有用上电话的年代，我们寄出去一封信，是怎么知道收信人有没有收到信件？只有收到回信，才知道寄出的信被收到了。

所以，这个是生产者最终确定消费者有没有消费成功的两种方式：

1） 消费者收到消息，处理完毕后，调用生产者的 API（思考：是否破坏解耦？）

2） 消费者收到消息，处理完毕后，发送一条响应消息给生产者

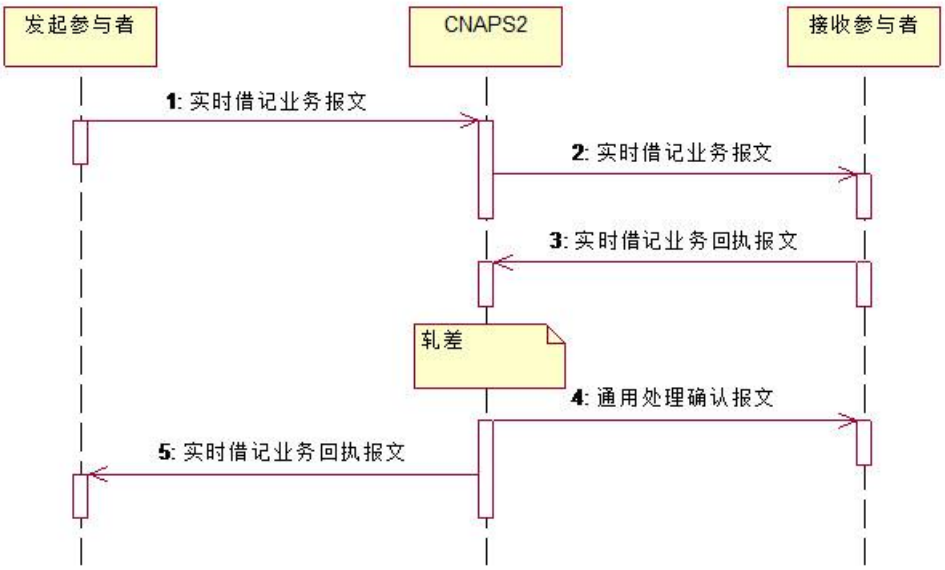
1.1.5. 消费者回调

1） 调用生产者 API

例如：提单系统给其他系统发送了碎屏保消息后，其他系统必须在处理完消息后调用提单系统提供的 API，来修改提单系统中数据的状态。只要 API 没有被调用，数据状态没有被修改，提单系统就认为下游系统没有收到这条消息。

2） 发送响应消息给生产者

例如：商业银行与人民银行二代支付通信，无论是人行收到了商业银行的消息，还是商业银行收到了人行的消息，都必须发送一条响应消息（叫做回执报文）。



1.1.6. 补偿机制

如果生产者的 API 就是没有被调用，也没有收到消费者的响应消息，怎么办?

不要着急，可能是消费者处理时间太长或者网络超时。

生产者与消费者之间应该约定一个超时时间，比如 5 分钟，对于超出这个时间没有得到响应的消息，可以设置一个定时重发的机制，但要发送间隔和控制次数，比如每隔 2 分钟发送一次，最多重发 3 次，否则会造成消息堆积。

重发可以通过消息落库+定时任务来实现。

重发，是否发送一模一样的消息?

参考:

ATM 机上运行的系统叫 C 端（ATMC），前置系统叫 P 端（ATMC），它接收 ATMC 的消息，再转发给卡系统或者核心系统。

1）如果客户存款，没有收到核心系统的应答，不知道有没有记账成功，最多发送 5

次存款确认报文，因为已经吞钞了，所以要保证成功；

2）如果客户取款，ATMC 未得到应答时，最多发送 5 次存款冲正报文。因为没有吐钞，所以要保证失败。

1.1.7. 消息幂等性

如果消费者每一次接收生产者的消息都成功了，只是在响应或者调用 API 的时候出了问题，会不会出现消息的重复处理？例如：存款 100 元，ATM 重发了 5 次，核心系统一共处理了 6 次，余额增加了 600 元。

所以，为了避免相同消息的重复处理，必须要采取一定的措施。RabbitMQ 服务端是没有这种控制的（同一批的消息有个递增的 DeliveryTag），它不知道你是不是就要把一条消息发送两次，只能在消费端控制。

如何避免消息的重复消费?

消息出现重复可能会有两个原因：

1、生产者的问题，环节①重复发送消息，比如在开启了 Confirm 模式但未收到确认，消费者重复投递。

2、环节④出了问题，由于消费者未发送 ACK 或者其他原因，消息重复投递。

3、生产者代码或者网络问题。

对于重复发送的消息，可以对每一条消息生成一个唯一的业务 ID，通过日志或者消息落库来做重复控制。

参考：银行的重账控制环节。

1.1.8. 最终一致

如果确实是消费者宕机了，或者代码出现了 BUG 导致无法正常消费，在我们尝试多次重发以后，消息最终也没有得到处理，怎么办?

例如存款的场景，客户的钱已经被吞了，但是余额没有增加，这个时候银行出现了长款，应该怎么处理？如果客户没有主动通知银行，这个问题是怎么发现的？银行最终怎么把这个账务做平?

在我们的金融系统中，都会有双方对账或者多方对账的操作，通常是在一天的业务结束之后，第二天营业之前。我们会约定一个标准，比如 ATM 跟核心系统对账，肯定是以核心系统为准。ATMC 获取到核心的对账文件，然后解析，登记成数据，然后跟自己记录的流水比较，找出核心有 ATM 没有，或者 ATM 有核心没有，或者两边都有但是金额不一致的数据。

对账之后，我们再手工平账。比如取款记了账但是没吐钞的，做一笔冲正。存款吞了钞没记账的，要么把钱退给客户，要么补一笔账。

1.1.9. 消息的顺序性

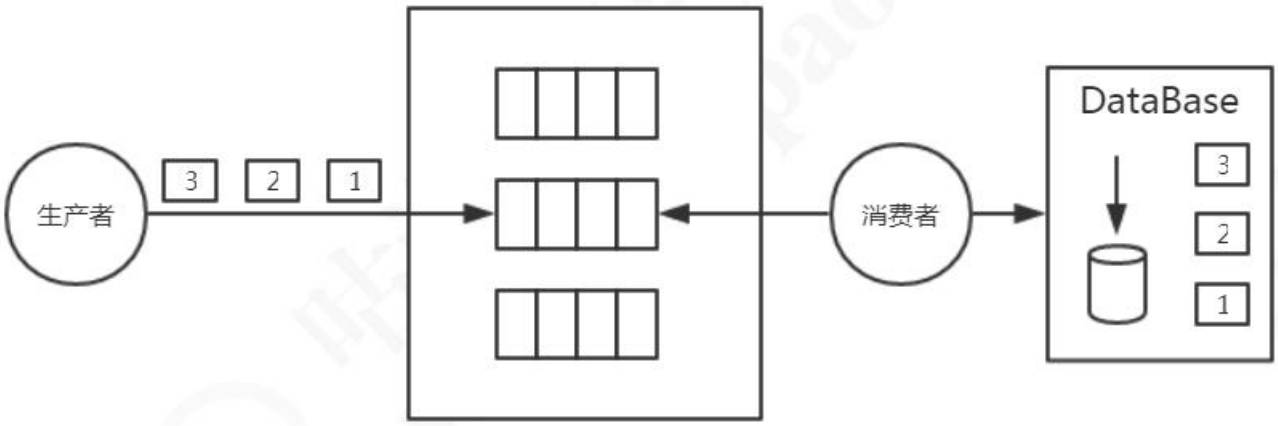
消息的顺序性指的是消费者消费消息的顺序跟生产者生产消息的顺序是一致的。

例如：商户信息同步到其他系统，有三个业务操作：1、新增门店 2、绑定产品 3、激活门店，这种情况下消息消费顺序不能颠倒（门店不存在时无法绑定产品和激活）。

又比如：1、发表微博；2、发表评论；3、删除微博。顺序不能颠倒。

在 RabbitMQ 中，一个队列有多个消费者时，由于不同的消费者消费消息的速度是不一样的，顺序无法保证。只有一个队列仅有一个消费者的情况才能保证顺序消费（不

同的业务消息发送到不同的专用的队列）。



除非负载的场景，不要用多个消费者消费消息。

1.2. 集群与高可用

1.2.1. 为什么要做集群？

集群主要用于实现高可用与负载均衡。

高可用：如果集群中的某些 MQ 服务器不可用，客户端还可以连接到其他 MQ 服务器。

负载均衡：在高并发的场景下，单台 MQ 服务器能处理的消息有限，可以分发给多台 MQ 服务器。

RabbitMQ 有两种集群模式：普通集群模式和镜像队列模式。

1.2.2. RabbitMQ 如何支持集群？

应用做集群，需要面对数据同步和通信的问题。因为 Erlang 天生具备分布式的特性，所以 RabbitMQ 天然支持集群，不需要通过引入 ZK 或者数据库来实现数据同步。

RabbitMQ 通过/var/lib/rabbitmq/.erlang.cookie 来验证身份，需要在所有节点上保持一致。

1.2.3.RabbitMQ 的节点类型？

集群有两种节点类型，一种是磁盘节点（Disc Node），一种是内存节点（RAM Node）。

磁盘节点：将元数据（包括队列名字属性、交换机的类型名字属性、绑定、vhost）放在磁盘中。

内存节点：将元数据放在内存中。

PS：内存节点会将磁盘节点的地址存放在磁盘（不然重启后就没有办法同步数据了）。如果是持久化的消息，会同时存放在内存和磁盘。

集群中至少需要一个磁盘节点用来持久化元数据，否则全部内存节点崩溃时，就无从同步元数据。未指定类型的情况下，默认为磁盘节点。

我们一般把应用连接到内存节点（读写快），磁盘节点用来备份。

集群通过 25672 端口两两通信，需要开放防火墙的端口。

需要注意的是，RabbitMQ 集群无法搭建在广域网上，除非使用 federation 或者 shovel 等插件（没这个必要，在同一个机房做集群）。

集群的配置步骤：

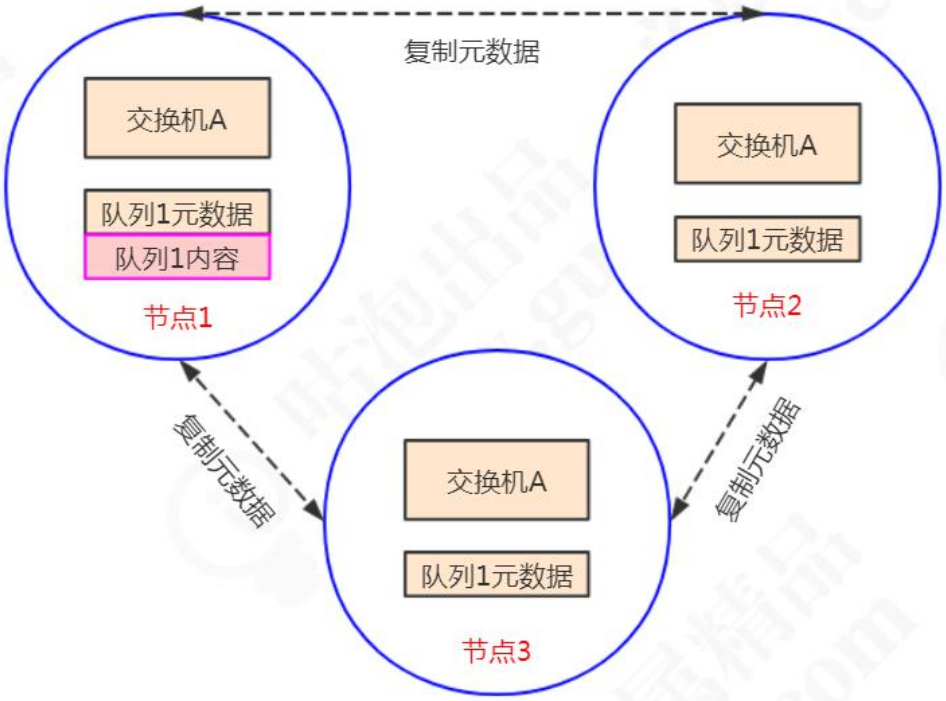
1、配置 hosts

2、同步 erlang.cookie

3、加入集群（join cluster）

1.2.4. 普通集群

普通集群模式下，不同的节点之间只会相互同步元数据。



疑问：为什么不直接把队列的内容（消息）在所有节点上复制一份？

主要是出于存储和同步数据的网络开销的考虑，如果所有节点都存储相同的数据，就无法达到线性地增加性能和存储容量的目的（堆机器）。

假如生产者连接的是节点 3，要将消息通过交换机 A 路由到队列 1，最终消息还是会转发到节点 1 上存储，因为队列 1 的内容只在节点 1 上。

同理，如果消费者连接是节点 2，要从队列 1 上拉取消息，消息会从节点 1 转发到节点 2。其它节点起到一个路由的作用，类似于指针。

普通集群模式不能保证队列的高可用性，因为队列内容不会复制。如果节点失效将导致相关队列不可用，因此我们需要第二种集群模式。

1.2.5. 镜像集群

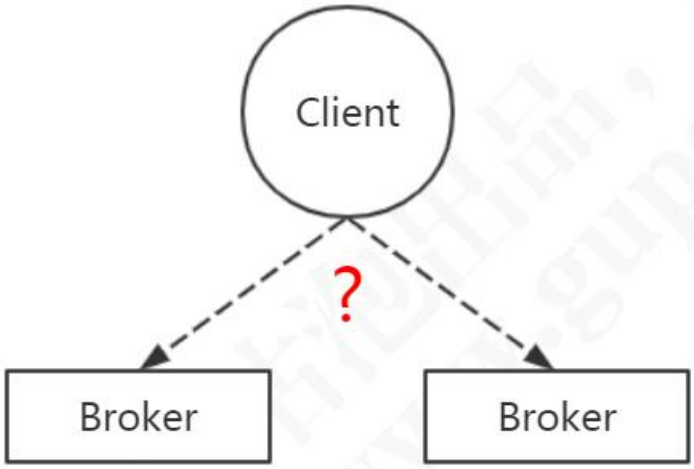
第二种集群模式叫做镜像队列。

镜像队列模式下，消息内容会在镜像节点间同步，可用性更高。不过也有一定的副作用，系统性能会降低，节点过多的情况下同步的代价比较大。

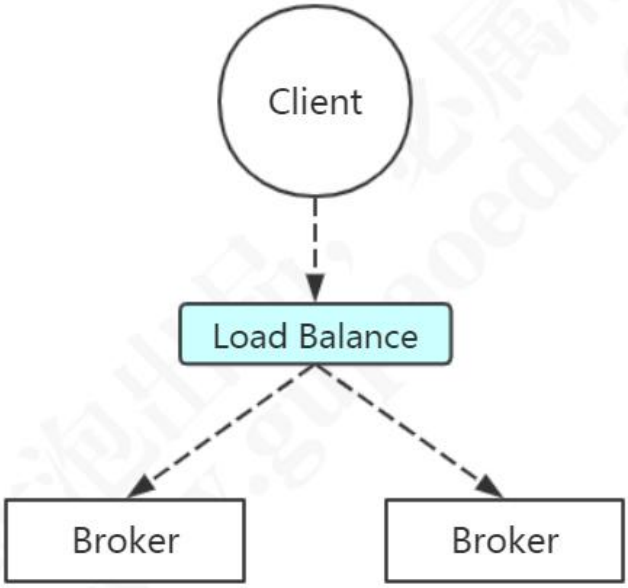
操作方式	命令或步骤
rabbitmqctl (Windows)	rabbitmqctl set_policy ha-all "^ha." "{""ha-mode"":""all""}"
HTTP API	PUT /api/policies/%2f/ha-all {"pattern":"^ha.", "definition":{"ha-mode":"all"}}
Web UI	1、avigate to Admin > Policies > Add / update a policy 2、Name 输入：mirror_image 3、Pattern 输入：^（代表匹配所有） 4、Definition 点击 HA mode，右边输入：all 5、Add policy

1.2.6. 高可用

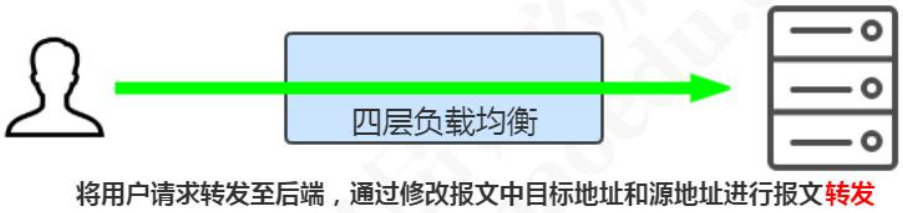
集群搭建成功后，如果有多个内存节点，那么生产者和消费者应该连接到哪个内存节点？如果在我们的代码中根据一定的策略来选择要使用的服务器，那每个地方都要修改，客户端的代码就会出现很多的重复，修改起来也比较麻烦。

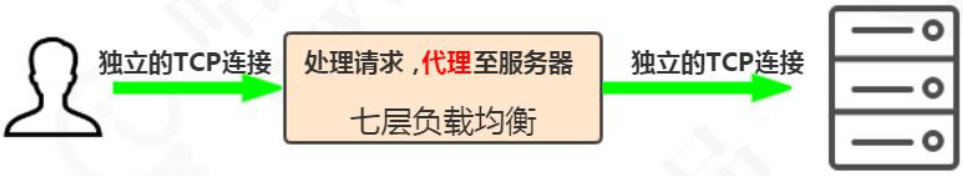


所以需要一个负载均衡的组件（例如 HAProxy，LVS，Nignx），由负载的组件来做路由。这个时候，只需要连接到负载组件的 IP 地址就可以了。



负载分为四层负载和七层负载。



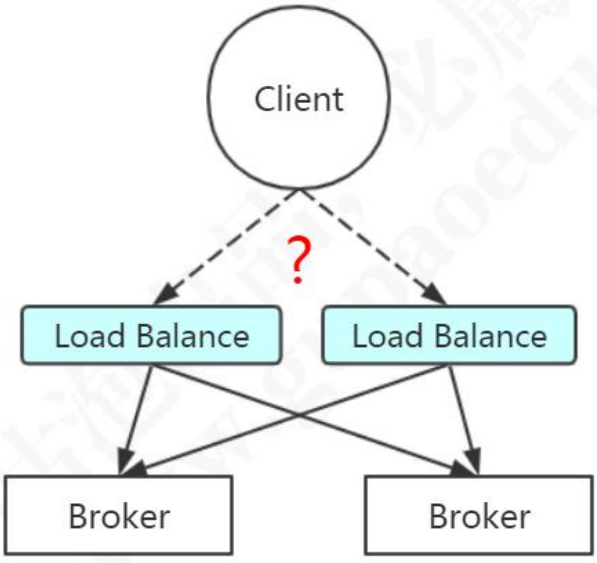


四层负载：工作在 OSI 模型的第四层，即传输层（TCP 位于第四层），它是根据 IP 端口进行转发（LVS 支持四层负载）。RabbitMQ 是 TCP 的 5672 端口。

七层负载：工作在第七层，应用层（HTTP 位于第七层）。可以根据请求资源类型分配到后端服务器（Nginx 支持七层负载；HAProxy 支持四层和七层负载）。

https://www.cnblogs.com/readygood/p/9757951.html

但是，如果这个负载的组件也挂了呢？客户端就无法连接到任意一台 MQ 的服务器了。所以负载软件本身也需要做一个集群。新的问题又来了，如果有两台负载的软件，客户端应该连哪个？



负载之上再负载？陷入死循环了。这个时候我们就要换个思路了。

我们应该需要这样一个组件：

1、 它本身有路由（负载）功能，可以监控集群中节点的状态（比如监控 HAProxy），如果某个节点出现异常或者发生故障，就把它剔除掉。

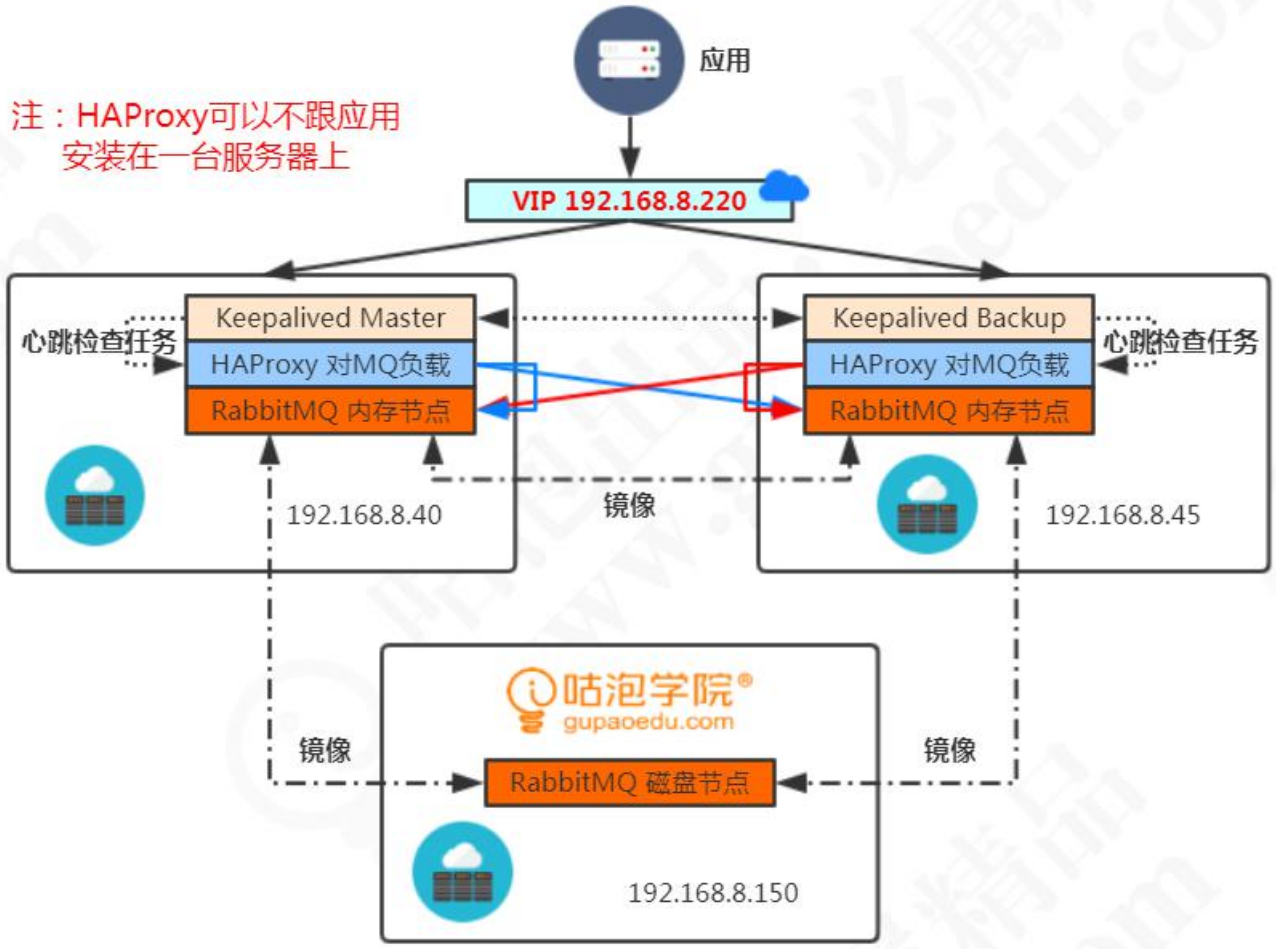
2、 为了提高可用性，它也可以部署多个服务，但是只有一个自动选举出来的 MASTER 服务器（叫做主路由器），通过广播心跳消息实现。

3、 MASTER 服务器对外提供一个虚拟 IP，提供各种网络功能。也就是谁抢占到 VIP，就由谁对外提供网络服务。应用端只需要连接到这一个 IP 就行了。

这个协议叫做 VRRP 协议（虚拟路由冗余协议 Virtual Router Redundancy Protocol），这个组件就是 Keepalived，它具有 Load Balance 和 High Availability 的功能。

下面我们看下用 HAProxy 和 Keepalived 如何实现 RabbitMQ 的高可用（MySQL、Mycat、Redis 类似）。

1.2.7. 基于 Docker 安装 HAproxy 负载+Keepalived 高可用



规划：

内存节点 1：192.168.8.40

内存节点 2：192.168.8.45

磁盘节点：192.168.8.150

VIP：192.168.8.220

1、我们规划了两个内存节点，一个磁盘节点。所有的节点之间通过镜像队列的方式同步数据。内存节点用来给应用访问，磁盘节点用来持久化数据。

2、为了实现对两个内存节点的负载，我们安装了两个 HAProxy，监听两个 5672 和 15672 的端口。

3、安装两个 Keepalived，一主一备。两个 Keepalived 抢占一个 VIP192.168.8.220。谁抢占到这个 VIP，应用就连接到谁，来执行对 MQ 的负载。

这种情况下，我们的 Keepalived 挂了一个节点，没有影响，因为 BACKUP 会变成 MASTER，抢占 VIP。HAProxy 挂了一个节点，没有影响，我们的 VIP 会自动路由的可用的 HAProxy 服务。RabbitMQ 挂了一个节点，没有影响， 因为 HAProxy 会自动负载到可用的节点。

1.3. 实践经验总结

1.2.8. 资源管理

到底在消费者创建还是在生产者创建?

如果 A 项目和 B 项目有相互发送和接收消息,应该创建几个 vhost,几个 Exchange?

交换机和队列，实际上是作为资源，由运维管理员创建的。

为什么仍然需要在代码中定义？重复创建不报错吗?

RabbitMQ 生 产 者 资 源 申 请 单

申请人姓名		申请人联系方式		部门	
申请时间		岗位		期望完成时间	
创建位置	□[redacted]IDC □[redacted]IDC □[redacted] □阿里云				
申请类型	□PRD环境	□UAT环境		□SIT环境	
申请原因					
需求描述					
项目名称					
Virtual-host					
消息模式	□单一模式 □普通模式 □镜像模式				
Queue名称					
Exchange模式	□Direct exchange □Topic exchange □Fanout exchange □Headers exchange				
Exchange名称					
Bingding key					
其他要求					
开通周期	年 月 日 至 年 月 日				
需求审批					
申请部门经理审批			中间件管理部门审批		
年 月 日			年 月 日		
（以下由Rabbitmq管理员填写并交付）					
IP地址					
PORT端口					
用户账号					

RabbitMQ 消 费 者 资 源 申 请 单					
申请人姓名		申请人联系方式		部门	
申请时间		岗位		期望完成时间	
创建位置	□ IDC □ IDC □ □阿里云				
申请类型	□PRD环境	□UAT环境		□SIT环境	
申请原因					
需求描述					
项目名称					
Queue名称					
Exchange模式	□Direct exchange □Topic exchange □Fanout exchange □Headers exchange				
Exchange名称					
Bingding key					
其他要求					
开通周期	年 月 日 至 年 月 日				
需求审批					
申请部门经理审批		中间件管理部门审批			
年 月 日		年 月 日			
（以下由Rabbitmq管理员填写并交付）					
IP地址					
PORT端口					
用户账号					
Virtual-host					

1.2.9. 配置文件与命名规范

1、元数据的命名集中放在 properties 文件中，不要用硬编码。如果有多个系统，可以配置多个 xxx_mq.properties。

2、命名体现元数据的类型

虚拟机命名： XXX_VHOST

交换机命名：XXX_EXCHANGE

队列命名：_QUEUE

3、命名体现数据来源和去向

例如：销售系统发往产品系统的交换机：SALE_TO_PRODUCT_EXCHANGE。做到见名知义，不用去查文档（当然注释是必不可少的）。

1.2.10. 调用封装

在项目中可以对 Template 做进一步封装，简化消息的发送。

例如：如果交换机、路由键是固定的，封装之后就只需要一个参数：消息内容。

另外，如果想要平滑地迁移不同的 MQ（如果有这种需求的话），也可以再做一层简单的封装。

```
GpSendMsg(){
    JmsTemplate.send(destination,msg);
}
```

这时，如果要把 ActiveMQ 替换为 RabbitMQ，只需要修改：

```
GpSendMsg(){
    RabbitTemplate.send(exchange,routingKey,msg);
}
```

1.2.11. 信息落库+定时任务

将需要发送的消息保存在数据库中，可以实现消息的可追溯和重复控制，需要配合定时任务来实现。

1） 将需要发送的消息登记在消息表中。

2） 定时任务一分钟或半分钟扫描一次，将未发送的消息发送到 MQ 服务器，并且修改状态为已发送。

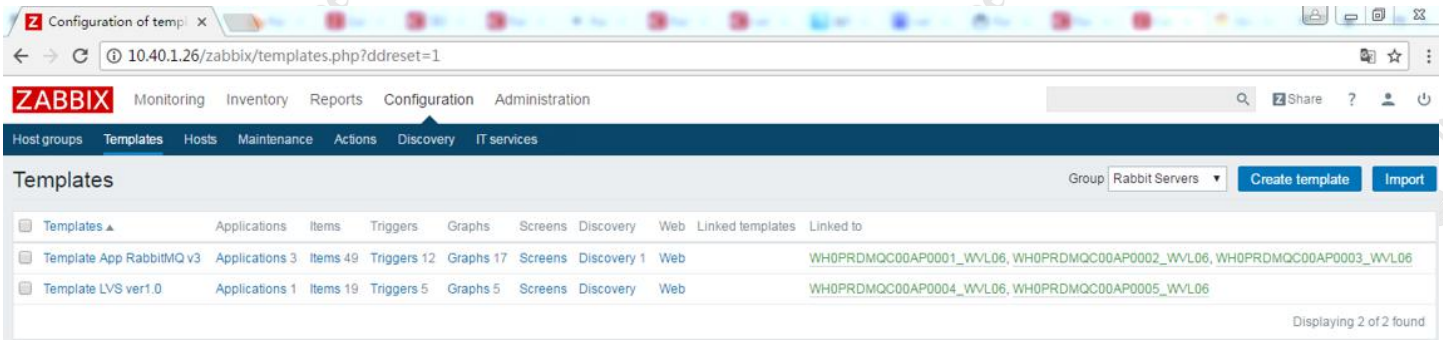
3） 如果需要重发消息，将指定消息的状态修改为未发送即可。

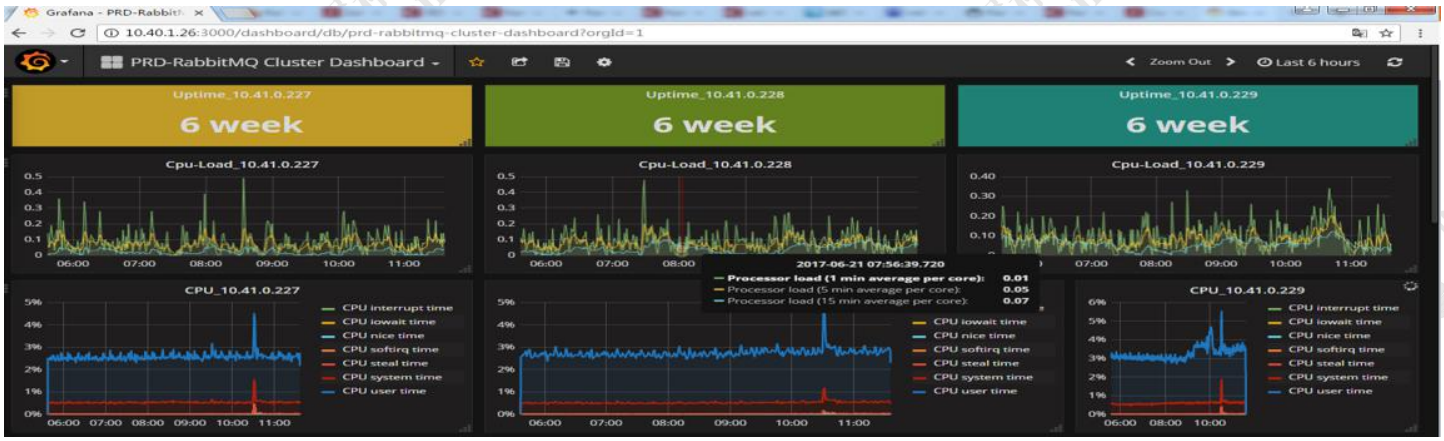
副作用：降低效率，浪费存储空间。

1.2.12. 生产环境运维监控

虽然 RabbitMQ 提供了一个简单的管理界面，但是如果对于系统性能、高可用和其他参数有一些定制化的监控需求的话，我们就需要通过其他方式来实现监控了。

主要关注：磁盘、内存、连接数





1.2.13. 日志追踪

RabbitMQ 可以通过 Firehose 功能来记录消息流入流出的情况，用于调试，排错。

它是通过创建一个 TOPIC 类型的交换机（amq.rabbitmq.trace），把生产者发送给 Broker 的消息或者 Broker 发送给消费者的消息发到这个默认的交换机上面来实现的。

另外 RabbitMQ 也提供了一个 Firehose 的 GUI 版本，就是 Tracing 插件。

启用 Tracing 插件后管理界面右侧选项卡会多一个 Tracing，可以添加相应的策略。

RabbitMQ 还提供了其他的插件来增强功能。

https://www.rabbitmq.com/firehose.html

https://www.rabbitmq.com/plugins.html

1.2.14. 如何减少连接数

在发送大批量消息的情况下，创建和释放连接依然有不小的开销。我们可以跟接收方约定批量消息的格式，比如支持 JSON 数组的格式，通过合并消息内容，可以减少生产者/消费者与 Broker 的连接。

比如：活动过后，要全范围下线产品，通过 Excel 导入模板，通常有几万到几十万条解绑数据，合并发送的效率更高。

建议单条消息不要超过 4M（4096KB），一次发送的消息数需要合理地控制。

```
/**
 * 将MQ消息分成10个一条记录，并批量登记数据库
 * @param list
 * @param dType
 */
private void partAndInsertMq(List<StoreProdToAlsBean> list,String dType) {
    // 1条记录包含10个MQ消息
    List<List<StoreProdToAlsBean>> partLists = getAverRange(list,10);
    List<MQTask> mqList = new ArrayList<MQTask>();
    for( List<SimsStoreProdToAlsBean> partList: partLists ){
        JSONArray jsonArry = JSONArray.fromObject(partList);
        JSONObject pdDelJson = new JSONObject();
        pdDelJson.put("data", jsonArry);
        pdDelJson.put("title", "storeProd");
        pdDelJson.put("dType", dType);
        MQTask mqMsg = new MQTask();
        mqMsg.setRoutingKey(Constant.MQRoutingKeys.MQ_CXF.getRoutingKey());
        mqMsg.setJsonDatastr(pdDelJson.toString());
        Long regDay = Long.parseLong(DatesUtil.dateToString(new Date(), DatesUtil.SIMPLE_TIME));
        mqMsg.setRegDay(regDay);
        mqMsg.setLastUpTms(regDay);
        mqMsg.setMqStatus("0");

        mqList.add(mqMsg);
    }
    NewBatchImportDao.batachInsertMQTask(mqList);
}
```

```java
/**
 * 把一个数组按每份N个划分，返回一个大的数组
 * @param list 数组
 * @param average 每份包含多少个
 * @return 新的数组
 */
public <T> List<List<T>> getAverRange( List<T> list, int average ){
    List<List<T>> newArrays = new ArrayList<List<T>>();
    int start = 0;
    int end = 0;
    for (int i = 0; i <= list.size() / average; i++) {
        List<T> partList = new ArrayList<T>();
        end = start + average;
        if (start >= list.size()) {
            break;
        }
        if (end >= list.size()) {
            end = list.size() ;
        }
        partList = list.subList(start, end);
        newArrays.add(partList);
        start = end;
    }
    return newArrays;
}
```

```
msgContent = {
   "type":"add",
   "num":3,
   "detail": [
      { "merchName":"黄金手机店", "address":"黄金路 999 号" ] },
      { "merchName":"银星手机店", "address":"银星路 168 号" ] },
      { "merchName":"青山手机店", "address":"青山路 73 号" ] }
  ]
}
```

1.4. 面试题

1、 消息队列的作用与使用场景？

要点：关键词+应用场景

2、 Channel 和 vhost 的作用是什么?

Channel：减少 TCP 资源的消耗。也是最重要的编程接口。

Vhost：提高硬件资源利用率，实现资源隔离。

3、 RabbitMQ 的消息有哪些路由方式？适合在什么业务场景使用?

Direct、Topic、Fanout

4、 交换机与队列、队列与消费者的绑定关系是什么样的?

多个消费者监听一个队列时（比如一个服务部署多个实例），消息会重复消费吗?

多对多;

轮询（平均分发）

5、 无法被路由的消息，去了哪里?

直接丢弃。可用备份交换机（alternate-exchange）接收。

6、 消息在什么时候会变成 Dead Letter（死信）？

消息过期；消息超过队列长度或容量；消息被拒绝并且未设置重回队列

7、 如果一个项目要从多个服务器接收消息，怎么做?

如果一个项目要发送消息到多个服务器，怎么做?

定义多个 ConnectionFactory，注入到消费者监听类/Temaplate。

8、 RabbitMQ 如何实现延迟队列?

基于数据库+定时任务;

或者消息过期+死信队列;

或者延迟队列插件。

9、 哪些情况会导致消息丢失？怎么解决?

哪些情况会导致消息重复？怎么解决?

从消息发送的整个流程来分析。

10、一个队列最多可以存放多少条消息?

由硬件决定。

11、 可以用队列的 x-max-length 最大消息数来实现限流吗？例如秒杀场景。

不能，因为会删除先入队的消息，不公平。

12、 如何提高消息的消费速率?

创建多个消费者。

13、 AmqpTemplate 和 RabbitTemplate 的区别?

Spring AMQP 是 Spring 整合 AMQP 的一个抽象。Spring-Rabbit 是一个实现。

14、 如何动态地创建消费者监听队列?

通过 ListenerContainer

com.gupaoedu.amqp.container.ContainerConfig.java

```
container.setQueues(getSecondQueue(), getThirdQueue()); //监听的队列
```

15、 Spring AMQP 中消息怎么封装？用什么转换?

Message，MessageConvertor

16、 如何保证消息的顺序性?

一个队列只有一个消费者

17、 RabbitMQ 的集群节点类型?

磁盘节点和内存节点

18、 如何保证 RabbitMQ 的高可用?

HAProxy（LVS）+Keepalived

19、 大量消息堆积怎么办?

1） 重启（不是开玩笑的）

2） 多创建几个消费者同时消费

3） 直接清空队列，重发消息

20、MQ 选型分析?

使用和管理

API、与 Spring 集成

管理控制台，权限、安全、监控

扩展性

社区支持

性能

并发性

消息吞吐量

消息堆积能力

功能

事务性消息

顺序性消息

消息重试

死信队列

优先级队列

延迟队列

可靠+可用

集群

持久化

消息同步

21、 设计一个 MQ，你的思路是什么?

存储与转发。

存储：内存：用什么数据结构?

磁盘：文件系统？数据库?

通信：通信协议（TCP HTTP AMQP ）？一对一？一对多?

推模式？拉模式?

其他特性……

作者：咕泡学院-青山

最后更新时间：2019 年 8 月 26 日 09:21:06