RabbitMQ-AMQP-0-9-1模型详解

Vhost

创建方式：

- 命令行工具来创建： rabbitmqctl add_vhost qa1
- 管理控制台创建

Connection

Spring CachingConnectionFactory

默认情况下 单连接的连接工厂。

默认，首先会缓存一个Channel，然后再慢慢增 ，高并发使用场景、channelCacheSize 设置缓存的数量 100

默认的缓存模式是缓存通道。

如果把缓存模式设为CacheMode.CONNECTION，则缓存连接以及连接上创建Channel. connectionCacheSize 属性设置缓存多少个连接

CacheMode.CONNECTION 与 Rabbit Admin (AmqpAdmin) 不兼容，不会自动创建exchange、queues 等

exchange

```
Exchange.DeclareOk exchangeDeclare(String exchange,
String type,
boolean durable,    // 交换器是否持久化  避免重启后，要再次创建。 和消息的持久化没关系。
boolean autoDelete,  // 当没有队列绑定到它时 是否自动删除
boolean internal,   // 是否是 MQ 内部使用的， 我们就不能在客户端中使用。
Map<String, Object> arguments)
```

Queue

- **Name** 应用程序可以选择队列名称，或者要求代理为它们生成一个名称，最长 255 字节 UTF-8字符。如想要Broker为我们生成队列名，可以在声明创建Queue时传入空字符""，在返回值中可以取得生成的队列名。
- **Durable** 是否持久存储，如为false， broker restart就没有了
- **Exclusive** 独占，被一个connection独占使用，当connection 关闭时Queue也被删除
- **Auto-delete** 是否在Queue的最后一个消费者关闭时自动删除Queue
- **Arguments** 可选的被插件和Broker特殊特性使用的参数，如message TTL, queue length limit 等

【注意】以amq.开头的队列名称 是保留给Broker内部使用的，如果用户创建这样的队列会异常。

【注意】队列的持久性，跟消息的持久化也没关系。

Queue 的 TTL TIME TO LIVE

autoDelete 队列空闲一段时间之后再删除。

- policy方式

  ```
  rabbitmqctl set_policy expiry ".*" '{"expires":1800000}' --apply-to queues
  ```

  expiry 策略名称 自定义

  ".*" 作用目标名称的正则表达式

  '{"expires":1800000}' 策略定义 过期时间设置 单位毫秒

  --apply-to queues 应用于哪一类实体

代码中声明队列是指定

```
    Map<String, Object> args = new HashMap<String, Object>();
            args.put("x-expires", 1800000);
            channel.queueDeclare("myqueue", false, false, false, args);

    channel.queueDeclare("queue1", false, false, false, args);
```

Publisher

路由不可达

可能情况：

- 交换没有绑定队列
- 交换没法根据消息的路由key把消息路由到队列。

默认情况： 消息丢弃

可以的处理办法：

- 退回
- 死信队列（备用交换）

退回

void basicPublish(String exchange, String routingKey, boolean mandatory, BasicProperties props, byte[] body)

mandatory：true 强制退回， false 不需退回，直接丢弃。

备用交换

- policy

```
rabbitmqctl set_policy mike "^my-direct$" '{"alternate-exchange":"my-ae"}'
```

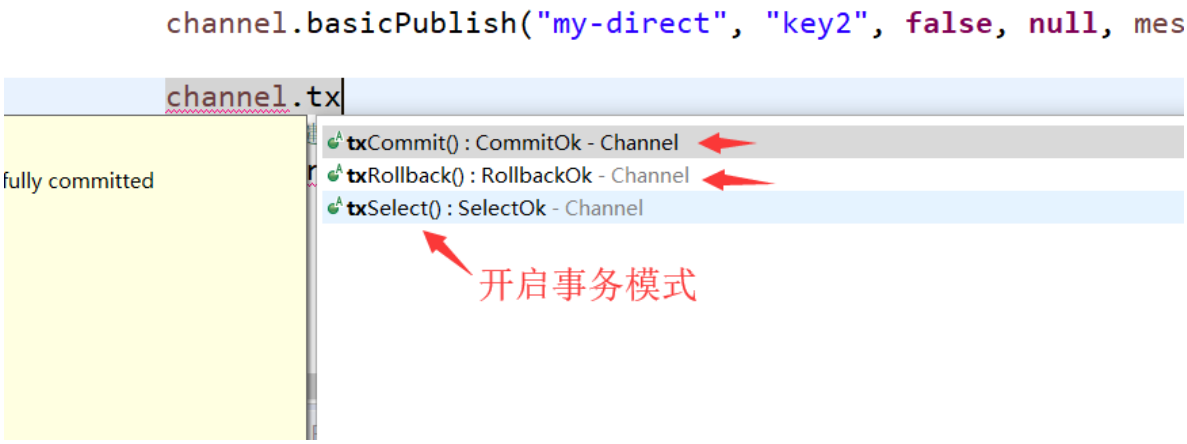
- 代码中声明交换时通过参数指定备用交换

```
//声明参数
Map<String, Object> args = new HashMap<String, Object>();
args.put("alternate-exchange", "my-ae"); //备用交换参数指定
channel.exchangeDeclare("my-direct", "direct", false, false, args);
channel.exchangeDeclare("my-ae", "fanout");
channel.queueDeclare("routed");
channel.queueBind("routed", "my-direct", "key1");
channel.queueDeclare("unrouted");
channel.queueBind("unrouted", "my-ae", "");
```

一定要去动手

事务机制

可靠发布



spring 事务管理需要那些组件?

事务管理器 TransactionManager

```
@Configuration
public class TxConfiguration {
    @Bean       // 配置事务管理器
    public RabbitTransactionManager rabbitTransactionManager(ConnectionFactory
connectionFactory) {
        return new RabbitTransactionManager(connectionFactory);
    }
}
```

在spring 该怎么玩事务就怎么玩.

RabbitTransactionManager 只能做Rabbitmq的消息事务管理 只能是单连接的连接工厂

没有分布式事务管理器实现。

rabbitmq中事务机制来保证消息的可靠发布，性能是比较差

发布确认机制

性能是事务机制的250倍。

发布者发布消息，一般是走异步。

channel

有三种确认模式

- 异步流式确认 事件驱动 开销低，吞吐量大
- 批量发布确认 批次等待，确认不ok 一批重发
- 单条确认 发一条就等待确认

broker给出确认会有三种结果

- ack 接收成功
- nack 接收失败
- 发布者收不到Broker的确认（超时）

异步流式确认

Consumer

两种消息消费模式

- push 推模式

- pull 拉模式

push 模式说明

broker client 消费者

client 向 broker 注册对某个队列的消费者

```
// 对感兴趣的队列注册消费者，返回Server生成的consumerTag（消费者标识）
String consumerTag = channel.basicConsume(queueName, true, callback, consumerTag
-> {});
```

取消消费者注册

```
channel.basicCancel(consumerTag);
```

独占消费者

独占队列：被创建它的连接独占 这个连接上的channel 可以共享。连接关闭，独占队列没有了。

独占消费者：消费者独占一个对队列进行消息消费，适用场景： 消息一定要严格按序消费处理。

消费者优先级

x-priority

```
Map<String, Object> args = new HashMap<String, Object>();
args.put("x-priority", 10); // 整数，数值越大优先级越高。 默认 0
channel.basicConsume("my-queue", false, args, consumer);
```

prefetch 20

Broker 轮询发送， 同优先级时是轮询 消息优先发给优先级高的消费者，直到prefetch满了 或 block.

消息确认

消息传递的模式（确认）

- Automatic 自动 送货无需确认
- Manual 手动 需要客户签收

手动确认的情况 和prefetch 配合使用

prefetch spring中默认值为： 250

手动确认的3个操作

- basic.ack 用于正面确认，消费者确认消息被妥善处理，broker可以移除该消息了。
- basic.nack 用于负面确认，扩展了basic.reject，以支持批量确认。是RabbitMQ对AMQP-0-9-1的扩展。
- basic.reject 用于负面确认

负面确认，可以指示Broker 移除消息以及是否重发。

```
// 批量Nack,并重发
GetResponse gr1 = channel.basicGet("some.queue", false);
GetResponse gr2 = channel.basicGet("some.queue", false);
channel.basicNack(gr2.getEnvelope().getDeliveryTag(), true, true); //第二个参数true表示批量
```

Pull 拉模式消费

/usr/lib/rabbitmq/plugins/

安全

guest用户管理

guest guest

改密码或删除guest用户。

rabbitmqctl delete_user guest

rabbitmqctl change_password guest newpass

访问控制

Vhost 资源 （exchagne queue） 用户

- 资源： exchange、queue
- 权限：RabbitMQ区分了对资源的configure（配置）、write（写）和read（读）操作。
  - configure操作：指创建或销毁资源，或更改它们的行为。
  - write 写操作：将消息注入到资源中。
  - read操作：从资源中读取消息。

```
rabbitmqctl set_permissions [-p vhost] user conf write read
```

- **-p vhost** 指定 vhost 不指定则默认为 "/" vhost。

- **user** 要设置的用户
- **conf** 匹配资源名称的正则表达式，授予用户哪些资源的配置权限。
- **write** 匹配资源名称的正则表达式，授予用户哪些资源的写权限。
- **read** 匹配资源名称的正则表达式，授予用户哪些资源的读权限。

```
rabbitmqctl set_permissions -p my-vhost mike “^mike-.*” “.*” “.*”
```

channel.exchangeDeclare("aaaaa") 不可以

创建 交换 队列 删除 configure

绑定 读写

发消息 写

消费消息 读

具体操作需要的权限说明表

AMQP 0-9-1 Operation		configure	write	read
exchange.declare	(passive=false)	exchange		
exchange.declare	(passive=true)			
exchange.declare	(with AE)	exchange	exchange (AE)	exchange
exchange.delete		exchange		
queue.declare	(passive=false)	queue		
queue.declare	(passive=true)			
queue.declare	(with DLX)	queue	exchange (DLX)	queue
queue.delete		queue		
exchange.bind			exchange (destination)	exchange (source)
exchange.unbind			exchange (destination)	exchange (source)
queue.bind			queue	exchange
queue.unbind			queue	exchange
basic.publish			exchange	
basic.get				queue
basic.consume				queue
queue.purge				queue

topic 权限

设置用户对topic类型的交换器的消息权限

```
rabbitmqctl set_topic_permissions [-p vhost] user exchange write read
```

- write 允许发布的消息的路由键的匹配正则表达式
- read 允许消费的路由键的匹配正则表达式

```
rabbitmqctl set_topic_permissions -p my-vhost mike amq.topic “^mike-.*” “^mike-.*”
```

Lazy Queue 【了解】

消费者可能出现消费速度突然远低于生产者的情况。

Lazy : 今早地把消息存储到磁盘，只在内存中存极少量 。

主要用途： 能够支持非常长的队列（数百万条消息）

```
rabbitmqctl set_policy Lazy "^lazy-queue$" '{"queue-mode":"lazy"}' --apply-to queues
```

```
Map<String, Object> args = new HashMap<String, Object>();
  args.put("x-queue-mode", "lazy");   //defalut  或  lazy
  channel.queueDeclare("myqueue", false, false, false, args);
```

Number of messages	Message body size	Message type	Producers	Consumers
1,000,000	1,000 bytes	persistent	1	0

Queue mode	Queue process memory	Messages in memory	Memory used by messages	Node memory
default	257 MB	386,307	368 MB	734 MB
lazy	159 KB	0	0	117 MB

Flow Control 流控

当积压的消息达到一定量后，RabbitMQ Broker会自动开启流控，即自动间歇性的阻塞生产者，以达到流量控制的目的。

集群高可用

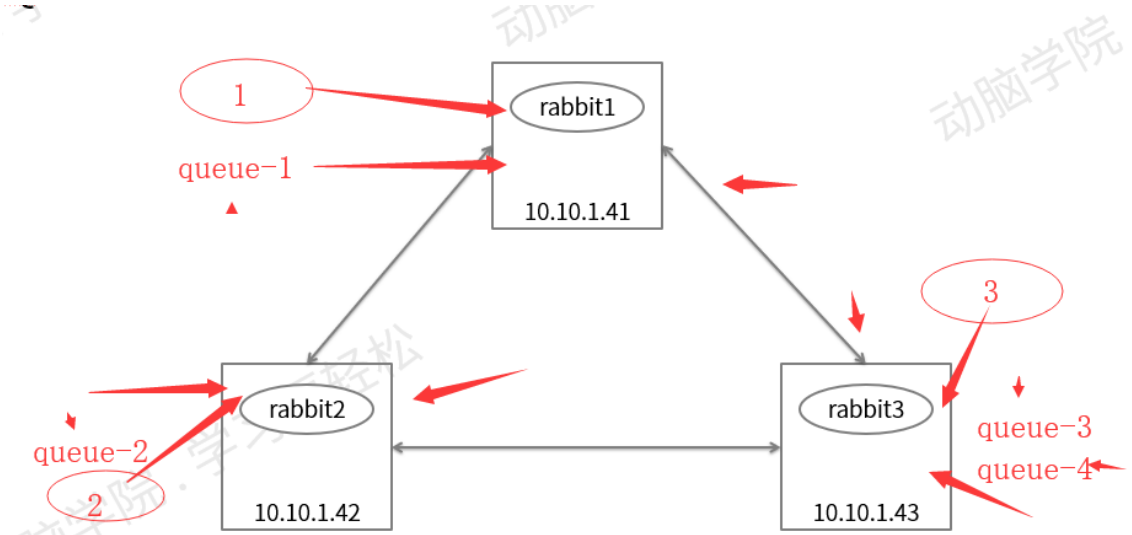
Cluster

1、负载均衡能不能做到？ 高并发 大量链接

能 客户端建立连接时是从节点地址列表中随机选择一个。

2、是分布式的吗？ 大量消息，是否是分布在不同的节点上进行存储处理的

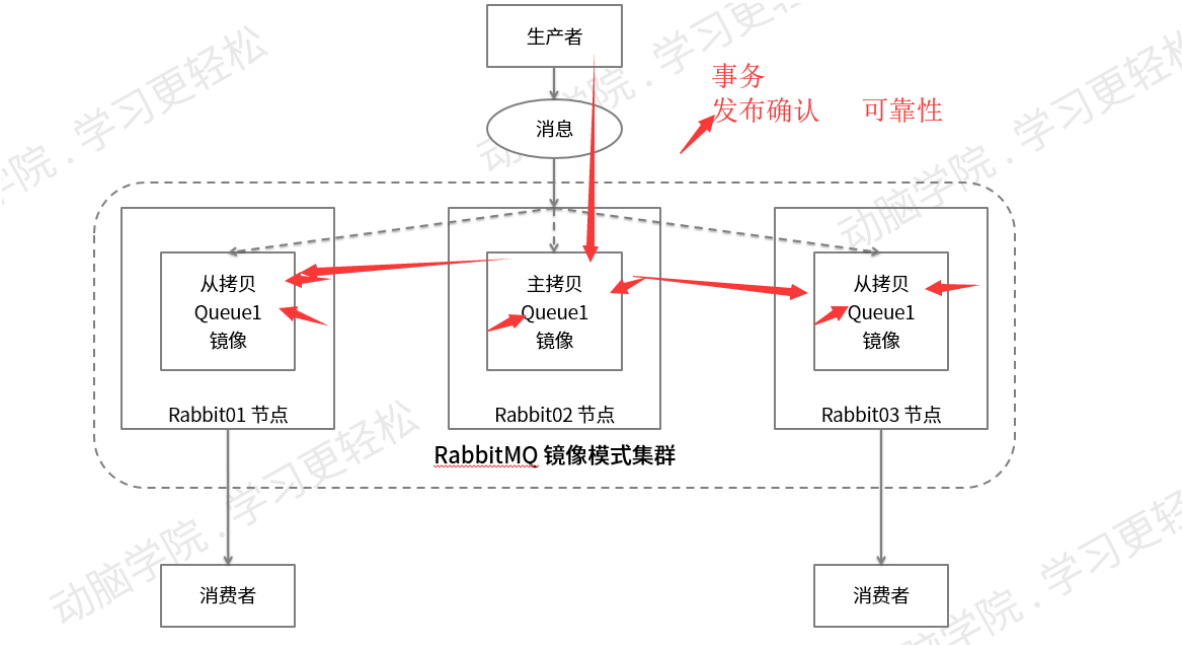
是的



Cluster : 集群提升了可用性、队列消息数据安全性 、并发客户端连接数。

高并发、海量数据（消息）

镜像队列



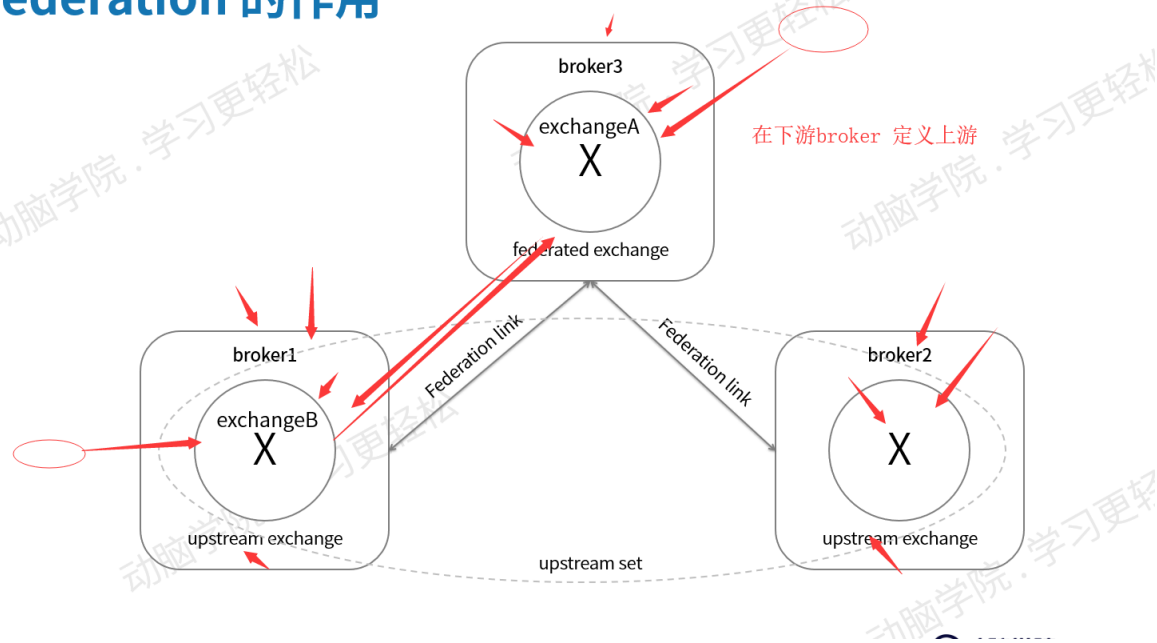
镜像 数量多少合适？ N / 2 + 1 大多数 3

CAP CP A

Federation

联合

Federation 的作用



1、开启插件支持 上游下游都得开。 cluster 和 cluster 之间也可连接 （cluster 中每个节点都必须开启联合支持）

```
rabbitmq-plugins enable rabbitmq_federation      #开启 federation 插件
rabbitmq-plugins enable rabbitmq_federation_management  #开启 federation 管理控制台插件
```

2、在下游broker 上定义上游

命令行、http api 、管理控制台

```
rabbitmqctl set_parameter federation-upstream my-upstream \
'{"uri":"amqp://admin:admin@192.168.120.89"}'
```

3、定义一个policy 说明下游哪些交换器、队列需要建立联合

Shovel

在两个彼此独立的Broker之间建立消息的联通

1、启用插件 在哪个broker上定义铲子就在哪里启用Shovel插件

```
rabbitmq-plugins enable rabbitmq_shovel                # 启动shovel插件
rabbitmq-plugins enable rabbitmq_shovel_management  # 启用shovel管理插件
```

2、配置Shovel

两种配置方式：

静态配置： 在配置文件中配 /etc/rabbitmq/advanced.conf

动态配置：通过运行参数来配置 实时生效。

Shovel 的工作流程

- 建立到源到目标的连接 AMQP
- 消费源的消息
- 重发布到目标