图灵:楼兰

你的神秘技术宝藏

这一章节作为课程收尾,给大家讨论一些在生产使用时需要注意的一些问题。

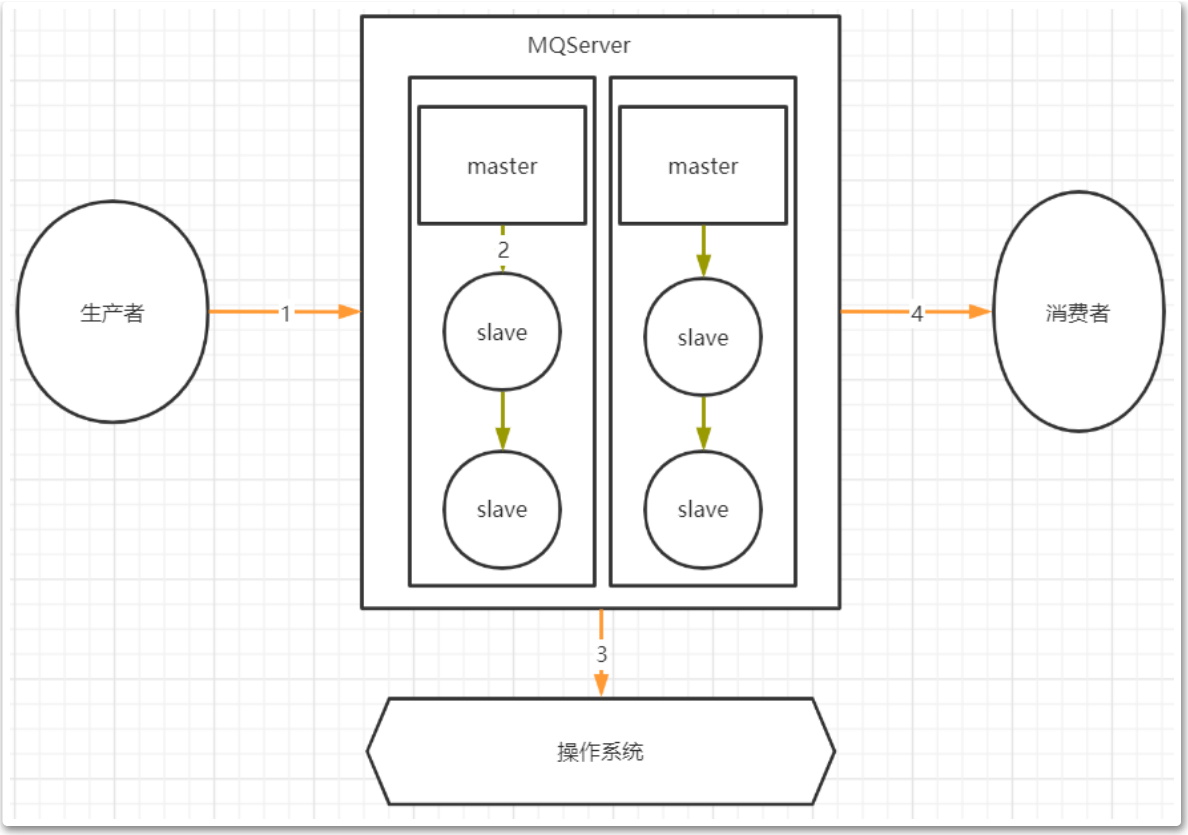
一、RabbitMQ如何保证消息不丢失?

这是面试时最喜欢问的问题,其实这是个所有MQ的一个共性的问题,大致的解决思路也是差不多的,但是针对不同的MQ产品会有不同的解决方案。而RabbitMQ设计之处就是针对企业内部系统之间进行调用设计的,所以他的消息可靠性是比较高的。

> 千万不要再回答 手动确认了

1、哪些环节会有丢消息的可能?

我们考虑一个通用的MQ场景:



其中，1，2，4三个场景都是跨网络的，而跨网络就肯定会有丢消息的可能。

然后关于3这个环节，通常MQ存盘时都会先写入操作系统的缓存page cache中，然后再由操作系统异步的将消息写入硬盘。这个中间有个时间差，就可能会造成消息丢失。如果服务挂了，缓存中还没有来得及写入硬盘的消息就会丢失。这也是任何用户态的应用程序无法避免的。

对于任何MQ产品，都应该从这四个方面来考虑数据的安全性。那我们看看用RabbitMQ时要如何解决这个问题。

2、RabbitMQ消息零丢失方案：

1》生产者保证消息正确发送到RibbitMQ

对于单个数据，可以使用生产者确认机制。通过多次确认的方式，保证生产者的消息能够正确的发送到RabbitMQ中。

RabbitMQ的生产者确认机制分为同步确认和异步确认。同步确认主要是通过在生产者端使用Channel.waitForConfirmsOrDie()指定一个等待确认的完成时间。异步确认机制则是通过channel.addConfirmListener(ConfirmCallback var1, ConfirmCallback var2)在生产者端注入两个回调确认函数。第一个函数是在生产者发送消息时调用，第二个函数则是生产者收到Broker的消息确认请求时调用。两个函数需要通过sequenceNumber自行完成消息的前后对应。sequenceNumber的

生成方式需要通过channel的序列获取。int sequenceNumber = channel.getNextPublishSeqNo();

在RabbitMQ中，另外还有一种手动事务的方式，可以保证消息正确发送。

手动事务机制主要有几个关键的方法： channel.txSelect() 开启事务； channel.txCommit() 提交事务； channel.txRollback() 回滚事务； 用这几个方法来进行事务管理。但是这种方式需要手动控制事务逻辑，并且手动事务会对channel产生阻塞，造成吞吐量下降

2》 RabbitMQ消息存盘不丢消息

这个在RabbitMQ中比较好处理，对于Classic经典队列，直接将队列声明成为持久化队列即可。而新增的Quorum队列和Stream队列，都是明显的持久化队列，能更好的保证服务端消息不会丢失。

3》 RabbitMQ 主从消息同步时不丢消息

这涉及到RabbitMQ的集群架构。首先他的普通集群模式，消息是分散存储的，不会主动进行消息同步了，是有可能丢失消息的。而镜像模式集群，数据会主动在集群各个节点当中同步，这时丢失消息的概率不会太高。

另外，启用Federation联邦机制，给包含重要消息的队列建立一个远端备份，也是一个不错的选择。

4》 RabbitMQ消费者不丢失消息

RabbitMQ在消费消息时可以指定是自动应答，还是手动应答。如果是自动应答模式，消费者会在完成业务处理后自动进行应答，而如果消费者的业务逻辑抛出异常，RabbitMQ会将消息进行重试，这样是不会丢失消息的，但是有可能会造成消息一直重复消费。

将RabbitMQ的应答模式设定为手动应答可以提高消息消费的可靠性。

```
channel.basicConsume(queueName, false, new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
                                       BasicProperties properties, byte[] body)
                    throws IOException {
                long deliveryTag = envelope.getDeliveryTag();
                channel.basicAck(deliveryTag, false);
            }
        });
channel.basicConsume(queueName, true, myconsumer);
```

另外这个应答模式在SpringBoot集成案例中，也可以在配置文件中通过属性spring.rabbitmq.listener.simple.acknowledge-mode 进行指定。可以设定为AUTO 自动应答； MANUAL 手动应答；NONE 不应答； 其中这个NONE不应答，就是不启动应答机制，RabbitMQ只管往消费者推送消息后，就不再重复推送消息了，相当于RocketMQ的sendoneway， 这样效率更高，但是显然会有丢消息的可能。

最后，任何用户态的应用程序都无法保证绝对的数据安全，所以，备份与恢复的方案也需要考虑到。

二、如何保证消息幂等?

1、RabbitMQ的自动重试功能：

当消费者消费消息处理业务逻辑时，如果抛出异常，或者不向RabbitMQ返回响应，默认情况下，RabbitMQ会无限次数的重复进行消息消费。

处理幂等问题，**首先要设定RabbitMQ的重试次数**。在SpringBoot集成RabbitMQ时，可以在配置文件中指定spring.rabbitmq.listener.simple.retry开头的一系列属性，来制定重试策略。

然后，需要在业务上处理幂等问题。

处理幂等问题的关键是要给每个消息一个唯一的标识。

在SpringBoot框架集成RabbitMQ后，可以给每个消息指定一个全局唯一的MessageID，在消费者端针对MessageID做幂等性判断。关键代码：

```
//发送者指定ID字段
Message message2 =
MessageBuilder.withBody(message.getBytes()).setMessageId(UUID.randomUUID().
toString()).build();
        rabbitTemplate.send(message2);
//消费者获取MessageID，自己做幂等性判断
@RabbitListener(queues = "fanout_email_queue")
public void process(Message message) throws Exception {
    // 获取消息Id
    String messageId = message.getMessageProperties().getMessageId();
    ...
}
```

> 要注意下这里用的message要是
> org.springframework.amqp.core.Message

在原生API当中，也是支持MessageId的。当然，在实际工作中，最好还是能够添加一个具有业务意义的数据作为唯一键会更好，这样能更好的防止重复消费问题对业务的影响。比如，针对订单消息，那就用订单ID来做唯一键。在RabbitMQ中，消息的头部就是一个很好的携带数据的地方。

```
// ==== 发送消息时，携带sequenceNumber和orderNo
AMQP.BasicProperties.Builder builder = new AMQP.BasicProperties.Builder();
builder.deliveryMode(MessageProperties.PERSISTENT_TEXT_PLAIN.getDeliveryMod
e());
builder.priority(MessageProperties.PERSISTENT_TEXT_PLAIN.getPriority());
//携带消息ID
builder.messageId(""+channel.getNextPublishSeqNo());
Map<String, Object> headers = new HashMap<>();
//携带订单号
headers.put("order", "123");
builder.headers(headers);
channel.basicPublish("", QUEUE_NAME, builder.build(),
message.getBytes("UTF-8"));

// ==== 接收消息时，拿到sequenceNumber
Consumer myconsumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope
envelope,
                    BasicProperties properties, byte[] body)
                    throws IOException {
                //获取消息ID
                System.out.println("messageId:"+properties.getMessageId());
                //获取订单ID
```

```
                properties.getHeaders().forEach((key,value)->
System.out.println("key: "+key +"; value: "+value));
                // (process the message components here ...)
                //消息处理完后，进行答复。答复过的消息，服务器就不会再次转发。
                //没有答复过的消息，服务器会一直不停转发。
                channel.basicAck(deliveryTag, false);
            }
        };
channel.basicConsume(QUEUE_NAME, false, myconsumer);
```

三、如何保证消息的顺序?

某些场景下，需要保证消息的消费顺序，例如一个下单过程，需要先完成扣款，然后扣减库存，然后通知快递发货，这个顺序不能乱。如果每个步骤都通过消息进行异步通知的话，这一组消息就必须保证他们的消费顺序是一致的。

在RabbitMQ当中，针对消息顺序的设计其实是比较弱的。唯一比较好的策略就是 单队列+单消息推送。即一组有序消息，只发到一个队列中，利用队列的FIFO特性保证消息在队列内顺序不会乱。但是，显然，这是以极度消耗性能作为代价的，在实际适应过程中，应该尽量避免这种场景。

然后在消费者进行消费时，保证只有一个消费者，同时指定prefetch属性为1，即每次RabbitMQ都只往客户端推送一个消息。像这样：

```
spring.rabbitmq.listener.simple.prefetch=1
```

而在多队列情况下，如何保证消息的顺序性，目前使用RabbitMQ的话，还没有比较好的解决方案。在使用时，应该尽量避免这种情况。

四、关于RabbitMQ的数据堆积问题

RabbitMQ一直以来都有一个缺点，就是对于消息堆积问题的处理不好。当RabbitMQ中有大量消息堆积时，整体性能会严重下降。而目前新推出的Quorum队列以及Stream队列，目的就在于解决这个核心问题。但是这两种队列的稳定性和周边生态都还不够完善，因此，在使用RabbitMQ时，还是要非常注意消息堆积的问题。尽量让消息的消费速度和生产速度保持一致。

而如果确实出现了消息堆积比较严重的场景，就需要从数据流转的各个环节综合考虑，设计适合的解决方案。

首先在消息生产者端：

对于生产者端，最明显的方式自然是降低消息生产的速度。但是，生产者端产生消息的速度通常是跟业务息息相关的，一般情况下不太好直接优化。但是可以选择尽量多采用批量消息的方式，降低IO频率。

然后在RabbitMQ服务端：

从前面的分享中也能看出，RabbitMQ本身其实也在着力于提高服务端的消息堆积能力。对于消息堆积严重的队列，可以预先添加懒加载机制，或者创建Sharding分片队列，这些措施都有助于优化服务端的消息堆积能力。另外，尝试使用Stream队列，也能很好的提高服务端的消息堆积能力。

接下来在消息消费者端：

要提升消费速度最直接的方式，就是增加消费者数量了。尤其当消费端的服务出现问题，已经有大量消息堆积时。这时，可以尽量多的申请机器，部署消费端应用，争取在最短的时间内消费掉积压的消息。但是这种方式需要注意对其他组件的性能压力。

对于单个消费者端，可以通过配置提升消费者端的吞吐量。例如

```
# 单次推送消息数量
spring.rabbitmq.listener.simple.prefetch=1
# 消费者的消费线程数量
spring.rabbitmq.listener.simple.concurrency=5
```

灵活配置这几个参数，能够在一定程度上调整每个消费者实例的吞吐量，减少消息堆积数量。

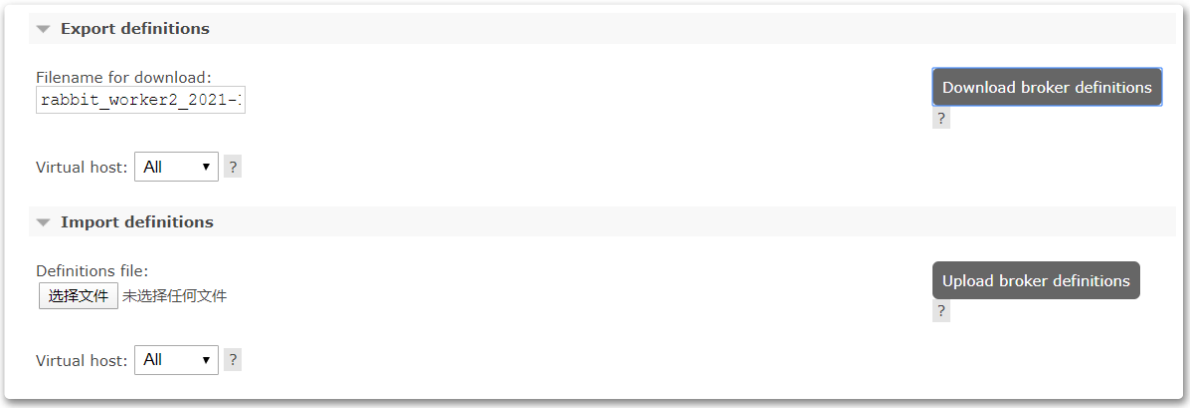
当确实遇到紧急状况，来不及调整消费者端时，可以紧急上线一个消费者组，专门用来将消息快速转录。保存到数据库或者Redis，然后再慢慢进行处理。

五、RabbitMQ的备份与恢复

> 文档地址： https://www.rabbitmq.com/backup.html

RabbitMQ有一个data目录会保存分配到该节点上的所有消息。我们的实验环境中，默认是在`/var/lib/rabbitmq/mnesia`目录下 这个目录里面的备份分为两个部分，一个是元数据(定义结构的数据)，一个是消息存储目录。

对于元数据，可以在Web管理页面通过json文件直接导出或导入。



而对于消息，可以手动进行备份恢复

其实对于消息，由于MQ的特性，是不建议进行备份恢复的。而RabbitMQ如果要进行数据备份恢复，也非常简单。

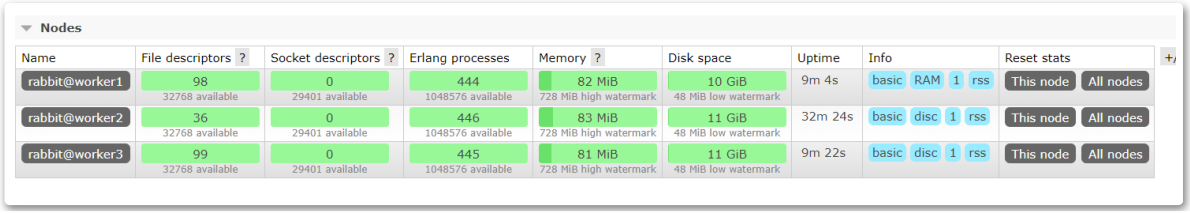
首先，要保证要恢复的RabbitMQ中已经有了全部的元数据，这个可以通过上一步的json文件来恢复。

然后，备份过程必须要先停止应用。如果是针对镜像集群，还需要把整个集群全部停止。

最后，在RabbitMQ的数据目录中，有按virtual hosts组织的文件夹。你只需要按照虚拟主机，将整个文件夹复制到新的服务中即可。持久化消息和非持久化消息都会一起备份。 我们实验环境的默认目录是`/var/lib/rabbitmq/mnesia/rabbit@worker2/msg_stores/vhosts`

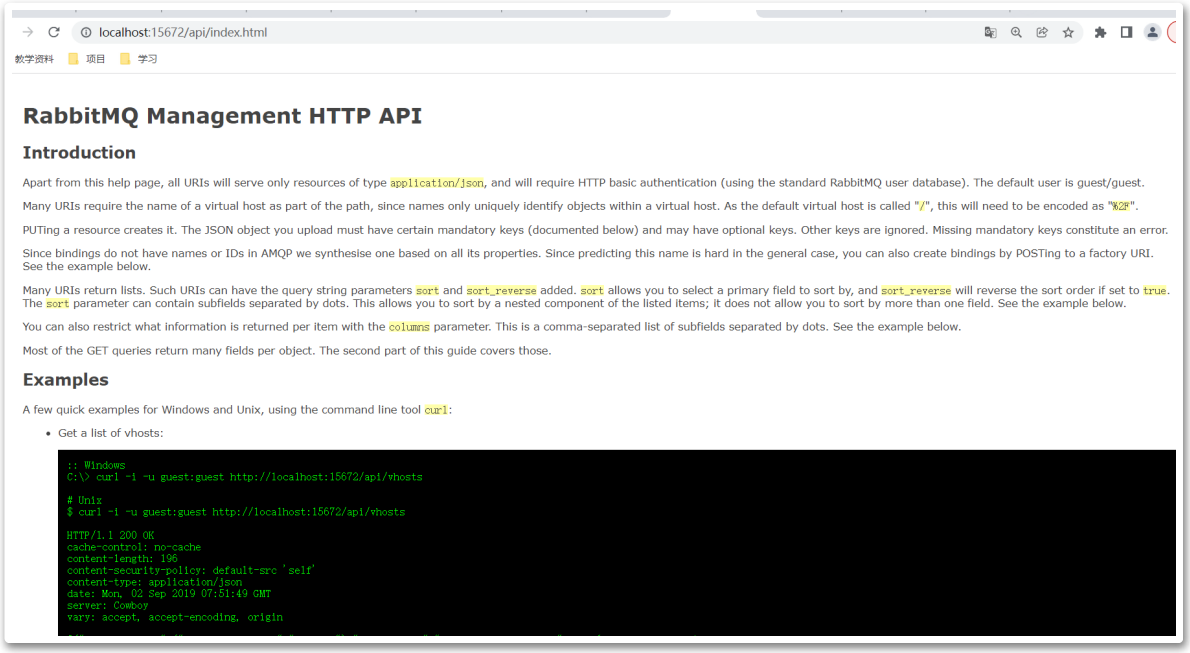
六、RabbitMQ的性能监控

关于RabbitMQ的性能监控，在管理控制台中提供了非常丰富的展示。例如在下面这个简单的集群节点图中，就监控了非常多系统的关键资源。



还包括消息的生产消费频率、关键组件使用情况等等非常多的信息，都可以从这个管理控制台上展现出来。但是，对于构建一个自动化的性能监控系统来说，这个管理页面就不太够用了。为此，RabbitMQ也提供了一系列的HTTP接口，通过这些接口可以非常全面的使用并管理RabbitMQ的各种功能。

这些HTTP的接口不需要专门去查手册，在部署的管理控制台页面下方已经集成了详细的文档，我们只需要打开HTTP API的页面就能看到。



比如最常用的 http://[server:port]/api/overview 接口，会列出非常多的信息，包含系统的资源使用情况。通过这个接口，就可以很好的对接Promethus、Grafana等工具，构建更灵活的监控告警体系。

可以看到，这里面的接口相当丰富，不光可以通过GET请求获取各种消息，还可以通过其他类型的HTTP请求来管理RabbitMQ中的各种资源，因此在实际使用时，还需要考虑这些接口的安全性。

七、搭建HAProxy，实现高可用集群

我们之前搭建的镜像集群，已经具备了集群的功能，请求发送到任何一个节点上，数据都是在集群内共享的。但是，在企业使用时，通常还会选择在集群基础上增加负载均衡的能力。即希望将客户端的请求能够尽量均匀的分配到集群中各个节点上，这样可以让集群的压力得到平衡。

实现负载均衡的方式有很多，HAProxy就是其中一种可选方案。HAProxy是一个免费、快速并且可靠的解决方案，有很多大型互联网公司都在使用。通过HAProxy，应用可以直连一个单一的IP地址，然后HAProxy会将这个IP地址的TCP请求进行转发，并在转发过程中实现负载均衡。

> 很多有实力的大企业会采用F5等其他的一些负载均衡工具。

安装步骤如下：

1、安装HAProxy

```
#安装
yum install haproxy

#检测安装是否成功
haproxy

#查找haproxy.cfg文件的位置
find / -name haproxy.cfg

#配置haproxy.cfg文件 后面会列出参考配置
vim /etc/haproxy/haproxy.cfg

#启动haproxy
haproxy -f /etc/haproxy/haproxy.cfg

#查看haproxy进程状态
systemctl status haproxy.service
#状态如下说明 已经启动成功 Active: active (running)

#访问如下地址对mq节点进行监控
http://47.114.175.29:1080/haproxy_stats

#代码中访问mq集群地址，则变为访问haproxy地址:5672
```

2、配置HAProxy

修改haproxy.cfg文件。下面是参考配置。注意将节点的IP地址和端口换成你自己的环境。

```
#---------------------------------------------------------------------
# Example configuration for a possible web application. See the
# full configuration options online.
```

```
#
# http://haproxy.1wt.eu/download/1.4/doc/configuration.txt
#
#---------------------------------------------------------------------

#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
# to have these messages end up in /var/log/haproxy.log you will
# need to:
#16 # 1) configure syslog to accept network log events. This is done
# by adding the '-r' option to the SYSLOGD_OPTIONS in
# /etc/sysconfig/syslog
#
# 2) configure local2 events to go to the /var/log/haproxy.log
# file. A line like the following can be added to
# /etc/sysconfig/syslog
#
# local2.* /var/log/haproxy.log
#
log 127.0.0.1 local2

chroot /var/lib/haproxy
pidfile /var/run/haproxy.pid
maxconn 4000
user haproxy
group haproxy
daemon

# turn on stats unix socket
stats socket /var/lib/haproxy/stats

#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#---------------------------------------------------------------------
defaults
mode http
log global
option httplog
option dontlognull
option http-server-close
option forwardfor except 127.0.0.0/8
```

```
option redispatch
retries 3
timeout http-request 10s
timeout queue 1m
timeout connect 10s
timeout client 1m
timeout server 1m
timeout http-keep-alive 10s
timeout check 10s
maxconn 300059

#对MQ集群进行监听
listen rabbitmq_cluster
bind 0.0.0.0:5672
option tcplog
mode tcp
option clitcpka
timeout connect 1s
timeout client 10s
timeout server 10s
balance roundrobin
server node1 worker1:5672 check inter 5s rise 2 fall 3
server node2 worker2:5672 check inter 5s rise 2 fall 3
server node3 worker3:5672 check inter 5s rise 2 fall 3

#开启haproxy监控服务
listen http_front
bind 0.0.0.0:1080
stats refresh 30s
stats uri /haproxy_stats
stats auth admin:admin
```

八、总结

基于MQ的事件驱动机制，给庞大的互联网应用带来了不一样的方向。MQ的异步、解耦、削峰三大功能特点在很多业务场景下都能带来极大的性能提升，在日常工作过程中，应该尝试总结这些设计的思想。

虽然MQ的功能，说起来比较简单，但是随着MQ的应用逐渐深化，所需要解决的问题也更深入。对各种细化问题的挖掘程度，很大程度上决定了开发团队能不能真正Hold得住MQ产品。通常面向互联网的应用场景，更加注重MQ的吞吐量，需要将消息尽快的保存下来，再供后端慢慢消费。而针对企业内部的应用场景，更加注重MQ的数据安全性，在复杂多变的业务场景下，每一个消息都需要有更加严格的安全保障。而在当今互联网，Kafka是第一个场景的不二代表，但是他会丢失消息的特

性，让kafka的使用场景比较局限。RabbitMQ作为一个老牌产品，是第二个场景最有力的代表。当然，随着互联网应用不端成熟，也不断有其他更全能的产品冒出来，比如阿里的RocketMQ以及雅虎的Pulsar。但是不管未来MQ领域会是什么样子，RabbitMQ依然是目前企业级最为经典也最为重要的一个产品。他的功能最为全面，周边生态也非常成熟，并且RabbitMQ有庞大的Spring社区支持，本身也在吸收其他产品的各种优点，持续进化，所以未来RabbitMQ的重要性也会更加凸显。

整个课程，从RabbitMQ的安装、应用、扩展等多个方面，综合介绍了RabbitMQ的各种常用使用方法以及业务场景。希望能够带你打开一扇大门，更真实，更深入的理解MQ这个工具。

> 有道云笔记链接
>
> 文档：RabbitMQ4使用中的常见问题.md
> 链接：http://note.youdao.com/noteshare?id=5518647119bd4c4953d6525f69da4b2c&sub=7062028E725A423B8919227C0244D9BB