RabbitMQ快速上手以及核心概念详解

-- 楼兰

这一章节我们将快速搭建RabbitMQ服务，并了解RabbitMQ的核心工作机制。

一、MQ介绍

1、什么是MQ，有什么用

MQ即MessageQueue，消息对列。我们这次要学习的RabbitMQ就是一种典型的MQ产品。

那么到底什么是MQ呢？可以分两个部分来理解：消息Message：在不同应用程序之间传递的数据。队列Queue，一种FIFO 先进先出的数据结构。**将消息以队列的形式存储起来，并且在不同的应用程序之间进行传递，这就成了MessageQueue**。

MQ产品最直接的作用，是将同步的事件驱动改为异步的消息驱动。这话什么意思？我们从一个最常见的SpringBoot应用开始说起。

首先搭建一个普通的Maven项目，在pom.xml中引入SpringBoot的依赖：

```
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
   <version>2.4.5</version>
        </dependency>
    </dependencies>
```

然后增加一个监听器类

```
public class MyApplicationListener implements ApplicationListener<ApplicationEvent> {
    @Override
    public void onApplicationEvent(ApplicationEvent applicationEvent) {
        System.out.println("=====> MyApplicationListener: "+applicationEvent);
    }
}
```

接下来，添加一个SpringBoot启动类。在启动类中加入自己的这个监听器。

```
@SpringBootApplication
public class AppDemo implements CommandLineRunner {
    public static void main(String[] args) {
        SpringApplication application = new SpringApplication(AppDemo.class);
        application.addListeners(new MyApplicationListener());
        application.run(args);
    }

    @Resource
    private ApplicationContext applicationContext;
    @Override
    public void run(String... args) throws Exception {
        applicationContext.publishEvent(new ApplicationEvent("myEvent"){
        });
    }
}
```

好了。不用添加配置文件，直接启动就行。 然后可以看到这样的结果：

```
>>>>>>> MyApplicationListener >>>>>>: org.springframework.boot.context.event.ApplicationStartingEvent[source=org.springframework.boot.SpringApplicat
>>>>>>> MyApplicationListener >>>>>>: org.springframework.boot.context.event.ApplicationEnvironmentPreparedEvent[source=org.springframework.boot.Spr

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.14)

>>>>>>> MyApplicationListener >>>>>>: org.springframework.boot.context.event.ApplicationContextInitializedEvent[source=org.springframework.boot.Spri
2024-07-25 20:32:13.630  INFO 5127 --- [           main] com.roy.AppDemo                          : Starting AppDemo using Java 1.8.0_371 on royking
2024-07-25 20:32:13.631  INFO 5127 --- [           main] com.roy.AppDemo                          : No active profile set, falling back to 1 default
>>>>>>> MyApplicationListener >>>>>>: org.springframework.boot.context.event.ApplicationPreparedEvent[source=org.springframework.boot.SpringApplicat
>>>>>>> MyApplicationListener >>>>>>: org.springframework.context.event.ContextRefreshedEvent[source=org.springframework.context.annotation.Annotati
2024-07-25 20:32:14.124  INFO 5127 --- [           main] com.roy.AppDemo                          : Started AppDemo in 0.799 seconds (JVM running fo
>>>>>>> MyApplicationListener >>>>>>: org.springframework.boot.context.event.ApplicationStartedEvent[source=org.springframework.boot.SpringApplicati
>>>>>>> MyApplicationListener >>>>>>: org.springframework.boot.availability.AvailabilityChangeEvent[source=org.springframework.context.annotation.An
>>>>>>> MyApplicationListener >>>>>>: com.roy.AppDemo$1[source=myEvent]
>>>>>>> MyApplicationListener >>>>>>: org.springframework.boot.context.event.ApplicationReadyEvent[source=org.springframework.boot.SpringApplication
>>>>>>> MyApplicationListener >>>>>>: org.springframework.boot.availability.AvailabilityChangeEvent[source=org.springframework.context.annotation.An
>>>>>>> MyApplicationListener >>>>>>: org.springframework.context.event.ContextClosedEvent[source=org.springframework.context.annotation.AnnotationC
```

SpringBoot发布的事件

自己发布的事件

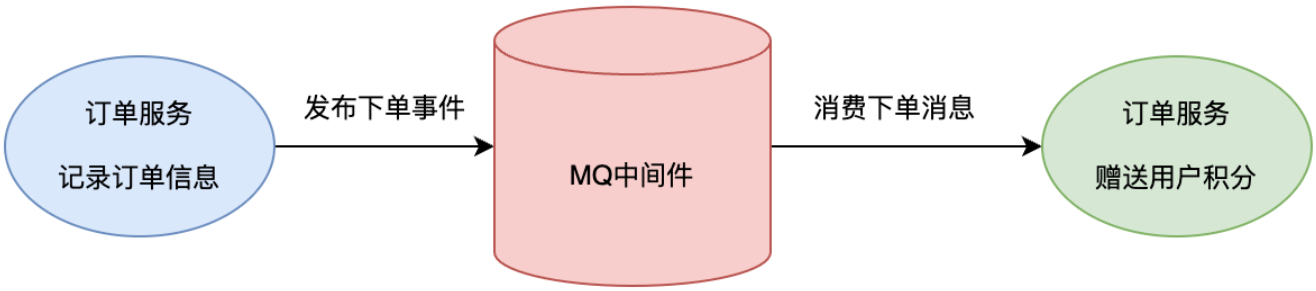
从这个示例看到，SpringBoot框架其实在启动时，就会尝试发布各种ApplicationEvent事件，表示自己启动到了哪个步骤。这时，SpringBoot框架就可以称为消息生产者Producer。同样的，只要有ApplicationEvent事件发布了，就会触发MyApplicationListener监听器，处理这些事件。MyApplicationListener就可以成为消息消费者Consumer。

Producer和Consumer他们的运行状况互不干涉，不管有没有Consumer，Producer一样会发布消息。反过来，不管Producer有没有发布消息，Consumer也一样会监听这些事件。这种方式，实际上就是通过事件中包含的消息在驱动Producer和Consumer工作，这种工作方式也就称为消息驱动。

> 与消息驱动形成对比的是常见的事件驱动。比如经常写的Controller，只有通过一个事件主动触发，才会调用。

从这个简单的例子可以看到，SpringBoot内部就集成了这种消息驱动的机制。但是，这些Producer和Consumer都只能在一个进程中使用。如果需要跨进行进行调用呢？这就需要独立一个中间服务，才能发布和接受这些消息。而这个中间服务，就是MQ中间件。

比如在一个大型电商项目中，订单服务完成下单，就可以发布下单事件，而下游的消费者，就可以消费这个下单事件，进行一些补充的业务。



在这个业务过程中，MQ中间件应该要起到什么作用呢?

- **解耦**：Producer和Consumer都只跟中间件进行交互，而不需要互相进行交互。这意味着，在Producer发送消息时，不需要考虑有没有Consumer或者有多少个Consumer。反之亦然。甚至，即便Producer和Consumer是用不同语言开发的，只要都能够与MQ中间件正常交互，那么他们就可以通过MQ中间件进行消息传递。
- **异步**：消息并不是从Producer发送出来后，就立即交由Consumer处理，而是在MQ中间件中暂存下来。等到Consumer启动后，自行去MQ中间件上处理。也就是说，错开了Producer发送消息和Consumer消费消息的时间。
- **削峰**：有了MQ做消息暂存，那么当Producer发送消息的速度与Consumer处理消息的速度不一致时，MQ就能起到削峰填谷的作用。

2、主流MQ产品对比

在MQ长期发展过程中，诞生了很多MQ产品，但是有很多MQ产品都已经逐渐被淘汰了。比如早期的ZeroMQ,ActiveMQ等。目前最常用的MQ产品包括Kafka、RabbitMQ和RocketMQ。我们对这三个产品做下简单的比较，重点需要理解他们的适用场景。

	优点	缺点	适用场景
Kafka	吞吐量非常大，性能非常好，技术生态完整	功能比较单一	分布式日志收集，大数据采集
RabbitMQ	消息可靠性高，功能全面	吞吐量较低。消息积压会影响性能。 erlang语言比较小众	企业内部系统调用
RocketMQ	高吞吐、高性能、高可用，高级功能非常全	技术生态相对没有那么完整	几乎全场景。尤其适合金融场景

> 好的产品都是在不断演进的，所以对这些产品的理解也需要与时俱进。比如现在还有个MQ产品Pulsar，非常适合于大型企业内部海量的系统调用，也体现了非常强大的竞争力。

二、RabbitMQ快速上手

1、RabbitMQ产品介绍

RabbitMQ的历史可以追随到2005年，他是一个非常老牌的MQ产品，使用非常广泛。同时期的很多MQ产品都已经逐渐被业界淘汰了，比如2003年诞生的ActiveMQ，2012年诞生的ZeroMQ，但是RabbitMQ却依然稳稳占据一席之地，足可见他的经典。官网地址 https://www.rabbitmq.com/ 。

目前最新的官网是这样介绍的：

Why RabbitMQ?

RabbitMQ is a reliable and mature messaging and streaming broker, which is easy to deploy on cloud environments, on-premises, and on your local machine. It is currently used by millions worldwide.

Interoperable

RabbitMQ supports several open standard protocols, including AMQP 1.0 and MQTT 5. There are multiple client libraries available, which can be used with your programming language of choice, just pick one. No vendor lock-in!

Flexible

RabbitMQ provides many options you can combine to define how your messages go from the publisher to one or many consumers. Routing, filtering, streaming, federation, and so on, you name it.

Reliable

With the ability to acknowledge message delivery and to replicate messages across a cluster, you can ensure your messages are safe with RabbitMQ.

> 最新的3.13版本官网做了一次大改版，由此可见RabbitMQ产品的开发活力依然非常强劲。

2、安装RabbitMQ

1、前置环境

我们这次选择的RabbitMQ版本是目前最新的3.13版本。其实就RabbitMQ最近的几个版本，核心的Quorum Queue 和 Stream Queue功能早在3.9.x版本就已经成型了。后续的版本主要是对这两个核心功能做一些修复以及增强，同时增加了很多新的功能插件。

RabbitMQ是基于Erlang语言开发的，所以安装RabbitMQ之前需要安装Erlang语言环境。需要注意下的是RabbitMQ与Erlang语言之间是有版本对应关系的。目前3.13版本的RabbitMQ需要Erlang语言版本26.0到26.2.x之间。

RabbitMQ version	Minimum required Erlang/OTP	Maximum supported Erlang/OTP	Notes
• 3.13.6 • 3.13.5 • 3.13.4 • 3.13.3 • 3.13.2 • 3.13.1 • 3.13.0	• 26.0	• 26.2.x	• The 3.13 release series is compatible with Erlang 26. • OpenSSL 3 support in Erlang is considered to be mature and ready for production use. • Erlang 26.1 and later versions supports FIPS mode on OpenSSL 3
• 3.12.13 • 3.12.12 • 3.12.11 • 3.12.10	• 25.0	• 26.2.x	• The 3.12 release series is compatible with Erlang 26. • OpenSSL 3 support in Erlang is considered to be mature enough for production. • Erlang 26.1 and later versions supports FIPS mode on OpenSSL 3
• 3.12.9 • 3.12.8 • 3.12.7 • 3.12.6 • 3.12.5	• 25.0	• 26.1.x	• The 3.12 release series is compatible with Erlang 26. • OpenSSL 3 support in Erlang is considered to be mature enough for production. • Erlang 26.1 supports FIPS mode on OpenSSL 3

需要先从官网下载操作系统对应的RabbitMQ安装包以及Erlang语言的安装包。

2、安装RabbitMQ服务

RabbitMQ服务有多种安装方式。但是在学习阶段，建议大家使用CentOS手动进行安装。这样更能接触产品的细节。之后使用其他操作系统或者使用Docker等技术安装时，才会更顺利。

需要注意的是，当前版本的RabbitMQ建议CentOS版本最好升级到CentOS9版本。至少不能低于CentOS8。

Erlang语言包的安装，建议使用RabbitMQ提供的zero dependency版本。下载地址：https://github.com/rabbitmq/erlang-rpm/releases

```
[root@192-168-65-112 ~]# rpm -ivh erlang-26.2.5.2-1.el9.x86_64.rpm
警告：erlang-26.2.5.2-1.el9.x86_64.rpm: 头V4 RSA/SHA256 Signature, 密钥 ID 6026dfca: NOKEY
Verifying...                          ################################# [100%]
准备中...                          ################################# [100%]
正在升级/安装...
   1:erlang-26.2.5.2-1.el9            ################################# [100%]

[root@192-168-65-112 ~]# erl -version
Erlang (SMP,ASYNC_THREADS) (BEAM) emulator version 14.2.5.2
```

接下来安装RabbitMQ。这里我们采用RPM安装包的方式。安装包下载地址：https://github.com/rabbitmq/rabbitmq-server/releases 。 这里我们下载无依赖版本： rabbitmq-server-3.13.6-1.el8.noarch.rpm

```
[root@192-168-65-112 ~]# rpm -ivh rabbitmq-server-3.13.6-1.el8.noarch.rpm
警告：rabbitmq-server-3.13.6-1.el8.noarch.rpm: 头V4 RSA/SHA512 Signature, 密钥 ID 6026dfca: NOKEY
Verifying...                          ################################# [100%]
准备中...                          ################################# [100%]
正在升级/安装...
   1:rabbitmq-server-3.13.6-1.el8     ################################# [100%]
/usr/lib/tmpfiles.d/rabbitmq-server.conf:1: Line references path below legacy directory /var/run/, updating /var/run/rabbitmq →
/run/rabbitmq; please update the tmpfiles.d/ drop-in file accordingly.
```

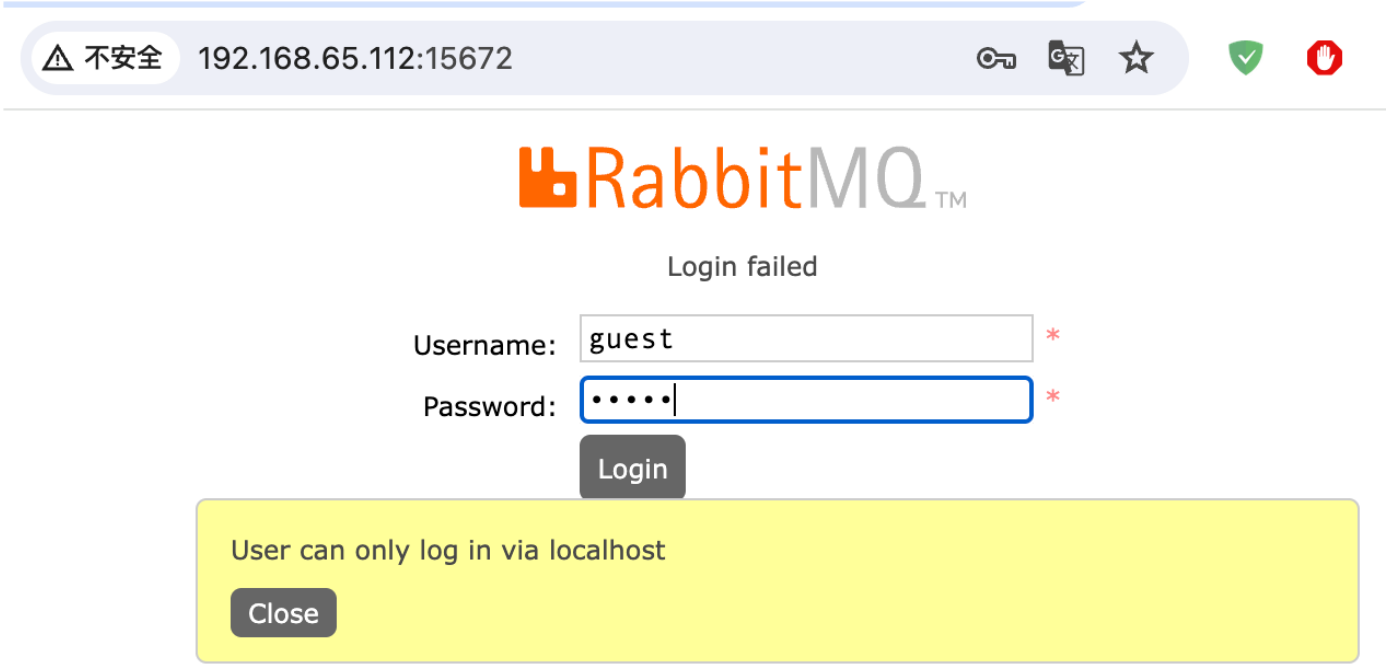
安装完成后，可以使用几个常用的指令维护RabbitMQ的服务状态。

```
service rabbitmq-server start --启动Rabbitmq服务。启动应用之前要先启动服务。

rabbitmq-server -deched  --后台启动RabbitMQ应用

rabbitmqctl start_app  --启动Rabbitmq

rabbitmqctl stop  --关闭Rabbitmq

rabbitmqctl status -- 查看RabbitMQ服务状态。
 出现Status为Runtime表示启动成功。
```

默认情况下， RabbitMQ只是一个后台服务，不便于管理。而RabbitMQ提供了管理插件，可以使用图形化的方式管理RabbitMQ。

```
[root@192-168-65-112 ~]# rabbitmq-plugins enable rabbitmq_management
Enabling plugins on node rabbit@192-168-65-112:
rabbitmq_management
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@192-168-65-112...
The following plugins have been enabled:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch

set 3 plugins.
Offline change; changes will take effect at broker restart.

--重启服务后生效
service rabbitmq-server start
rabbitmqctl start_app
```

插件激活后，就可以访问RabbitMQ的Web控制台了。访问端口15672.



RabbitMQ提供了默认的用户名guest，密码guest。但是默认情况下，只允许本地登录，远程访问是无法登录的。

这时，通常都会创建一个管理员账号单独对RabbitMQ进行管理。

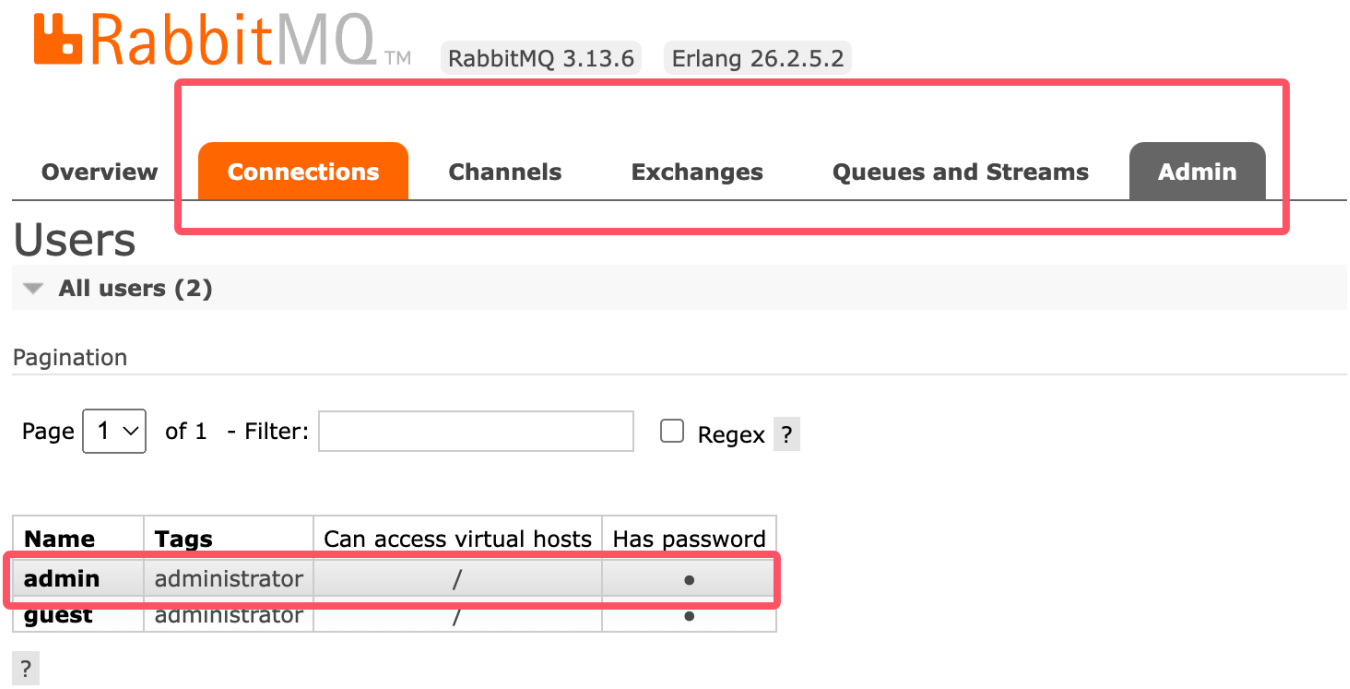
```
[root@192-168-65-112 ~]# rabbitmqctl add_user admin admin
Adding user "admin" ...
Done. Don't forget to grant the user permissions to some virtual hosts! See 'rabbitmqctl help set_permissions' to learn more.
[root@192-168-65-112 ~]# rabbitmqctl set_permissions -p / admin "." "." ".*"
Setting permissions for user "admin" in vhost "/" ...
[root@192-168-65-112 ~]# rabbitmqctl set_user_tags admin administrator
Setting tags for user "admin" to [administrator] ...
```

这样就可以用admin/admin用户登录Web控制台了。

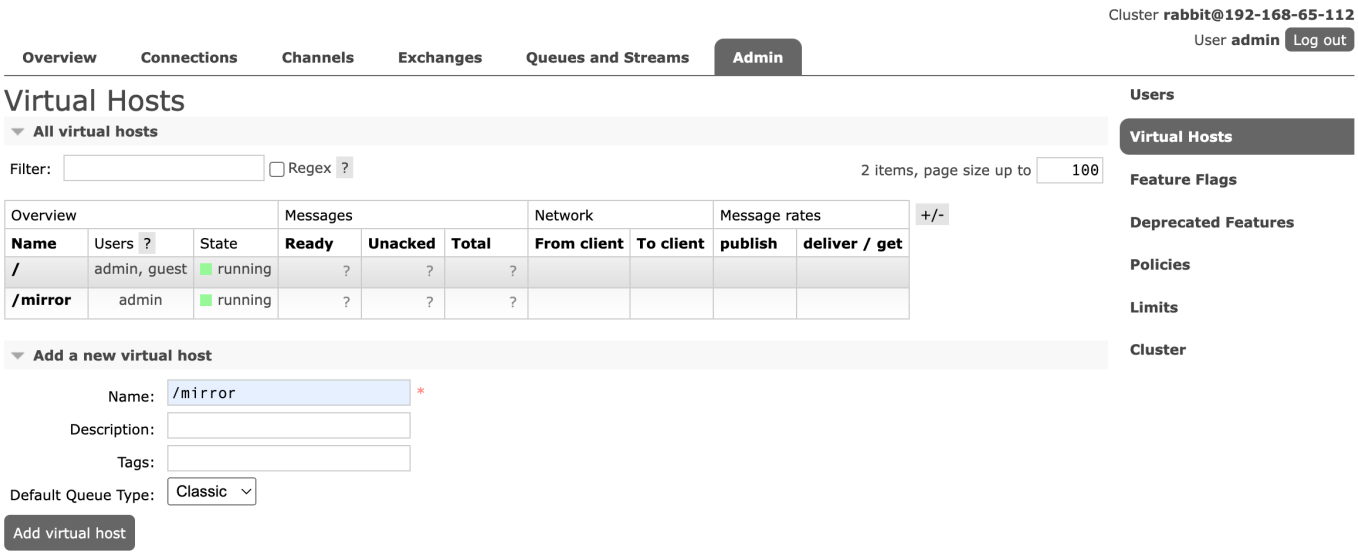
3、RabbitMQ基础使用

登录控制台后上方就能看到RabbitMQ的主要功能。其中Overview是概述，主要展示RabbitMQ服务的一些整体运行情况。后面Conections、Channels、Exchanges和Queues就是RabbitMQ的核心功能。最后的Admin则是一些管理功能。

例如我们之前创建的admin用户，就表现在Admin下的用户信息中。



例如，在Admin管理页面，可以创建一个虚拟机，virtual machine，并配置admin用户拥有访问的权限。



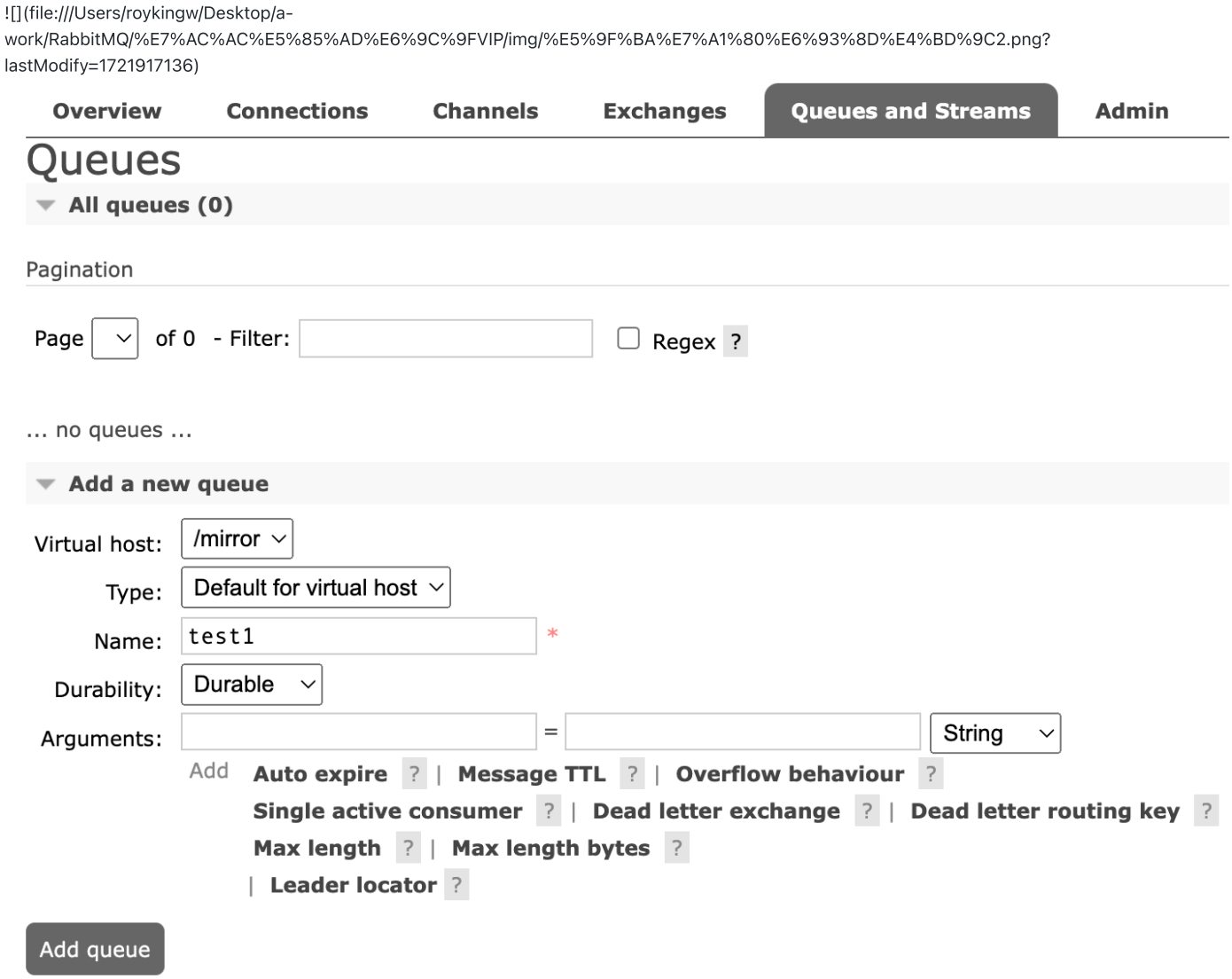
在RabbitMQ中，不同虚拟机之间的资源是完全隔离的。在资源充足的情况下，每个虚拟机可以当成一个独立的RabbitMQ服务来使用。其他管理功能这里就不详细介绍了，后续随着使用深入再做介绍。

接下来我们来上手使用一下RabbitMQ的核心功能。

1、理解Queue

Exchange和Queue是RabbitMQ中用来传递消息的核心组件。我们可以简单体验一下。

1、在Queues菜单，创建一个名为test1的经典对列



创建完成后，选择这个test1队列，就可以在页面上直接发送消息以及消费消息了。

在RabbitMQ中的消息都是通过Queue队列传递的，这个Queue其实就是一个典型的FIFO的队列数据结构。我们当前的演示是通过控制台页面来通过Queue进行收发消息。未来，我们编写客户端时，就是绑定对应的对列进行消息收发。

2、理解Exchange

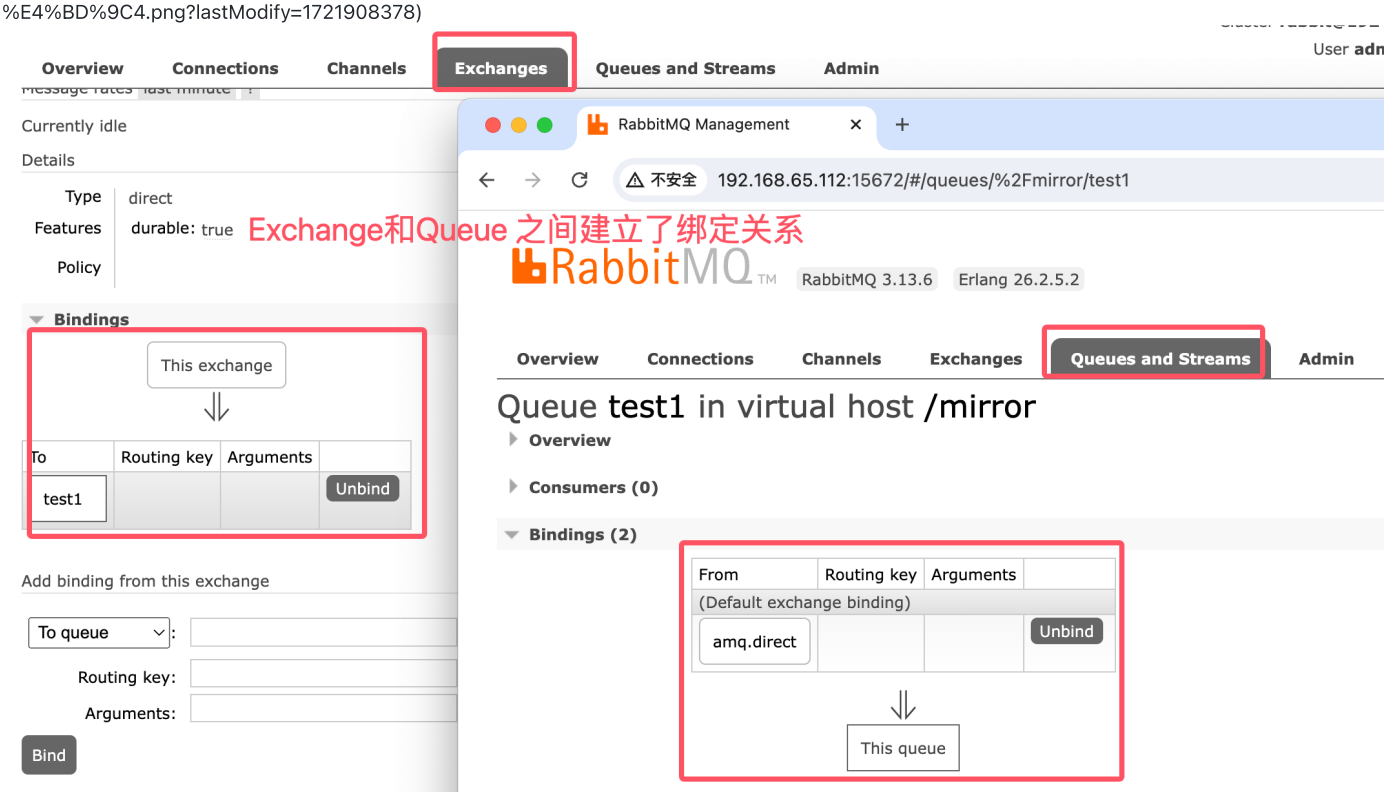
队列Queue即可以发消息，也可以收消息，那旁边的Exchange交换机是干什么的呢？其实他也是用来辅助发送消息的。Exchange与Queue之间会建立一种绑定关系，通过绑定关系，Exchange交换机里发送的消息就可以分发到不同的Queue上。

进入Exchanges菜单，可以看到针对每个虚拟机，RabbitMQ都预先创建了多个Exchange交换机。

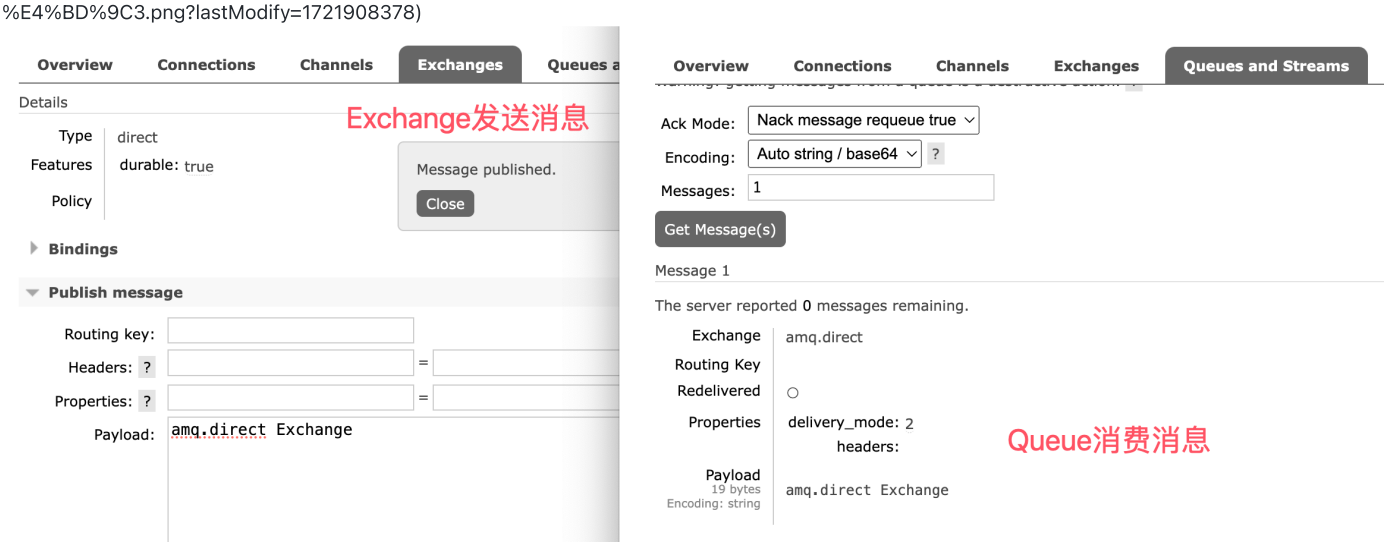
这里我们选择amq.direct交换机，进入交换机详情页，选择Binding，并将test1队列绑定到这个交换机上。

> 注意选择/mirror虚拟机上的Exchange

绑定完成后，可以在Exchange详情页以及Queue详情页都看到绑定的结果。



接下来就可以在Exchange的详情页里发送消息。然后在test1这个queue里就能消费到这条消息。



Exchange交换机并不实际存储消息，只是将发送到Exchange的消息转发到绑定的队列上。在具体使用时，通常只有消息生产者需要与Exchange打交道。而消费者，则并不需要与Exchange打交道，只要从Queue中消费消息就可以了。

另外，Exchange既然可以绑定一个队列，当然也可以绑定多个队列。在实际使用中，Exchange与Queue之间可以建立不同类型的绑定关系，然后通过一些不同的策略，选择将消息转发到哪些Queue上。这时候，Messaage上几个没有用上的参数，像Routing Key ,Headers，Properties这些参数就能派上用场了。

在这个过程中，我们都是通过页面操作完成的消息发送与接收。在实际应用时，其实就是通过RabbitMQ提供的客户端API来完成这些功能。但是整个执行的过程，其实跟页面操作是相同的。

3、理解Connection和Channel

这两个功能实际上是跟客户端应用的对应关系。一个Connection可以理解为一个客户端应用。而一个应用可以创建多个Channel，用来与RabbitMQ进行交互。

我们可以来搭建一个客户端应用了解一下。

1、创建一个Maven项目，在pom.xml中引入RabbitMQ客户端的依赖：

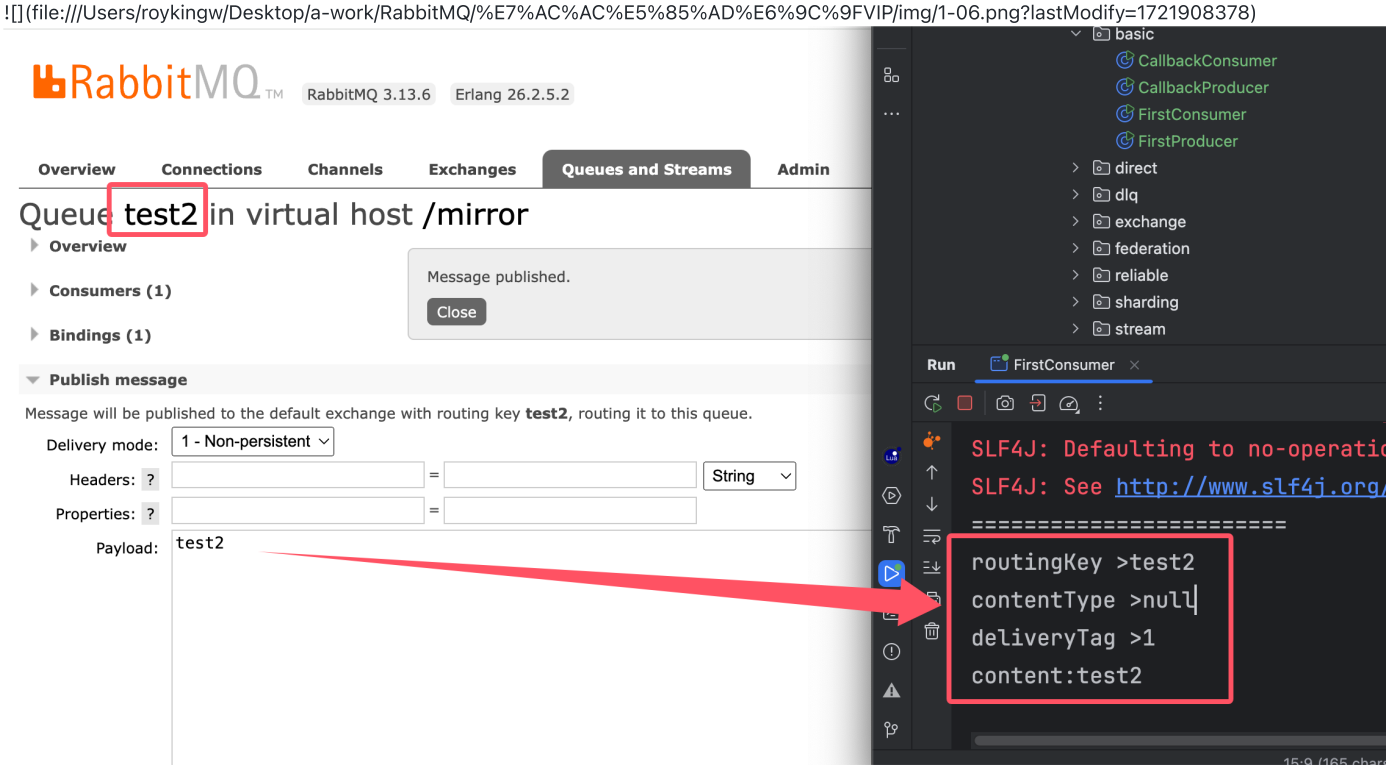
```
  <dependency>
            <groupId>com.rabbitmq</groupId>
            <artifactId>amqp-client</artifactId>
            <version>5.21.0</version>
        </dependency>
```

2、然后就可以创建一个消费者实例，尝试从RabbitMQ上的test1这个队列上拉取消息。

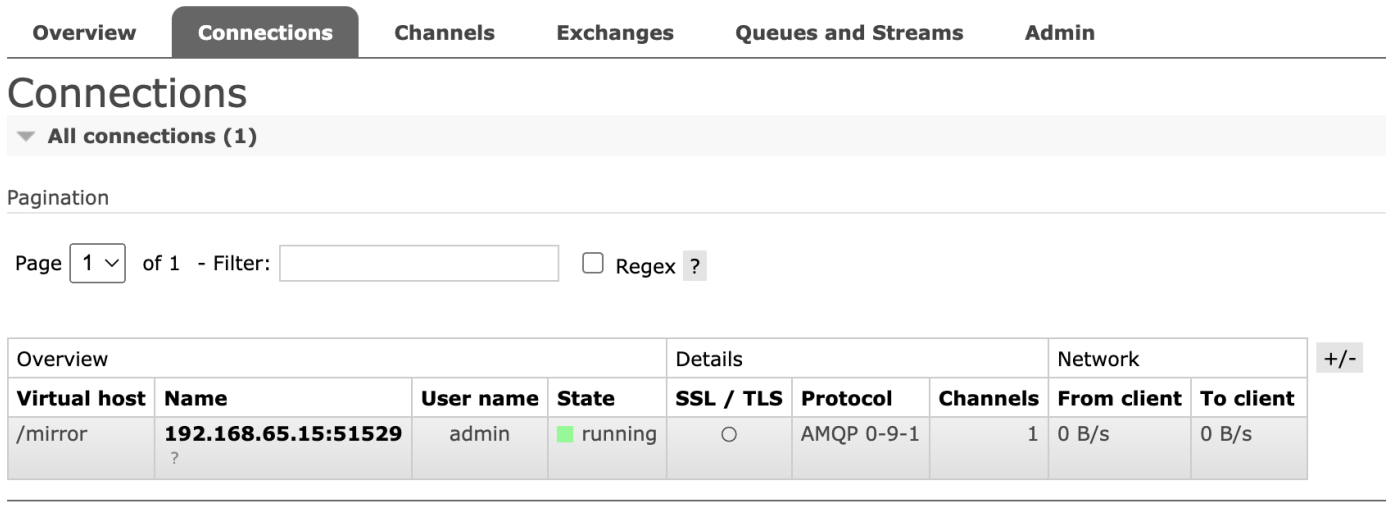
```
public class FirstConsumer {
    private static final String HOST_NAME="192.168.65.112";
    private static final int HOST_PORT=5672;
    private static final String QUEUE_NAME="test2";
    public static final String USER_NAME="admin";
    public static final String PASSWORD="admin";
    public static final String VIRTUAL_HOST="/mirror";

    public static void main(String[] args) throws Exception{
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(HOST_NAME);
        factory.setPort(HOST_PORT);
        factory.setUsername(USER_NAME);
        factory.setPassword(PASSWORD);
        factory.setVirtualHost(VIRTUAL_HOST);
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();
        /**
         * 声明一个对列。几个参数依次为：  队列名，durable是否实例化；exclusive：是否独占；autoDelete：是否自动删除；arguments:参数
         * 这几个参数跟创建队列的页面是一致的。
         * 如果Broker上没有队列，那么就会自动创建队列。
         * 但是如果Broker上已经由了这个队列。那么队列的属性必须匹配，否则会报错。
         */
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);
        //每个worker同时最多只处理一个消息
        channel.basicQos(1);
        //回调函数，处理接收到的消息
        Consumer myconsumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
                                       AMQP.BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("========================");
                String routingKey = envelope.getRoutingKey();
                System.out.println("routingKey >"+routingKey);
                String contentType = properties.getContentType();
                System.out.println("contentType >"+contentType);
                long deliveryTag = envelope.getDeliveryTag();
                System.out.println("deliveryTag >"+deliveryTag);
                System.out.println("content:"+new String(body,"UTF-8"));
                // (process the message components here ...)
                channel.basicAck(deliveryTag, false);
            }
        };
        //从test1队列接收消息
        channel.basicConsume(QUEUE_NAME, myconsumer);
    }
}
```

> 暂时不用过多纠结于实现细节，注意梳理整体实现流程。

执行这个应用程序后，就会在RabbitMQ上新创建一个test2的队列(如果你之前没有创建过的话)，并且启动一个消费者，处理test2队列上的消息。这时，我们可以从管理平台页面上往test2队列发送一条消息，这个消费者程序就会及时消费消息。



然后在管理平台的Connections和Channels里就能看到这个消费者程序与RabbitMQ建立的一个Connection连接与一个Channel通道。

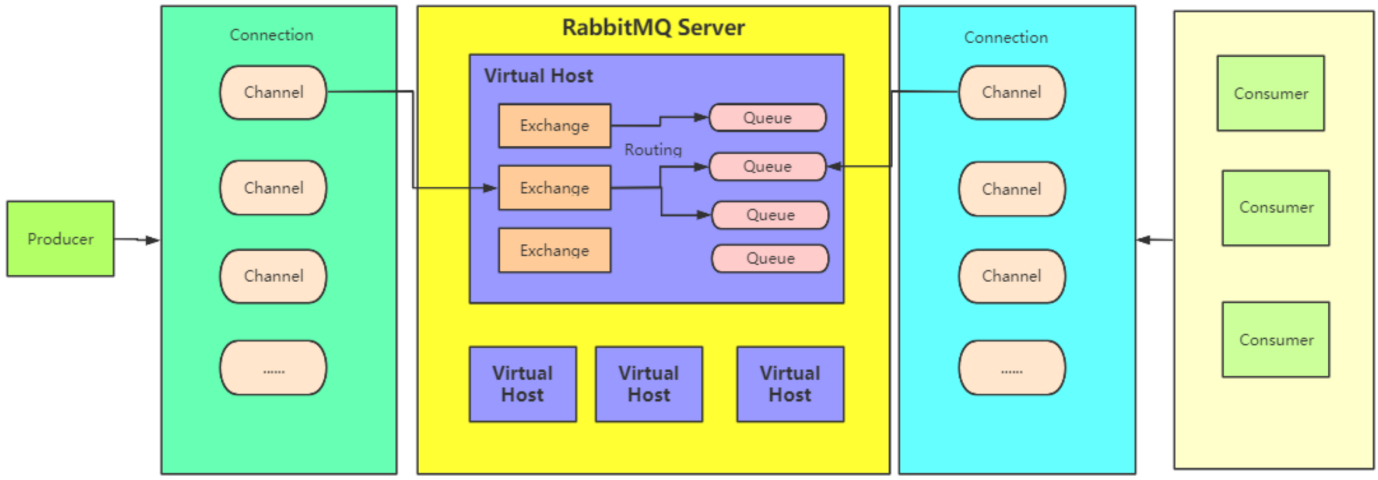


这里可以看到Connection就是与客户端的一个连接。只要连接还通着，他的状态就是running。而Channel是RabbitMQ与客户端进行数据交互的一个通道，没有数据交互时，状态就是idle闲置。有数据交互时，就会变成running。在他们后面，都会展示出数据交互的状态。

另外，从这个简单示例中可以看到，Channel是从Connection中创建出来的，这也意味着，一个Connection中可以创建出多个Channel。从这些Connection和Channel中可以很方面的了解到RabbitMQ当前的服务运行状态。

三、RabbitMQ中的核心概念总结

通过这些操作，我们就可以了解到RabbitMQ的消息流转模型。



这里包含了很多RabbitMQ的重要概念：

1、Queue对列

这是RabbitMQ中最核心的概念。他是实际保存数据的最小单元。Queue结构天生就具有FIFO的顺序。消息最终要被发送到Queue当中，然后才能被消费者进行消费处理。

2、Exchange交换机

这是RabbitMQ中进行数据路由的重要组件。Exchange并不实际保存消息，而是与Queue之间建立绑定关系，然后，如果有消息发送到了Exchange，Exchange就会将消息转发到Queue对列中，从而被对应的消费者消费处理。

在使用RabbitMQ时，Exchange并不是必须的，但是，通常Exchange是与应用开发联系最紧密的。因为RabbitMQ支持的很多业务场景都要Exchange参与。

3、virtual host虚拟主机

RabbitMQ出于服务器复用的想法，可以在一个RabbitMQ集群中划分出多个虚拟主机，每一个虚拟主机都有全套的基础服务组件，可以针对每个虚拟主机进行权限以及数据分配。不同虚拟主机之间是完全隔离的，如果不考虑资源分配的情况，一个虚拟主机就可以当成一个独立的RabbitMQ服务使用。

同时，也意味着不同虚拟主机之间是无法进行通信的，尽管他们是部署在同一个RabbitMQ服务上。例如，你无法通过虚拟机A的Exchange交换机将消息转发到虚拟机B的Queue上。

4、连接 Connection

客户端与RabbitMQ进行交互，首先就需要建立一个TPC连接，这个连接就是Connection。既然是通道，那就需要尽量注意在停止使用时要关闭，释放资源。

5、信道 Channel

一旦客户端与RabbitMQ建立了连接，就会分配一个AMQP信道 Channel。每个信道都会被分配一个唯一的ID。也可以理解为是客户端与RabbitMQ实际进行数据交互的通道，我们后续的大多数的数据操作都是在信道 Channel 这个层面展开的。

RabbitMQ为了减少性能开销，也会在一个Connection中建立多个Channel，这样便于客户端进行多线程连接，这些连接会复用同一个Connection的TCP通道，所以在实际业务中，对于Connection和Channel的分配也需要根据实际情况进行考量。

最后，对照这几个核心概念，尝试去理解下那个看不懂的Java客户端代码，这就是使用RabbitMQ的核心。

> 【有道云笔记】一、快速上手篇.md https://note.youdao.com/s/QJatzZ1t