### 初始MQ

#### 什么是同步通讯：

比如双方打电话，优点：等到妹子恢复再聊下一句；缺点：当其他妹子想和你聊天时，不好意思，进不来。

#### 什么是异步通讯：

比如双方微信聊天，优点：可以同时更N个妹子聊天，时间管理大师；缺点：可能不能立即得到妹子的回复

#### 同步调用的问题

微服务间基于Feign的调用就是同步调用，存在一些问题：

1：耦合高：每次加入新的需求，都要修改原来的代码

2：性能下降：调用者需要等待服务提供者响应，如果调用链过长则响应时间等于每次调用的时间之和；

3：资源浪费：调用链中的每个服务在等待响应过程中，不能释放请求占用的资源，高并发场景下会极度浪费系统资源；

4：级联失败：如果服务提供者出现问题了，所有的调用方都会跟着出问题，如同多米诺骨牌一样，迅速导致整个微服务故障。

总结：

1：同步调用的优点：

时效性较强，可以立即得到结果

2：同步调用的问题：

耦合度高；

性能和吞吐能力下降；

有额外的资源浪费；

有级联失败问题。

#### 异步调用方案

异步调用常见实现就是事件驱动模式：

通过案例说明：用户支付时，会调用支付服务，而支付服务在完成支付以后，后续需要订单服务、仓储服务、短信服务等各自完成对应自己的业务，而现在时异步调用，不能像同步调用一样有支付服务去调用订单服务、仓储服务、短信服务这些服务，所以引入一个概念叫Broker，即事件代理者，在我们这个业务中，一旦有人支付成功，就是一个事件，那么这个事件就会交给Broker去管理，订单服务、仓储服务、短信服务就会去找Broker，当有人支付成功了，Borker要通知订单服务、仓储服务、短信服务这些服务，即时间订阅；而支付服务发现有人支付成功，发布一个事件，Borker就会通知订单服务、仓储服务、短信服务这些服务，订单服务就会去更新订单状态，仓储服务就回去完成库存扣减、发货，短信服务就会去完成短信发送。在整个过程中，支付服务完成事件发布之后就立即结束了自己的业务，可以返回结果给用户了，不需要等待订单服务、仓储服务、短信服务这些服务完成业务，因为支付服务已经把事件发不出去了，Broker负责通知订单服务、仓储服务、短信服务这些服务，支付服务不用管订单服务、仓储服务、短信服务这些服务什么时候执行业务、什么时候执行完业务。

#### 异步调用方案优势：

##### 优势一：服务解耦；

同步调用当有新的需求，比如要新增积分服务，那么要改动支付服务，加上调用积分服务的代码，现在不用了，因为支付服务不负责调用这些服务，而是只发一个订单支付了的事件到Broker，至于谁接收事件，什么时候完成，跟支付服务没关系。所以一旦有新的业务（服务）出现，只需要去订阅Broker事件就可以了，于是就解除了服务与服务之间的耦合了。一样的，当有天不需要短信服务了，以前的同步调用方案需要去支付服务里删除代码，而现在的异步调用方案则只需要让短信服务取消Broker事件，将来支付通知就通知不到短信服务，那么短信就不发了。以上，无论是增加业务还是取消业务，都不需要修改支付服务的代码，于是就接触了服务之间的耦合。

##### 优势二：性能提升，吞吐量提高：

以前的同步调用方案支付服务支付完成后，是要来调用订单服务、仓储服务、短信服务的，每个服务都有哦一定的耗时，因此总耗时就是每个服务的耗时之和；而异步调用方案则是支付服务支付完成后，向Broker发布事件，这个时候，支付服务就可以立即结束告诉用户支付成功了，因为确实是支付成功了，后续的订单、仓储、短信跟支付是没什么关系的，这些动作由Broker去通知订单服务、仓储服务、短信服务去完成，这些动作什么时候完成、耗时多久跟支付服务没关系，只要这些动作最终能做完就行，因此业务总耗时就变成了支付服务里的耗时，耗时缩短了，性能就提升了，吞吐量提高了！

##### 优势三：服务没有强依赖关系，不用担心级联失败问题：

比如，现在仓储服务挂了，跟支付服务没关系，支付服务发完事件就结束了，因此不会出现因为仓储服务挂了导致支付服务跟着挂了的情况，所以级联失败问题也得到解决。同时因为没有强依赖关系，即不在支付服务里面调用其他服务，那么也就不用等待其他服务执行完，那么也就没有了资源浪费的问题，于是资源浪费问题也解决了。

##### 优势四：流量削峰：

比如，现在一秒内有一万个支付请求来到支付服务，订单服务、仓储服务、短信服务每秒只能处理一百个，此时Broker就能起到缓冲的作用，就好像洪水来了有个大坝拦住了，然后订单服务、仓储服务、短信服务能处理多少请求就处理多少请求，处理完了，再从Broker中取，这样订单服务、仓储服务、短信服务处理业务的速度是一直按照自己的能力来的，能处理几个就处理几个，压力都由Broker扛着，于是一个高度的并发就被砍平了：短时间内的海量请求变成较长时间处理完，起到了保护微服务的作用。

##### 总结：

异步通信优点：

1：耦合度低；

2：吞吐量提升；

3：故障隔离；

4：流量削峰

以上，会发现异步通信过程中，所有东西都依赖于Broker去实现，因此，一旦Broker挂了，那么整个微服务也完蛋了；而且流量削峰的过程中，高并发的流量用Broker去缓存，微服务“慢慢”从Broker中去取，因此，如果Broker的高并发能力不行，扛不住这么海量的高并发流量，就像大坝修的不好，洪水来了，可能冲毁大坝。

以上，所以对Broker的可用性、并发能力要求都很高！

异步通信缺点：

1：依赖于Broker的可靠性、安全性、吞吐能力；

2：架构复杂了，业务没有明显的流程线，不好追踪管理

#### 什么时候使用同步？什么时候使用异步？

事实上，大多数情况下都会去使用同步，因为大多数情况下对并发并没有很高的要求，相反对失效性要求很高，比如我调用你去查询一个信息，而这个信息又立马在接下来的业务中用到，必须要用同步调用。

而如果你不需要等调用的结果，同时对吞吐量的要求、并发的要求高，还希望接触服务间的耦合，那么就应该用异步方案。

#### 异步通讯的实现：MQ

##### 什么是MQ

MQ（Message Queue）消息队列，字面上看就是存放消息的队列，也就是上面说的事件驱动架构中的Broker。消息就是事件。

##### 常见的MQ：

**RabbitMQ：**基于Erlang语言 可用性高 吞吐一般 消息延迟微秒级 消息可靠性高

**ActiveMQ：**基于Java语言 可用性一般 吞吐量最低 消息延迟毫秒级 消息可靠性一般

**RocketMQ：**基于Java语言 可用性一般 吞吐量高 消息延迟毫秒级 消息可靠性高

**Kafka：**基于Scala&Java 可用性高 吞吐量最高 消息延迟低毫秒级 消息可靠性一般

#### RabbitMQ安装：

RabbitMQ是基于Erlang语言开发的开源消息通信中间件，因为Erlang是面向并发的编程语言，天生就是为了分布式系统设计的，而即然RabbitMQ是基于Erlang开发，那么自然就具备了这些特征，同时，对消息可靠性、稳定性、高可用非常优异。

##### Docker安装RabbitM：

1：拉取镜像：docker pull rabbitmq:3-management

2：安装MQ：--hostname mq1 是配置主机名，为了以后集群部署，单机不配也没问题；开放了两个端口，15672是管理平台的端口；5672是做消息通信的端口，也就是说将来发消息、收消息都需要通过这个端口去建立连接

docker run \

-e RABBITMQ\_DEFAULT\_USER=hyh \

-e RABBITMQ\_DEFAULT\_PASS=123456 \

--name mq \

--hostname mq1 \

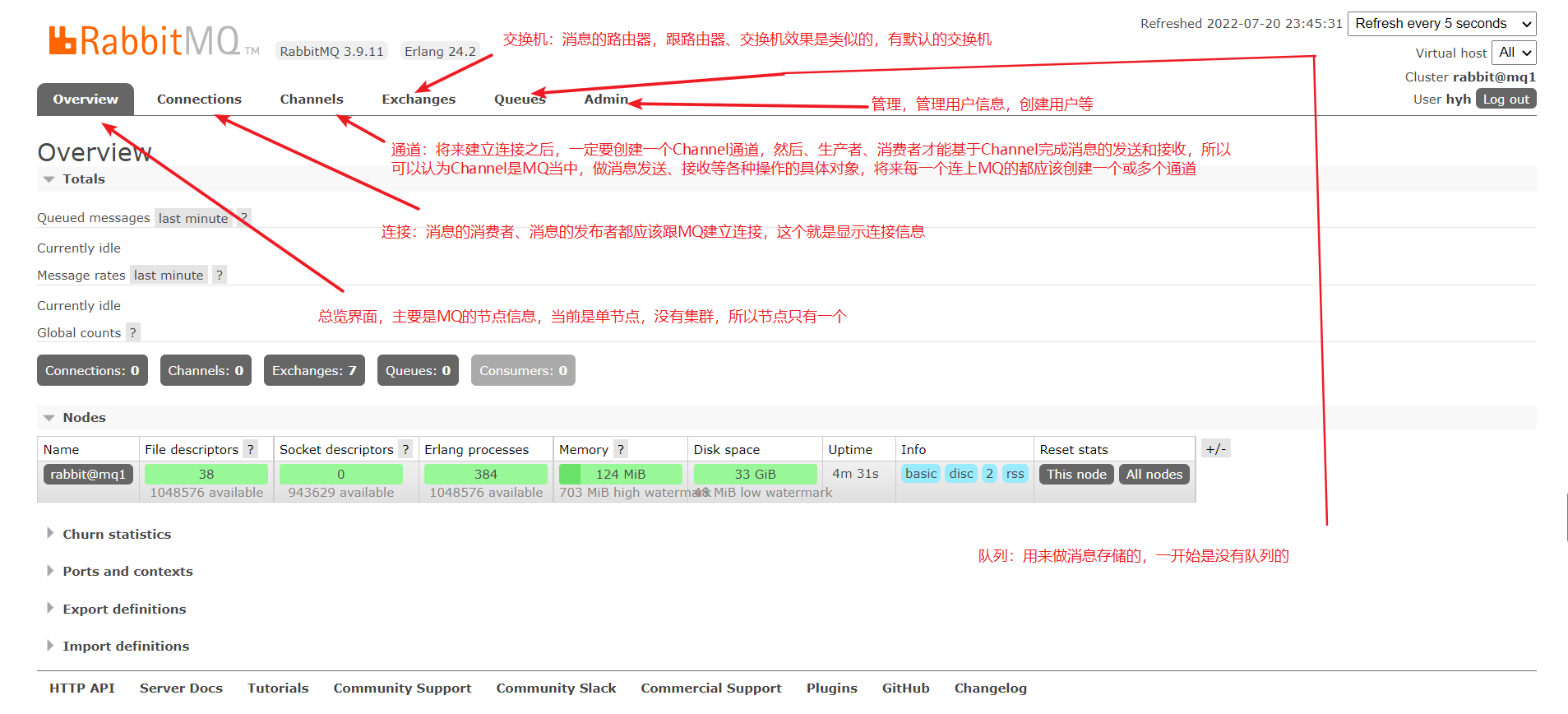
-p 15672:15672 \

-p 5672:5672 \

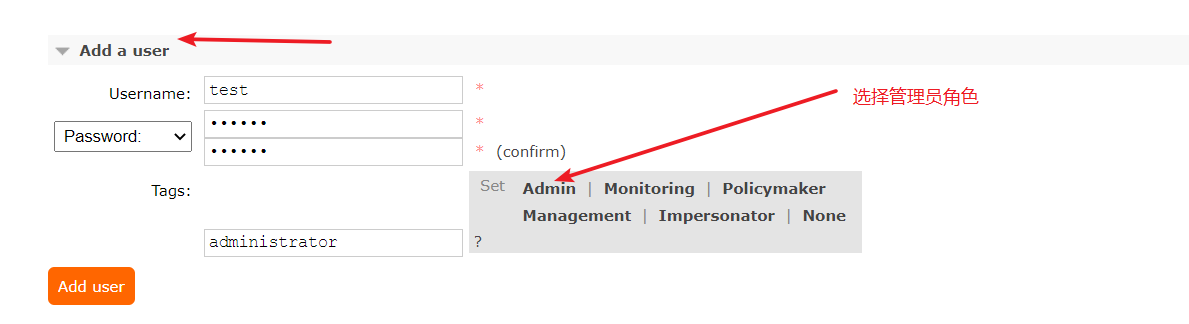
-d \

rabbitmq:3-management

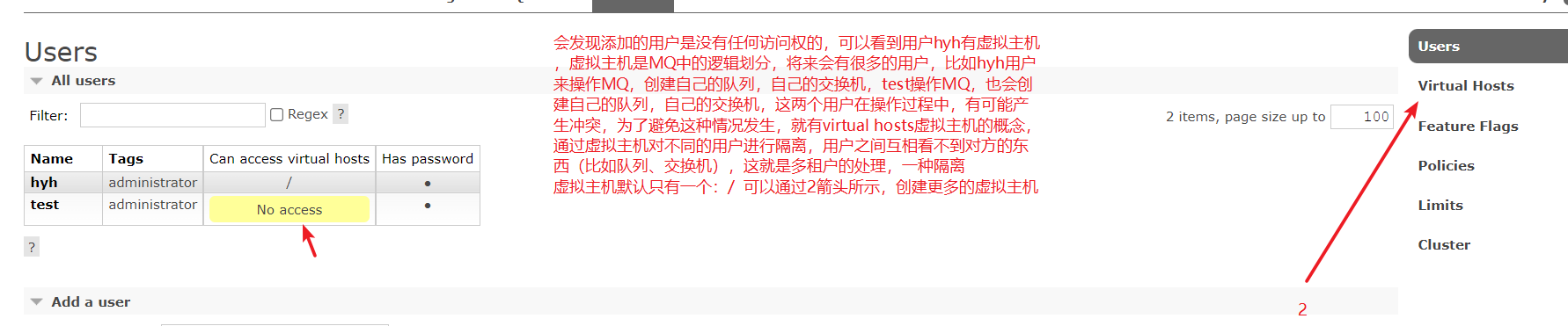
3：访问管理平台：<http://120.77.168.189:15672/> 账号、密码是上面配的：



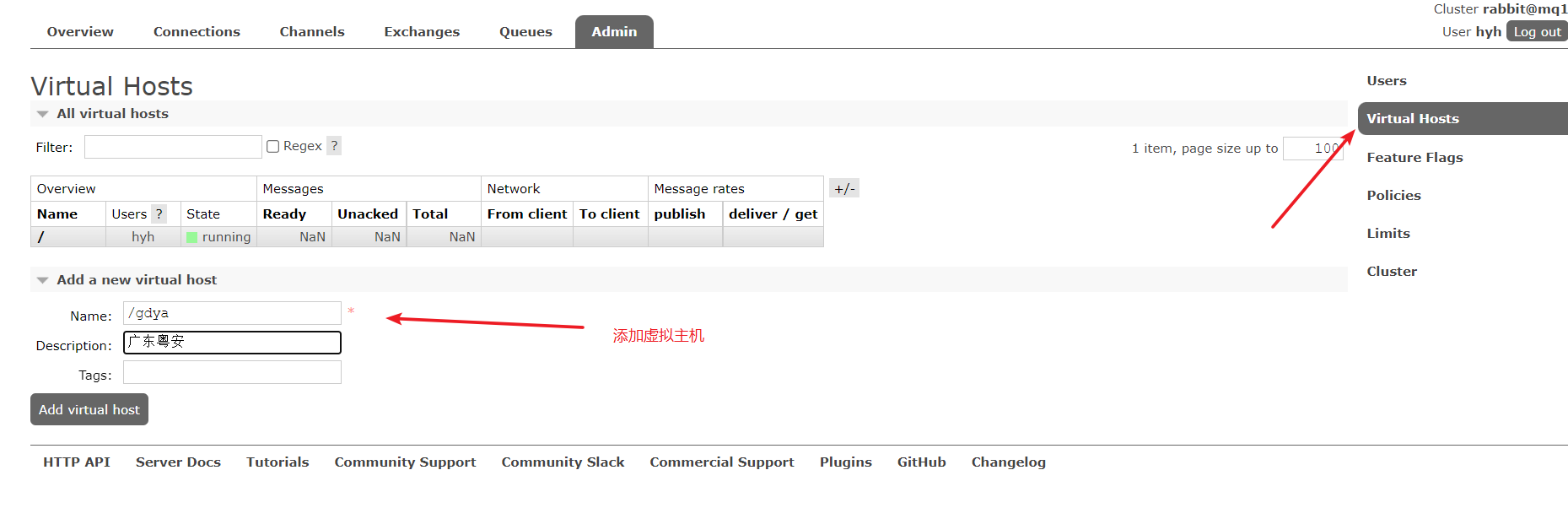
4：添加用户：



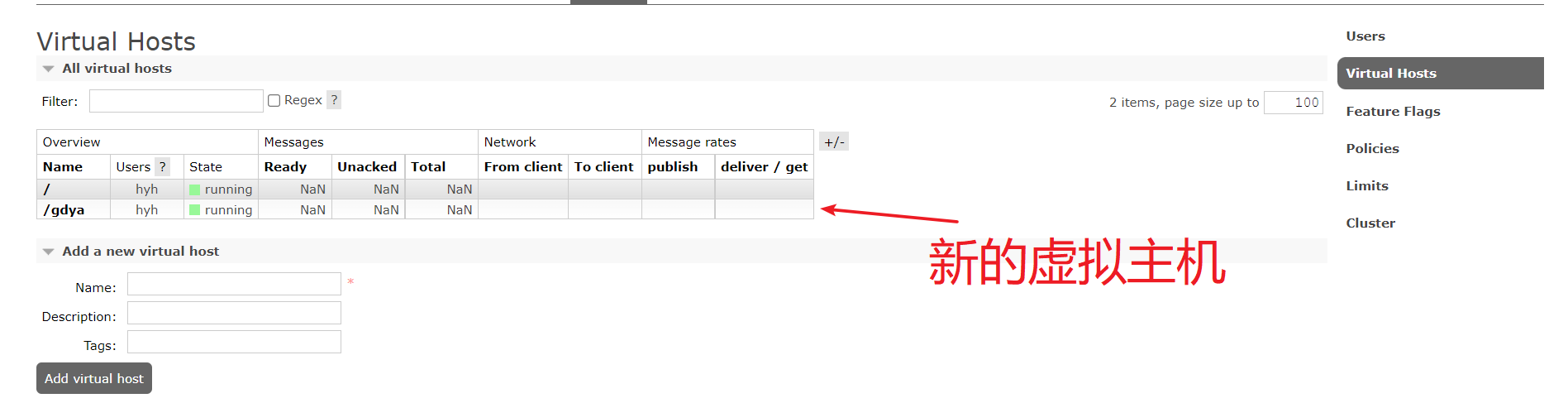
虚拟主机：



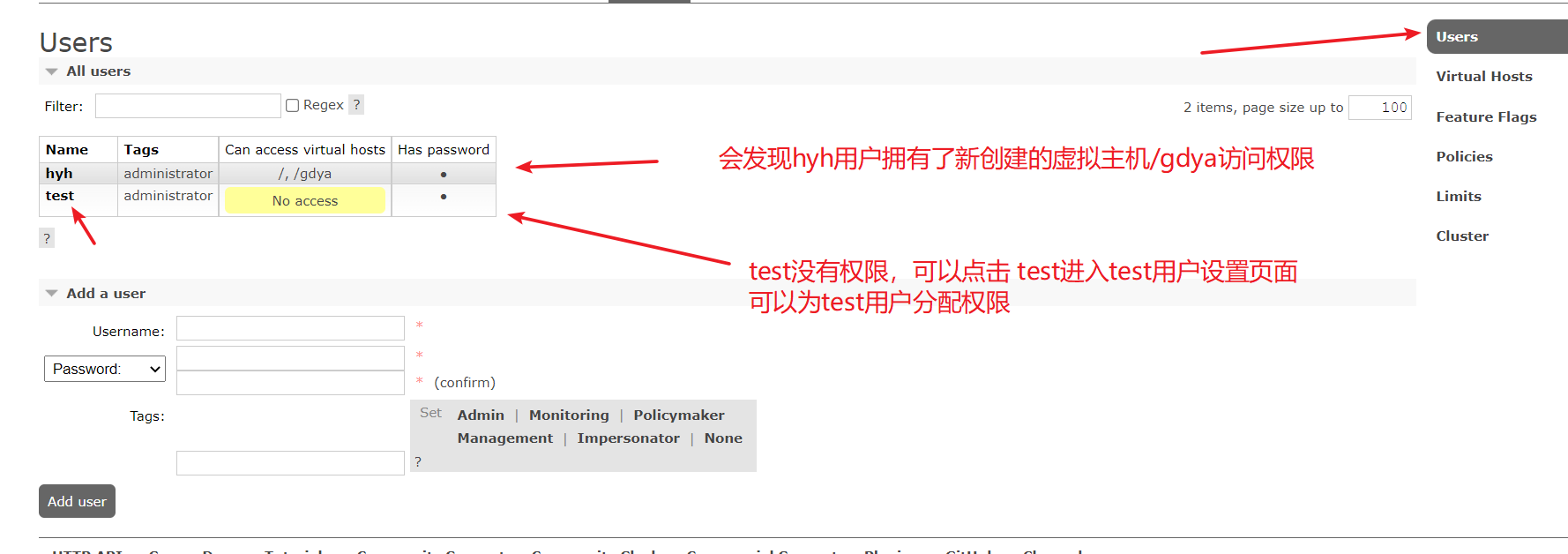
添加虚拟主机：



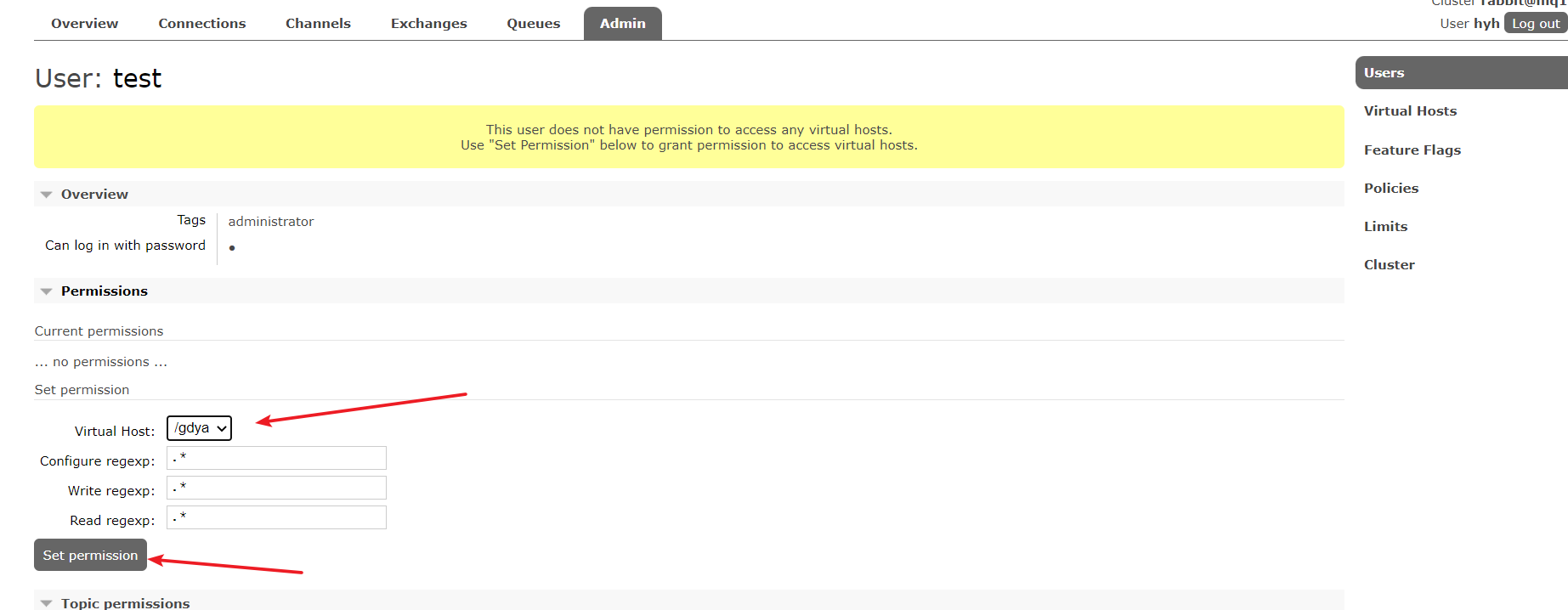
新的虚拟主机：



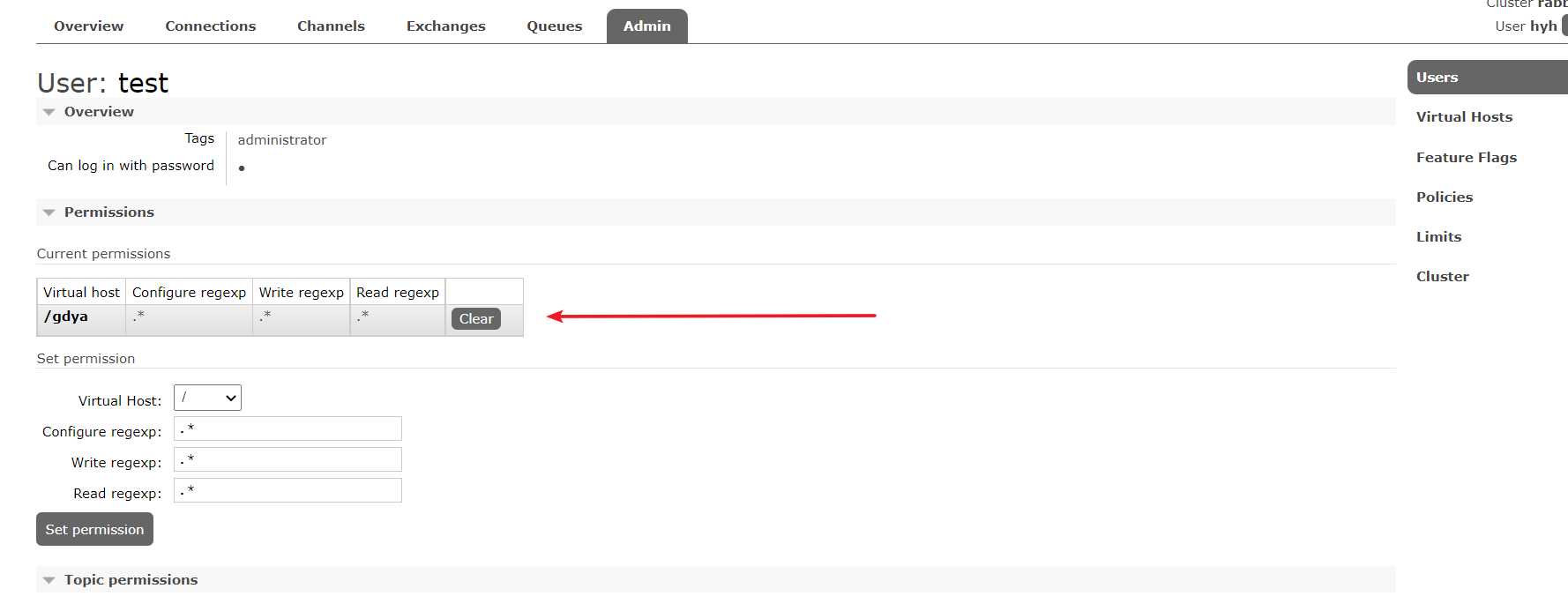
回到User界面：



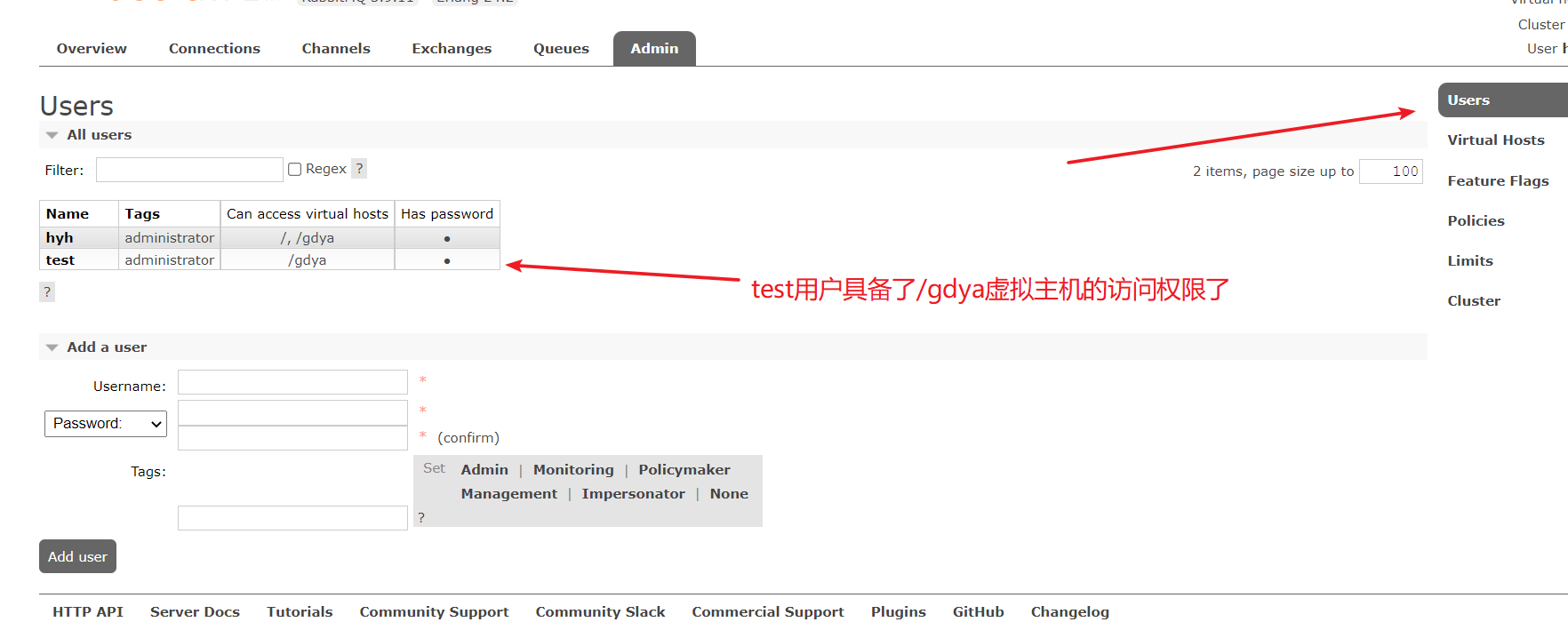
为用户分配虚拟主机访问权限



发现test用户具备了/gdya虚拟主机的访问权限了：



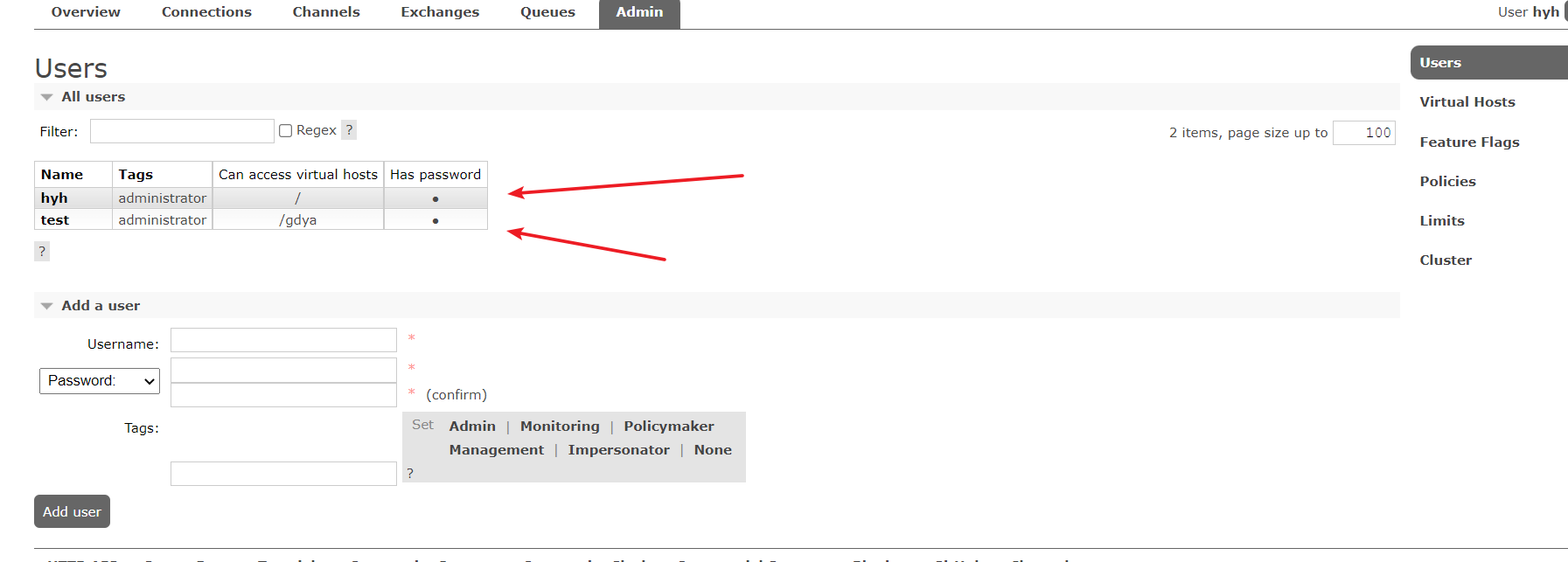
返回User,可以看到test用户具备了/gdya虚拟主机的访问权限了：



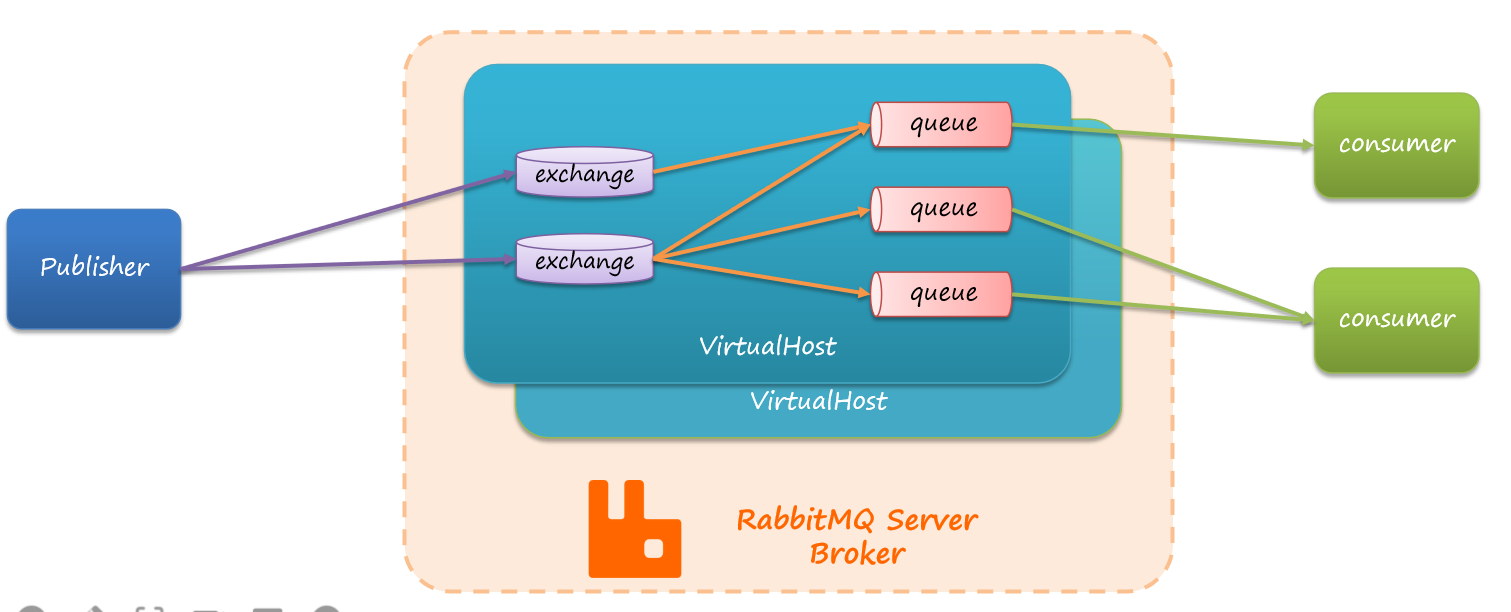
在exchanges查看交换机：



一般情况下，每个用户应该有自己独享的虚拟主机，以隔离不同用户的业务，回到User，把hyh用户的/gdya虚拟主机的访问权限清除：



##### RabbitMQ的结构和概念：



Publisher：消息发送着

Consumer：消息消费者

Publisher会把消息发送到exchange（交换机），交换机负责路由，再把消息投递到Queue（队列），队列负责暂存消息，而后消费者从队列中获取消息，然后处理消息，。

会看到图中有VirtualHost（虚拟主机），将来创建一个用户，用户应该有自己的虚拟主机，各个虚拟主机之间是相互隔离的、看不到的，这样可以避免干扰。

##### 总结：

RabbitMQ中的几个概念：

Channel：操作MQ的工具

Exchange：路有消息到队列中

Queue：缓存消息

Virtual host：虚拟主机，是对queue、exchange等资源的逻辑分组

#### RabbitMQ中五种常见消息模型

MQ的官方文档中给出了5个MQ的Demo实例，1-5是跟消息发送和接收有关的，对应了几种不同的用法：

<https://rabbitmq.com/getstarted.html>



##### 1：基本消息队列（BasicQueue）：

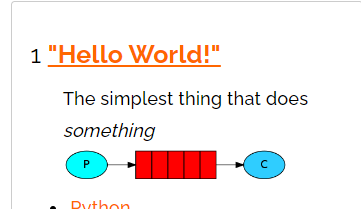
这个队列和下面的工作消息队列一样，消息的发送和接收都是直接基于队列来完成的，而并没有出现我们上面在架构中出现的交换机，不是一个完整的消息驱动模型。如图所示：

基本消息队列只包括三个角色：

Publisher：消息发布者，将消息发送到队列Queue

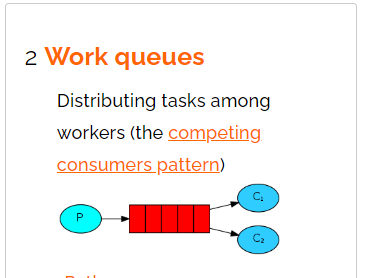
Queue：消息队列，负责接收并缓存消息

Consumer：订阅队列，处理队列中的消息



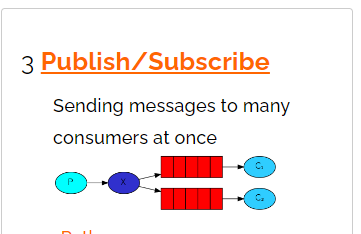
##### 2：工作消息队列（WorkQueue）：

这个队列和上面的基本消息队列一样，消息的发送和接收都是直接基于队列来完成的，而并没有出现我们上面在架构中出现的交换机，不是一个完整的消息驱动模型。如图所示。

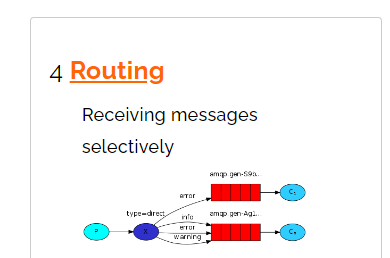


##### 发布订阅（Publish、Subscribe），剩下三种都是发布订阅，只不过这三种用到的交换机类型不同，所以根据交换机类型不同分为三种：

##### 3：Fanout Exchange：广播



##### 4：Direct Exchange：路由



##### 5：Topic Exchange：主题

