RabbitMq

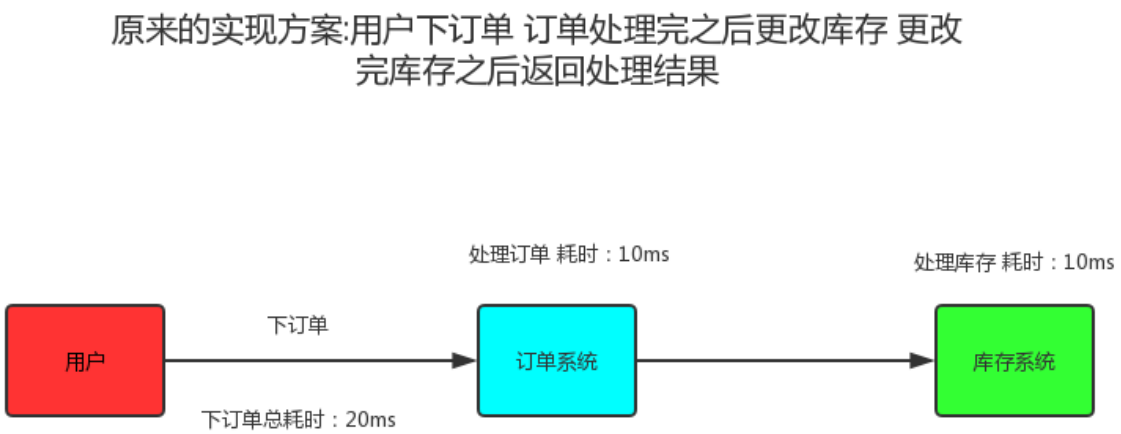
消息中间件介绍&为什么要使用消息中间件&什么时候使用消息中间件

我们用java来举例子， 打个比方 我们客户端发送一个下单请求给订单系统（order）订单系统发送了

一个请求给我们的库存系统告诉他需要更改库存了， 我已经下单了， 这里， 每一个请求我们都可以看作一条消息，

但是 我们客户端需要等待订单系统告诉我这条消息的处理结果（我到底有没有下单成功） 但是 订单系统不需要知道库存系统这条消息的处理情况 因为无论你库存有没有改动成功， 我订单还是下了， 因为是先下完了订单（下成功了） 才去更改库存， 库存如果更改出BUG了 那是库存系统的问题， 这个BUG不会影响订单系统。如果这里你能理解的话， 那么我们就能发现 **我们用户发送的这条消息（下订单）， 是需要同步的（我需要知道结果）， 订单发送给库存的消息，是可以异步的（我不想知道你库存到底改了没， 我只是通知你我这边成功下了一个订单）**

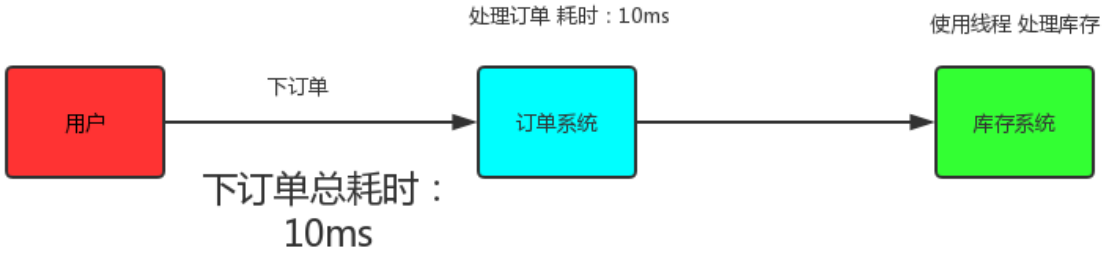
那么如果我们还按原来的方式去实现这个需求的话， 那么结果会是这样：



问题：笔记中有说到， 用户实际上只需要得到订单系统的返回信息， 但是这里其实是在等待订单和库存的处理结果，所以是没必要的， 而且库存系统出现问题会影响用户所看到的结果（报错） 其实对于用户来讲不关心库存 只关心订单

那可能有同学说了， 我们订单系统开辟线程去访问库存系统不就好了吗?

使用线程的解决方案：用户下订单 订单系统开辟线程去调用库存 同时返回下订单的信息

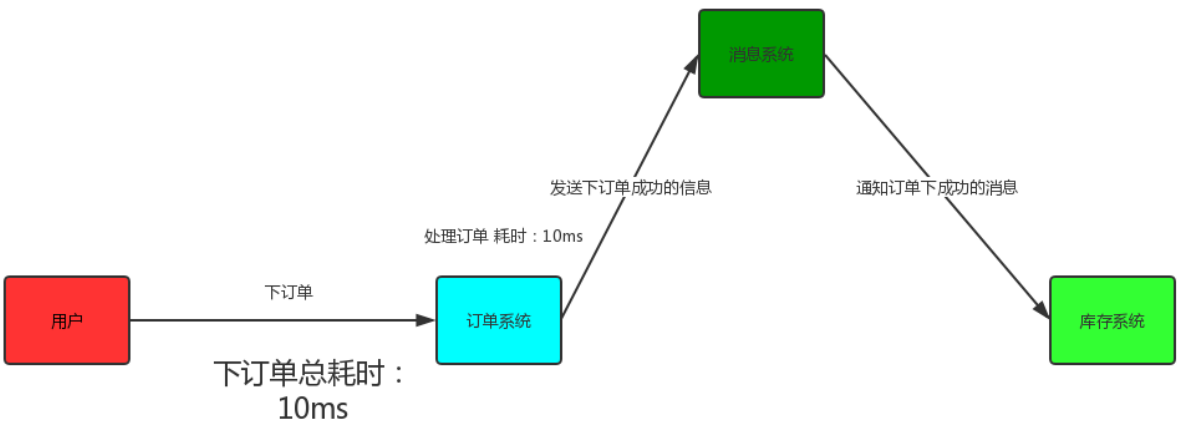


问题：我们能发现，现在用户不用等库存的处理结果了，而且库存也不会影响用户下订单了， 但是 我们需要管理线程池，代码耦合度严重。

##

使用线程池解决 确实可以， 但是也有他的缺点， 那么 到底怎么来完美解决这个问题呢?

最终方案：使用一个独立的系统来处理他们之间的消息

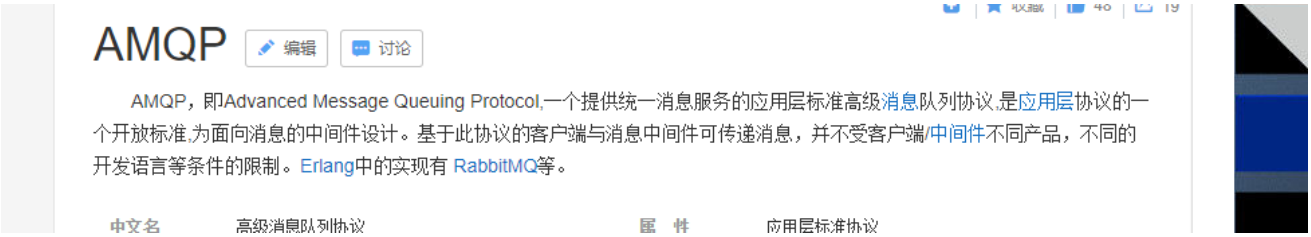


如果这张图能理解的话， 那么 这个消息系统， 就是我们的消息中间件。

RabbitMq介绍&AMQP介绍

导语:我们刚刚介绍了什么是消息中间件， 那么RabbitMq就是对于消息中间件的一种实现，市面上还有很多很多实现， 比如RabbitMq、ActiveMq、ZeroMq、kafka，以及阿里开源的RocketMQ等等 我们这节主要讲RabbitMq

AMQP



这里引用百度的一句话 再加以我的理解： AMQP 其实和Http一样 都是一种协议， 只不过 Http是针对网络传输的， 而AMQP是基于消息队列的

AMQP 协议中的基本概念：

- **Broker**: 接收和分发消息的应用，我们在介绍消息中间件的时候所说的消息系统就是Message Broker。
- **Virtual host**: 出于多租户和安全因素设计的，把AMQP的基本组件划分到一个虚拟的分组中，类似于网络中的namespace概念。当多个不同的用户使用同一个RabbitMQ server提供的服务时，可以划分出多个vhost，每个用户在自己的vhost创建exchange / queue等。
- **Connection**: publisher / consumer和broker之间的TCP连接。断开连接的操作只会在client端进行，Broker不会断开连接，除非出现网络故障或broker服务出现问题。
- **Channel**: 如果每一次访问RabbitMQ都建立一个Connection，在消息量大的时候建立TCP Connection的开销将是巨大的，效率也较低。Channel是在connection内部建立的逻辑连接，如果应用程序支持多线程，通常每个thread创建单独的channel进行通讯，AMQP method包含了channel id帮助客户端和message broker识别channel，所以channel之间是完全隔离的。Channel作为轻量级的Connection极大减少了操作系统建立TCP connection的开销。
- **Exchange**: message到达broker的第一站，根据分发规则，匹配查询表中的routing key，分发消息到queue中去。常用的类型有：direct (point-to-point), topic (publish-subscribe) and fanout (multicast)。
- **Queue**: 消息最终被送到这里等待consumer取走。一个message可以被同时拷贝到多个queue中。
- **Binding**: exchange和queue之间的虚拟连接，binding中可以包含routing key。Binding信息被保存到exchange中的查询表中，用于message的分发依据。

Exchange的类型:

direct :

这种类型的交换机的路由规则是根据一个routingKey的标识，交换机通过一个routingKey与队列绑定 ，在生产者生产消息的时候 指定一个routingKey 当绑定的队列的routingKey 与生产者发送的一样 那么交换机会吧这个消息发送给对应的队列。

fanout:

这种类型的交换机路由规则很简单，只要与他绑定了的队列， 他就会吧消息发送给对应队列（与routingKey没关系）

topic:(因为*在这个笔记软件里面是关键字，所以下面就用星替换掉了)

这种类型的交换机路由规则也是和routingKey有关 只不过 topic他可以根据:星,#（ 星号代表过滤一单词，#代表过滤后面所有单词， 用.隔开）来识别routingKey 我打个比方 假设 我绑定的routingKey 有队列A和B A的routingKey是：星.user B的routingKey是: #.user

那么我生产一条消息routingKey 为： error.user 那么此时 2个队列都能接受到， 如果改为 topic.error.user 那么这时候 只有B能接受到了

headers:

这个类型的交换机很少用到，他的路由规则 与routingKey无关 而是通过判断header参数来识别的， 基本上没有应用场景，因为上面的三种类型已经能应付了。

RabbitMQ

MQ： message Queue 顾名思义 消息队列， 队列大家都知道， 存放内容的一个东西， 存放的内容先进先出， 消息队列， 只是里面存放的内容是消息而已。

RabbitMq 是一个开源的 基于AMQP协议实现的一个完整的企业级消息中间件，服务端语言由Erlang（面向并发编程）语言编写 对于高并发的处理有着天然的优势，客户端支持非常多的语言：

- Python
- Java
- Ruby
- PHP
- C#
- JavaScript
- Go
- Elixir
- Objective-C
- Swift

主流MQ对比

特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka
单机吞吐量	万级，比 RocketMQ、Kafka 低一个数量级	同 ActiveMQ	10 万级，支撑高吞吐	10 万级，高吞吐，一般配合大数据类的系统来进行实时数据计算、日志采集等场景
topic 数量对吞吐量的影响			topic 可以达到几百/几千的级别，吞吐量会有较小幅度的下降，这是 RocketMQ 的一大优势，在同等机器下，可以支撑大量的 topic	topic 从几十到几百个时候，吞吐量会大幅度下降，在同等机器下，Kafka 尽量保证 topic 数量不要过多，如果要支撑大规模的 topic，需要增加更多的机器资源
时效性	毫秒级	微秒级，这是 RabbitMQ 的一大特点，延迟最低	毫秒级	毫秒级
可用性	高，基于主从架构实现高可用	同 ActiveMQ	非常高，分布式架构	非常高，分布式，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用
消息可靠性	有较低的概率丢失数据		经过参数优化配置，可以做到 0 丢失	同 RocketMQ
功能支持	MQ 领域的功能极其完备	基于 erlang 开发，并发能力很强，性能极好，延时很低	MQ 功能较为完善，还是分布式的，扩展性好	功能较为简单，主要支持简单的 MQ 功能，在大数据领域的实时计算以及日志采集被大规模使用

RabbitMQ服务端部署

在介绍消息中间件的时候所提到的“消息系统” 便是我们这节的主题：RabbitMq 如同redis一样 他也是采用c/s架构 由服务端 与客户端组成， 我们现在我们计算机上部署他的服务端

由于我们刚刚介绍过了RabbitMQ服务端是由Erlang语言编写所以我们这里先下载Erlang语言的环境

注意：如果是在官网下的RabbitMQ服务端的话 Erlang语言的版本不能太低， 不然要卸载掉旧的去装新的， 我们这里下载OTP21.0版本直接从外网下载会很慢， 我这里直接贴上百度网盘的地址(因为这个东西还是有点大的)

https://pan.baidu.com/s/1pZJ8l2f3omrgnuCm9a8DVA

我们再去官网下载 他的服务端安装包

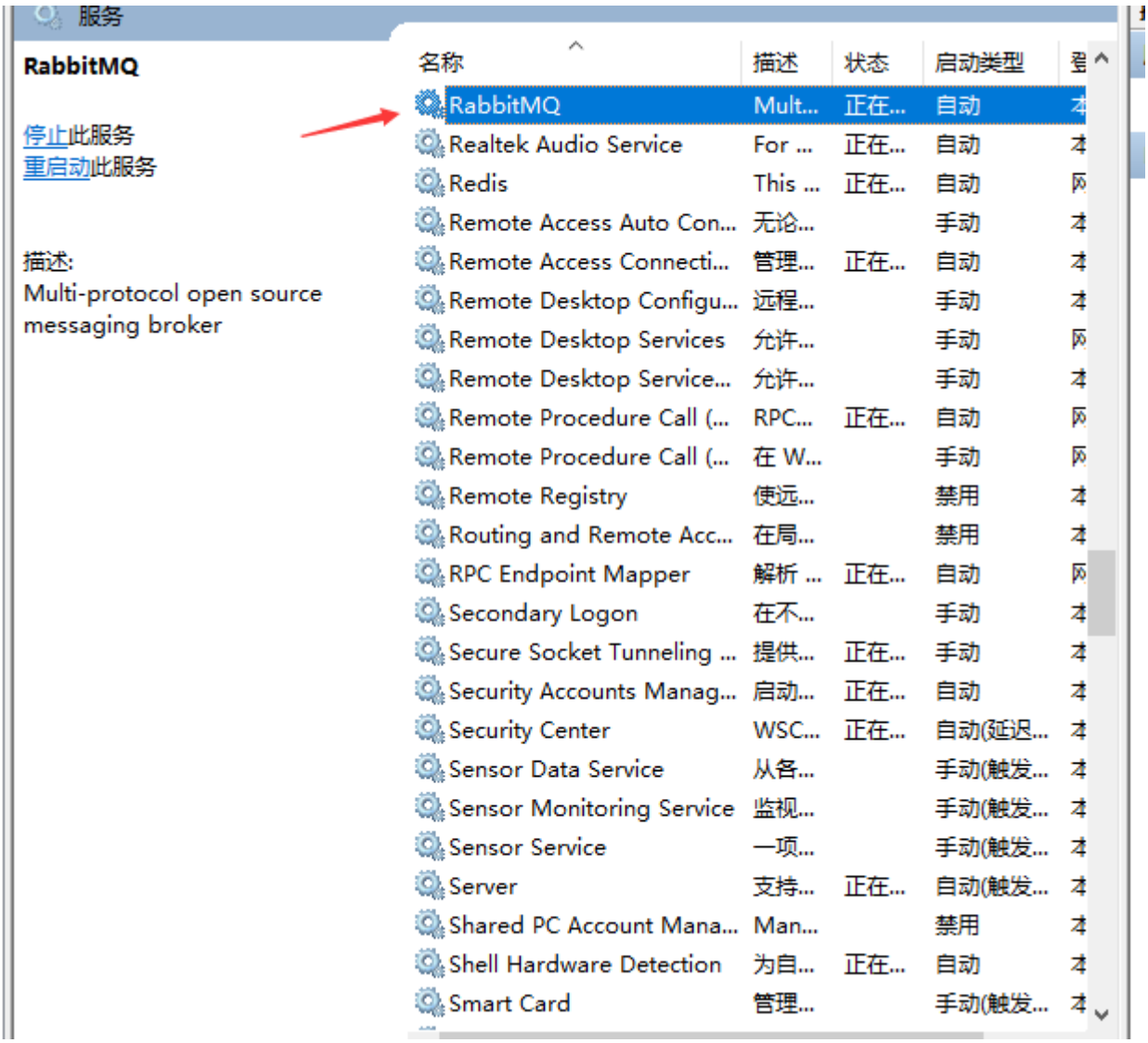
http://www.rabbitmq.com/download.html

根据自己的系统选择下载即可

注意！ 需要先下载Erlang再下载安装包安装， 不然安装RabbitMQ服务端的时候会提示你本地没有Erlang环境

安装的话， 基本上就是默认的选项不用改

如何看RabbitMq安装完成了？ 在系统-服务中找到如下即可：



包括启动 停止 重启 服务等

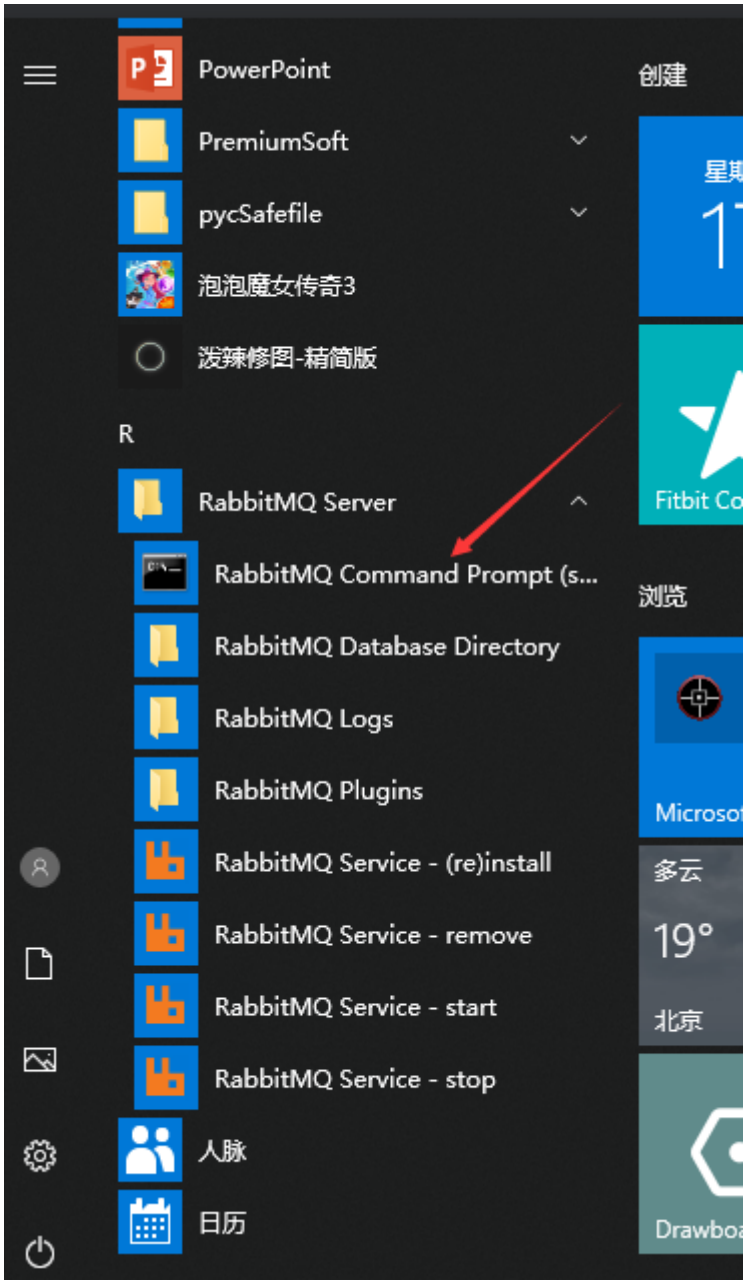
RabbitMQ安装会附带一个管理工具（方便我们能直观的查看整个RabbitMQ的运行状态和详细数据等，有点像Navicat 对应Mysql的关系） 值得一提的是， 管理工具和RabbitMQ是两码事 希望同学们不要混稀了。

管理工具启动方式：

到你们安装的 RabbitMQ Server\rabbitmq_server-3.7.12\sbin 目录下面 执行一条cmd命令：

rabbitmq-plugins enable rabbitmq_management

直接复制这条命令即可， 当然 嫌每次都要去目录中去执行的麻烦的话， 可以配置一个环境变量 或者在我们的开始菜单栏中找到这个：



输入完启动命令后 稍微等一下会有结果返回 然后可以打开浏览器 输入

http://127.0.0.1:15672

访问管理页面：

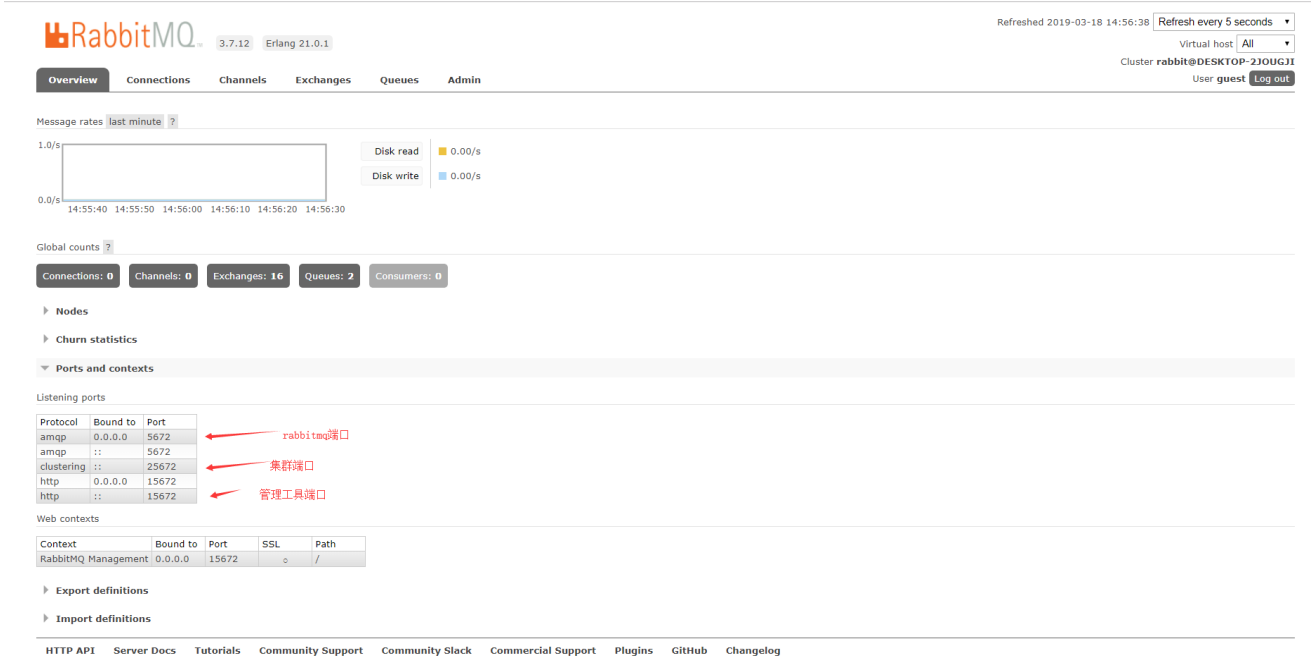
默认账号密码都是

guest 即

username： guest

password：guest

登录进去之后会看到如下界面（因为我不小心装了2次RabbitMq 所以这里能看到都重复了， 你们自己那不会重复，然后我们刚刚说了 管理工具和rabbitmq 是两码事 所以端口也就不一样）



这个页面在笔记里面介绍起来可能比较复杂， 就不一一介绍了， 我这里讲个重点， 就是线上环境下一定要吧guest用户（当然 guest这个用户只能本机才能登陆）删掉并且新加一个用户， 这里就演示一下这个功能

首先 点击admin页签， 在下面找到Add User

然后输入账号 密码 确认密码 这个Tags其实是一个用户权限标签， 关于他的介绍可以看官方介绍（点旁边那个小问号就好了，我这里直接翻译他的介绍）

3 items,

Comma-separated list of tags to apply to the user. Currently **supported by the management plugin**:

management
: User can access the management plugin

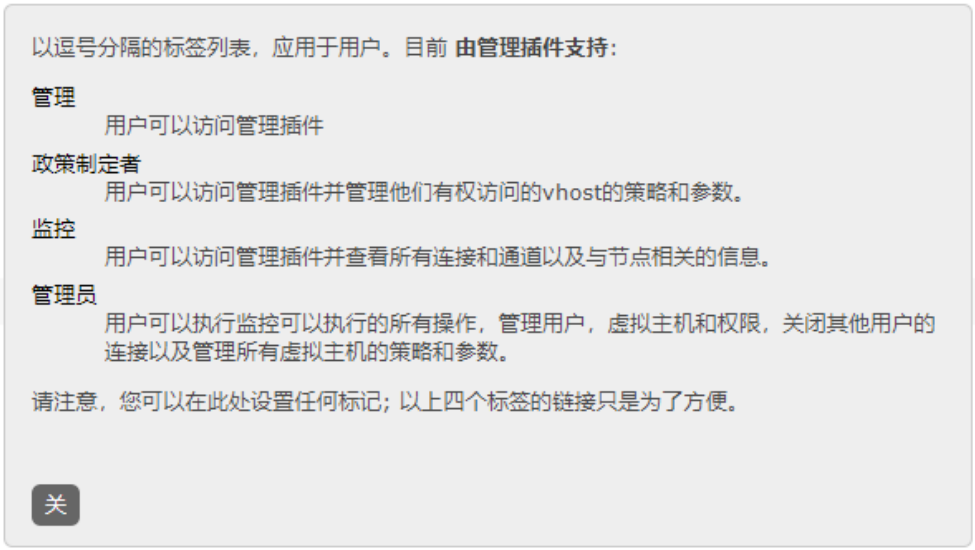
policymaker
: User can access the management plugin and manage policies and parameters for the vhosts they have access to.

monitoring
: User can access the management plugin and see all connections and channels as well as node-related information.

administrator
: User can do everything monitoring can do, manage users, vhosts and permissions, close other user's connections, and manage policies and parameters for all vhosts.

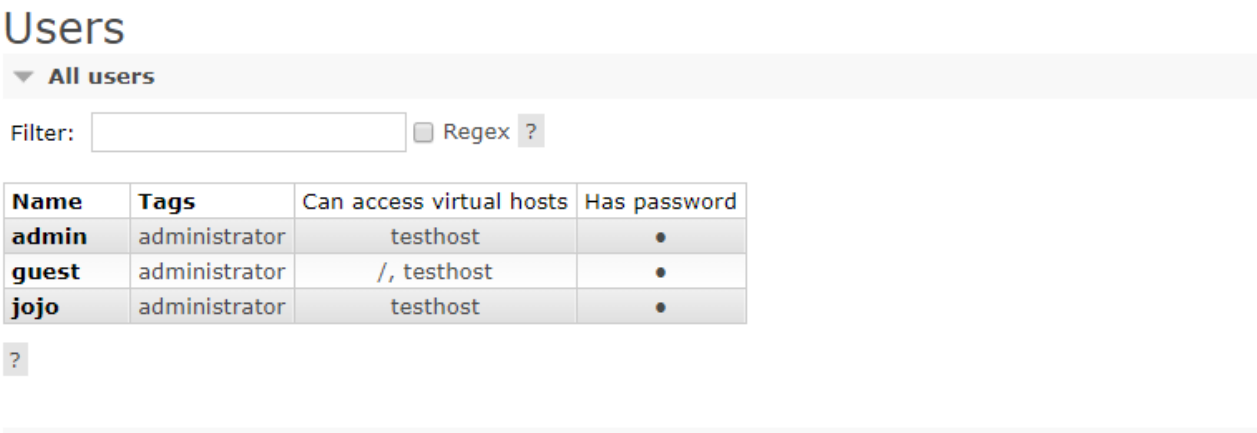
Note that you can set any tag here; the links for the above four tags are just for convenience.

Close

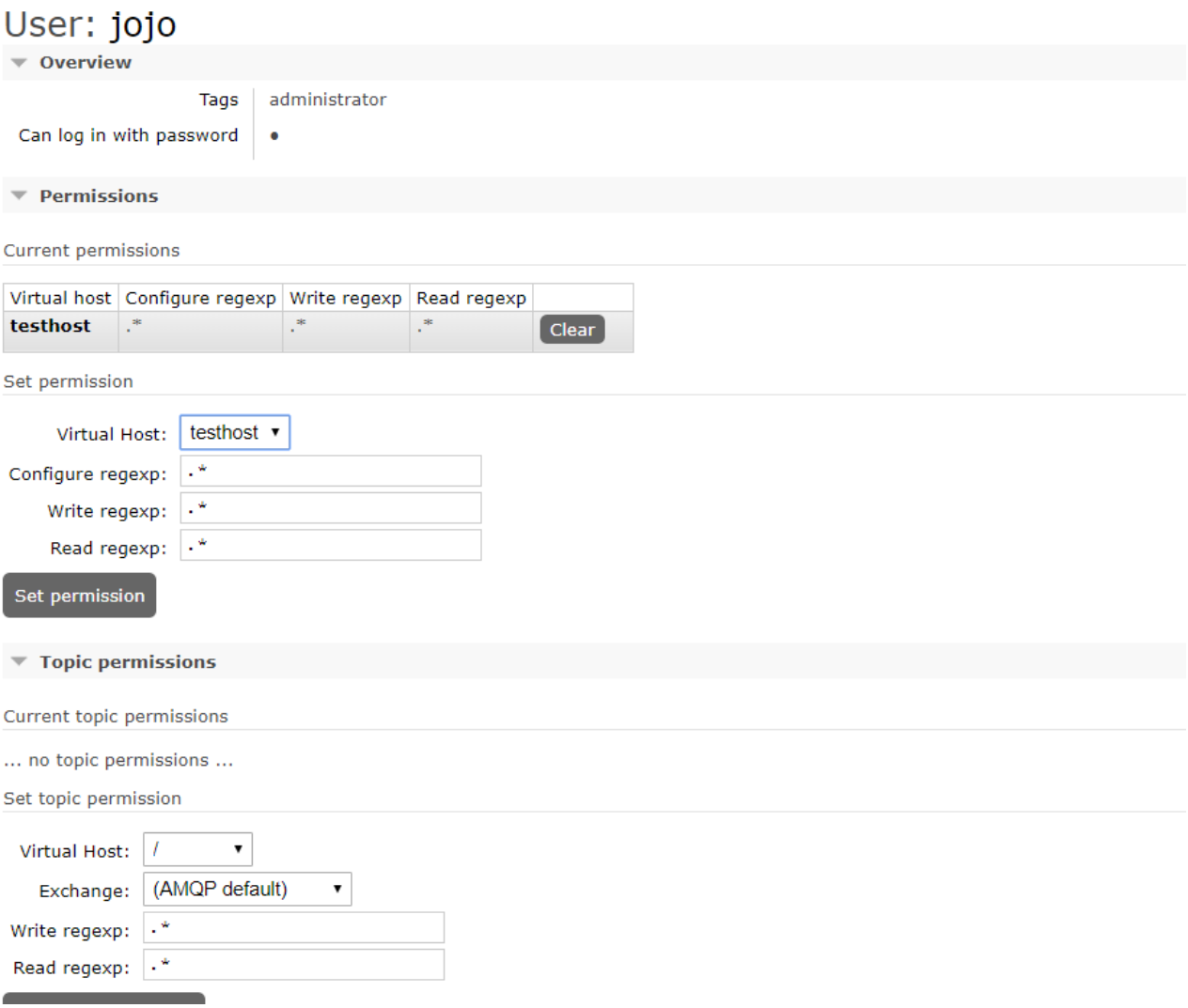


填写完之后点击AddUser 就可以添加一个用户了， 添加完用户之后还要给这个用户添加对应的权限（注：Targ不等于权限）

比如说 我刚刚添加了一个jojo角色



点击这个jojo可以进去给他添加权限 这个权限可以是 Virtual host 级别的 也可以是交换机级别的 甚至是细化到某一个读写操作 我这里就给他添加一个Virtual host权限



这里 我们给了他 testhost这个Virtual host的权限 正则匹配都是* 也就是所有权限

然后点击set添加完毕

那么管理页面 我们就讲到这里

RabbitMq快速开始

因为我们这里是用java来作为客户端， 我们首先引入maven依赖：

```
<dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>5.1.2</version>
</dependency>
```

（注意的是， 我这里引入的是5.x的rabbitmq客户端版本， 那么我们jdk的版本最好在8以上，反之， 这里就建议使用4.x的版本，这里仅仅讨论jdk8 其他的版本不做讨论）

首先 我们编写一个连接的工具类：

```
package com.luban.util;

import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * 需要咨询java高级VIP课程的同学可以木兰老师的QQ：2746251334
 * 需要往期视频的同学可以加加安其拉老师的QQ：3164703201
 * author：鲁班学院-商鞅老师
 */
public class ConnectionUtil {


    public static final String QUEUE_NAME  = "testQueue";

    public static  final  String EXCHANGE_NAME = "exchange";



    public static Connection getConnection() throws Exception{
        //创建一个连接工厂
        ConnectionFactory connectionFactory = new ConnectionFactory();
        //设置rabbitmq 服务端所在地址 我这里在本地就是本地
        connectionFactory.setHost("127.0.0.1");
        //设置端口号，连接用户名，虚拟地址等
        connectionFactory.setPort(5672);
        connectionFactory.setUsername("jojo");
        connectionFactory.setPassword("jojo");
        connectionFactory.setVirtualHost("testhost");
        return connectionFactory.newConnection();
    }



}
```

然后我们编写一个消费者（producer），和一个生产者（consumer）：

生产者：

```
public class Consumer {

    public static void sendByExchange(String message) throws Exception {

        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();
```

```
        //声明队列
        channel.queueDeclare(ConnectionUtil.QUEUE_NAME,true,false,false,null);
        // 声明exchange
        channel.exchangeDeclare(ConnectionUtil.EXCHANGE_NAME, "fanout");
        //交换机和队列绑定
        channel.queueBind(ConnectionUtil.QUEUE_NAME, ConnectionUtil.EXCHANGE_NAME, "");
        channel.basicPublish(ConnectionUtil.EXCHANGE_NAME, "", null,
message.getBytes());
        System.out.println("发送的信息为:" + message);
        channel.close();
        connection.close();
    }

}
```

消费者:

```
public class Producer {

    public static void getMessage() throws Exception {
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();
//        channel.queueDeclare(ConnectionUtil.QUEUE_NAME,true,false,false,null);
        DefaultConsumer deliverCallback = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println(new String(body, "UTF-8"));
            }
        };
        channel.basicConsume(ConnectionUtil.QUEUE_NAME, deliverCallback);
    }

}
```

这里， 我们演示绑定fanout的类型的交换机， 所以不需要routingKey 就可以路由只需要绑定即可

（可能有同学要问了， 如果没有绑定交换机怎么办呢？ 没有绑定交换机的话， 消息会发给rabbitmq默认的交换机里面 默认的交换机隐式的绑定了所有的队列，默认的交换机类型是direct 路由建就是队列的名字）

基本上这样子的话就已经进行一个快速入门了， 由于我们现在做项目基本上都是用spring boot（就算没用spring boot也用spring 吧） 所以后面我们直接基于spring boot来讲解（rabbitmq的特性，实战等）