

## Windows RabbitMQ 安装与入门

### 1. 环境下载

1. 先下载 erlang 并安装

<https://www.erlang.org/downloads>

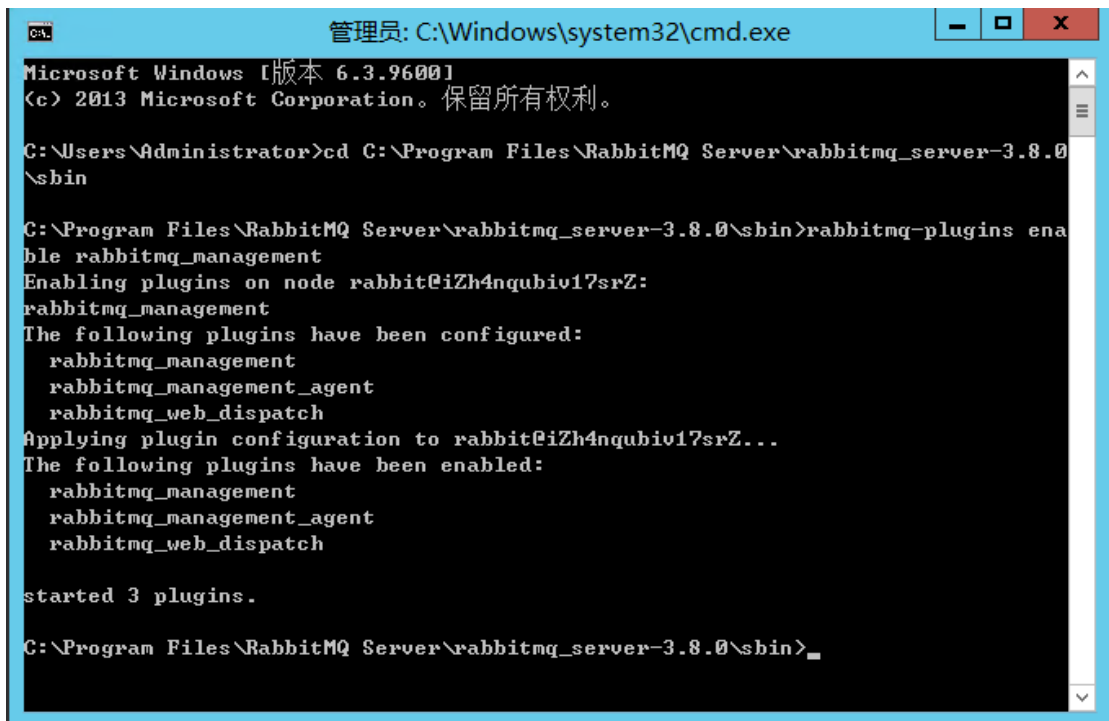
2. 下载 RabbitMQ

<https://www.rabbitmq.com/install-windows.html>

### 2. 安装 Web 插件

cmd 进入 C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.8.0\sbin 目录

输入 rabbitmq-plugins enable rabbitmq\_management 命令完成 web 插件的安装



```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd C:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.0\sbin

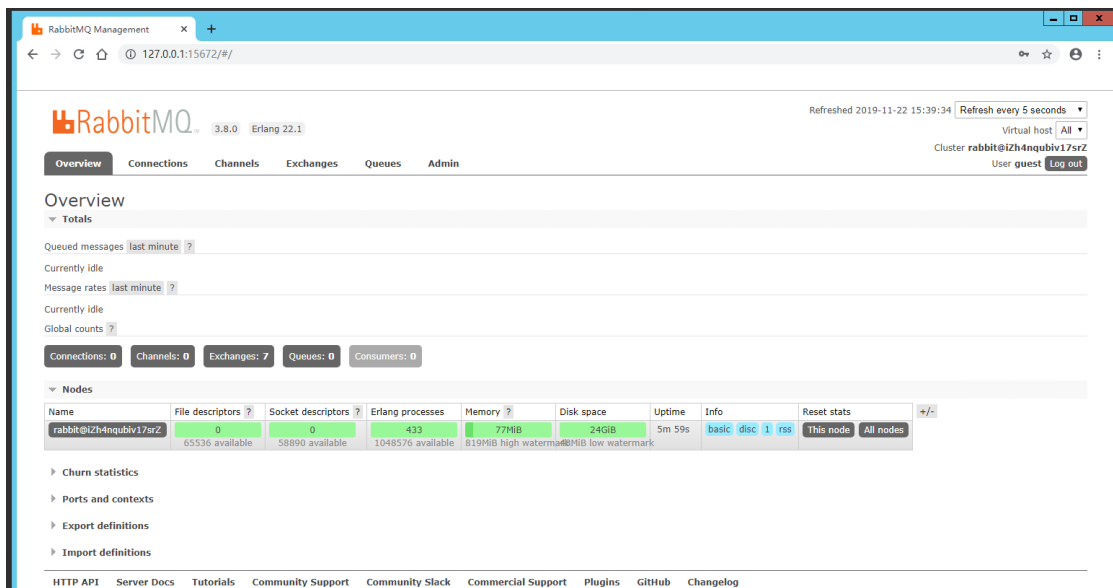
C:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.0\sbin>rabbitmq-plugins enable rabbitmq_management
Enabling plugins on node rabbit@iZh4nqubiv17srZ:
rabbitmq_management
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@iZh4nqubiv17srZ...
The following plugins have been enabled:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch

started 3 plugins.

C:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.0\sbin>
```

### 3. 启动

浏览器输入 <http://127.0.0.1:15672/>，使用默认账号 [ name:guest / password:guest ] 登录后界面如下，使用这个 UI 插件我们可以轻松的查看 RabbitMQ 中的交换机 (exchange)，队列 (queue) 等内容，也可以对 exchange，queue，用户等进行添加、修改、删除操作。



#### 4. Hello World 的开始

1. 创建 2 个控制台程序，并安装 RabbitMQ.Client 程序包
2. 生产者 (Producer) 代码：

```

static void Main(string[] args)
{
    var factory = new ConnectionFactory()
    {
        //rabbitmq-server所在设备ip, 这里就是本机
        HostName = "127.0.0.1",
        UserName = "wyy", //用户名
        Password = "123321" //密码
    };
    //第一步: 创建连接connection
    using (var connection = factory.CreateConnection())
    {
        //第二步: 创建通道channel
        using (var channel = connection.CreateModel())
        {
            //第三步: 声明交换机exchange
            channel.ExchangeDeclare(exchange: "myexchange",
                                    type: ExchangeType.Direct,
                                    durable: true,
                                    autoDelete: false,
                                    arguments: null);

            //第四步: 声明队列queue
            channel.QueueDeclare(queue: "myqueue",
                                  durable: true,
                                  exclusive: false,
                                  autoDelete: false,
                                  arguments: null);

            Console.WriteLine("生产者准备就绪....");
            //第五步: 绑定队列到交换机
            channel.QueueBind(queue: "myqueue", exchange: "myexchange", routingKey: "mykey");
            string message = "";
            //第六步: 发送消息
            //在控制台输入消息, 按enter键发送消息
            while (!message.Equals("quit", StringComparison.CurrentCultureIgnoreCase))
            {
                message = Console.ReadLine();
                var body = Encoding.UTF8.GetBytes(message);
                //基本发布
                channel.BasicPublish(exchange: "myexchange",
                                      routingKey: "mykey",
                                      basicProperties: null,
                                      body: body);

                Console.WriteLine($"消息 [{message}] 已发送到队列");
            }
        }
    }
    Console.ReadKey();
}

```

### 3. 消费者(Consumer)代码:

```

static void Main(string[] args)
{
    var factory = new ConnectionFactory()
    {
        //rabbitmq-server所在设备ip, 这里就是本机
        HostName = "127.0.0.1",
        UserName = "wyy", //用户名
        Password = "123321" //密码
    };
    //第一步: 创建连接connection
    using (var connection = factory.CreateConnection())
    {
        //第二步: 创建通道channel
        using (var channel = connection.CreateModel())
        {
            //第三步: 声明队列queue
            channel.QueueDeclare(queue: "myqueue",
                                durable: true,
                                exclusive: false,
                                autoDelete: false,
                                arguments: null);

            //第四步: 定义消费者
            var consumer = new EventingBasicConsumer(channel);
            consumer.Received += (model, ea) =>
            {
                var body = ea.Body;
                var message = Encoding.UTF8.GetString(body);
                Console.WriteLine($"接受到消息 [{message}] ");
            };
            Console.WriteLine("消费者准备就绪....");
            //第五步: 处理消息
            channel.BasicConsume(queue: "myqueue",
                                autoAck: true,
                                consumer: consumer);

            Console.ReadLine();
        }
    }
}

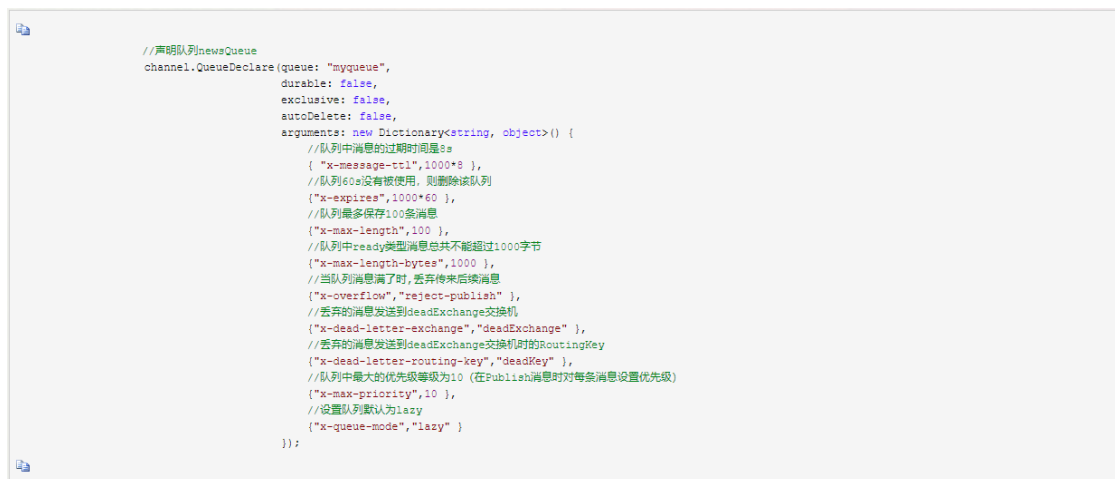
```

4. 依次运行 Producer 和 Consumer 两个应用程序，运行结果如下：



The image shows two command prompt windows. The top window, titled 'C:\Program Files\dotnet\dotnet.exe', is labeled 'Consumer' in red. It displays the following text: '消费者准备就绪....', '接受到消息【hello】', '接受到消息【world】', and '接受到消息【hello rabbitmq】'. The bottom window, titled 'C:\WINDOWS\system32\cmd.exe', is labeled 'Producer' in red. It displays: '生产者准备就绪....', 'hello', '消息【hello】已发送到队列', 'world', '消息【world】已发送到队列', 'hello rabbitmq', and '消息【hello rabbitmq】已发送到队列'.

## 5. QueueDeclare 方法详解



```
//声明队列newsQueue
channel.QueueDeclare(queue: "myqueue",
    durable: false,
    exclusive: false,
    autoDelete: false,
    arguments: new Dictionary<string, object>() {
        //队列中消息的过期时间8s
        { "x-message-ttl", 1000*8 },
        //队列60s没有被使用, 则删除该队列
        { "x-expires", 1000*60 },
        //队列最多保存100条消息
        { "x-max-length", 100 },
        //队列中ready类型消息总共不能超过1000字节
        { "x-max-length-bytes", 1000 },
        //当队列消息满了时, 丢弃传来后续消息
        { "x-overflow", "reject-publish" },
        //丢弃的消息发送到deadExchange交换机
        { "x-dead-letter-exchange", "deadExchange" },
        //丢弃的消息发送到deadExchange交换机时的routingKey
        { "x-dead-letter-routing-key", "deadKey" },
        //队列中最大的优先级等级为10 (在Publish消息时对每条消息设置优先级)
        { "x-max-priority", 10 },
        //设置队列默认为lazy
        { "x-queue-mode", "lazy" }
    });
```

QueueDeclare 方法的参数如下:

**queue**: 队列名字;

**durable**: 是否持久化。设置为 true 时, 队列信息保存在 rabbitmq 的内置数据库中, 服务器重启时队列也会恢复 (注意: 重启后队列内部的消息不会恢复, 怎么实现消息持久化以后会详细介绍);

**exclusive**: 是否排外。设置为 true 时只有首次声明该队列的 Connection 可以访问, 其他 Connection 不能访问该队列; 且在 Connection 断开时, 队列会被删除 (即使 durable 设置为 true 也会被删除);

**autoDelete**: 是否自动删除。设置为 true 时, 表示在最后一条使用该队列的连接 (Connection) 断开时, 将自动删除这个队列;

**arguments**: 设置队列的一些其它属性, 为 Dictionary<string, object> 类型, 下表总结了 arguments 中可以设置的常用属性。

## 6. ExchangeDeclare 方法详解



```
channel.ExchangeDeclare(exchange: "myexchange",
    type: ExchangeType.Direct,
    durable: true,
    autoDelete: false,
    arguments: new Dictionary<string, object> {
        { "alternate-exchange", "BeiYongExchange" } //如果消息不能路由到该交换机, 就把消息路由到备用交换机BeiYongExchange上
    });
```

**exchange**: 交换机名字。

**type**: 交换机类型。exchange 有 direct、fanout、topic、header 四种类型，在下一篇会详细介绍；

**durable**: 是否持久化。设置为 true 时，交换机信息保存在 rabbitmq 的内置数据库中，服务器重启时交换机信息也会恢复；

**autoDelete**: 是否自动删除。设置为 true 时，表示在最后一条使用该交换机的连接(Connection)断开时，自动删除这个 exchange；

**arguments**: 其他的一些参数，类型为 Dictionary<string,object> 。