# RabbitMQ教程

blog.csdn.net/hellozpc/article/details/81436980

如果此教程对您有帮助，就请有钱的捧个钱场，没钱的捧个人场（转载分享）哦~

推荐springCloud教程：
https://blog.csdn.net/hellozpc/article/details/83692496

推荐Springboot教程：
https://blog.csdn.net/hellozpc/article/details/82531834

**欢迎关注公众号**
**微信扫一扫**
**欢迎关注**
**扫一扫**

# RabbitMQ实战教程

## 1.什么是MQ

> 消息队列（Message Queue，简称MQ），从字面意思上看，本质是个队列，FIFO先入先出，只不过队列中存放的内容是message而已。
> 其主要用途：不同进程Process/线程Thread之间通信。

为什么会产生消息队列？有几个原因：

- 不同进程（process）之间传递消息时，两个进程之间耦合程度过高，改动一个进程，引发必须修改另一个进程，为了隔离这两个进程，在两进程间抽离出一层（一个模块），所有两进程之间传递的消息，都必须通过消息队列来传递，单独修改某一个进程，不会影响另一个；

- 不同进程（process）之间传递消息时，为了实现标准化，将消息的格式规范化了，并且，某一个进程接受的消息太多，一下子无法处理完，并且也有先后顺序，必须对收到的消息进行排队，因此诞生了事实上的消息队列；

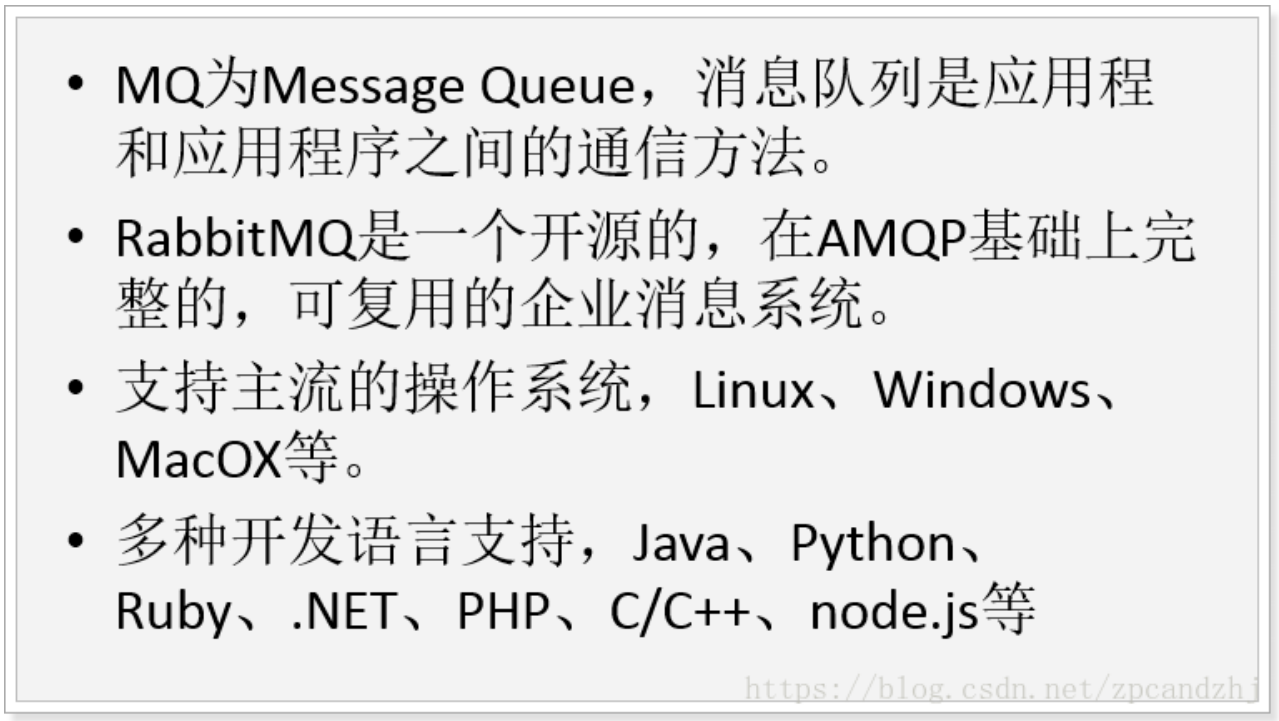
- 关于消息队列的详细介绍请参阅：
  《Java帝国之消息队列》
  《一个故事告诉你什么是消息队列》
  《到底什么时候该使用MQ》

- MQ框架非常之多，比较流行的有RabbitMq、ActiveMq、ZeroMq、kafka，以及阿里开源的RocketMQ。本文主要介绍RabbitMq。

- **本教程pdf及代码下载地址**：
  代码：https://download.csdn.net/download/zpcandzhj/10585077
  教程：https://download.csdn.net/download/zpcandzhj/10585092

## 2.RabbitMQ

### 2.1.RabbitMQ的简介




开发语言：Erlang – 面向并发的编程语言。

收藏 748 0

# Erlang 编辑

Erlang是一种通用的面向并发的编程语言，它由瑞典电信设备制造商爱立信所辖的CS-Lab开发，目的是创造一种可以应对大规模并发活动的编程语言和运行环境。Erlang问世于1987年，经过十年的发展，于1998年发布开源版本。Erlang是运行于虚拟机的解释性语言，但是现在也包含有乌普萨拉大学高性能Erlang计划（HiPE）开发的本地代码编译器，自R11B-4版本开始，Erlang也开始支持脚本式解释器。在编程范型上，Erlang属于多重范型编程语言，涵盖函数式、并发式及分布式。顺序执行的Erlang是一个及早求值、单次赋值和动态类型的函数式编程语言。

Erlang是一个结构化，动态类型编程语言，内建并行计算支持。最初是由爱立信专门为通信应用设计的，比如控制交换机或者变换协议等，因此非常适 合于构建分布式，实时软并行计算系统。使用Erlang编写出的应用运行时通常由成千上万个轻量级进程组成，并通过消息传递相互通讯。进程间上下文切换对于Erlang来说仅仅 只是一两个环节，比起C程序的线程切换要高效得多得多了。

使用Erlang来编写分布式应用要简单的多，因为它的分布式机制是透明的：对于程序来说并不知道自己是在分布式运行。Erlang运行时环境是一个虚拟机，有点像Java虚拟机，这样代码一经编译，同样可以随处运行。它的运行时系统甚至允许代码在不被中断 的情况下更新。另外如果需要更高效的话，字节代码也可以编译成本地代码运行。

| | | | |
|---|---|---|---|
| 中文名 | Erlang | 开发者 | CS-Lab |
| 类　型 | 编程语言 | 问　世 | 1987年 |



### 2.1.1.AMQP

**AMQP是消息队列的一个协议。**

收藏 28 18

# AMQP 编辑

AMQP，即Advanced Message Queuing Protocol,一个提供统一消息服务的应用层标准高级消息队列协议,是应用层协议的一个开放标准,为面向消息的中间件设计。基于此协议的客户端与消息中间件可传递消息，并不受客户端/中间件不同产品，不同开发语言等条件的限制。Erlang中的实现有 RabbitMQ等。

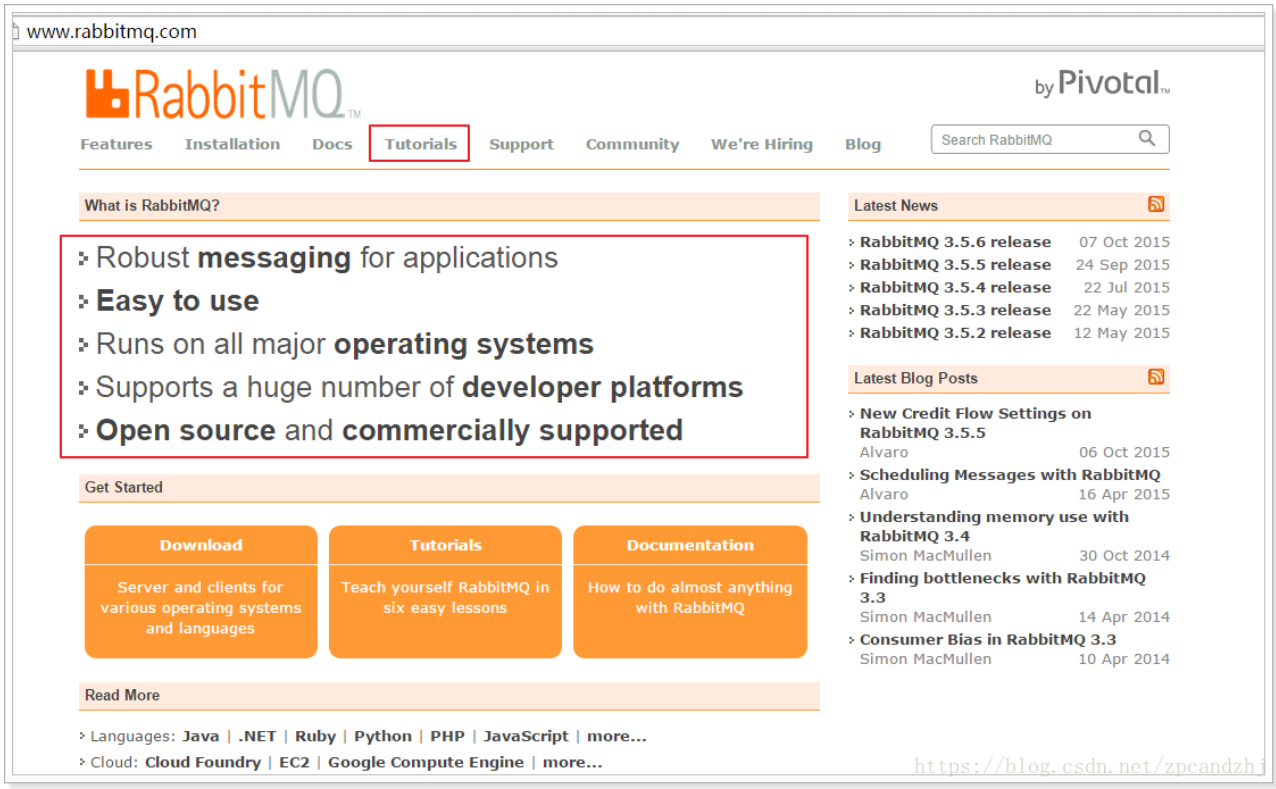
| | | | |
|---|---|---|---|
| 中文名 | 高级消息队列协议 | 属　性 | 应用层标准协议 |
| 外文名 | AMQP | 应用领域 | 计算机 |

目录



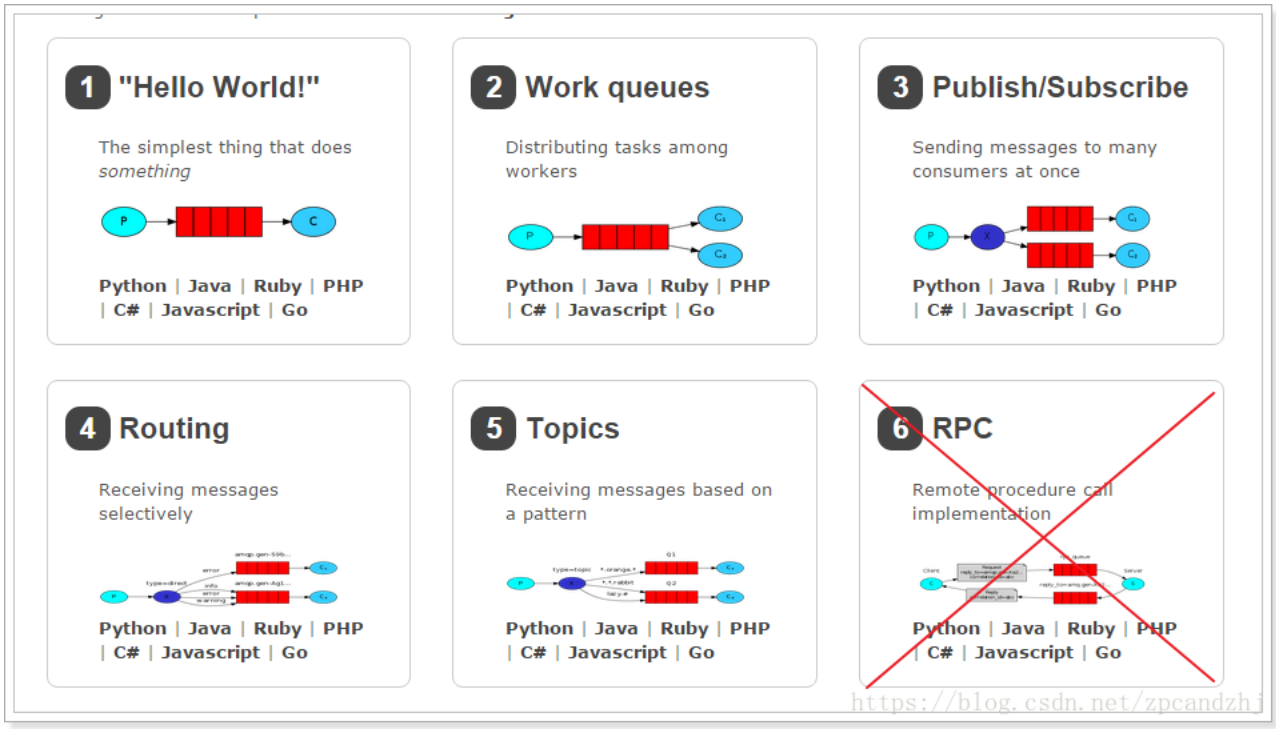


## 2.2.官网




## 2.3.MQ的其他产品


## 2.4.学习5种队列




## 2.5.安装文档

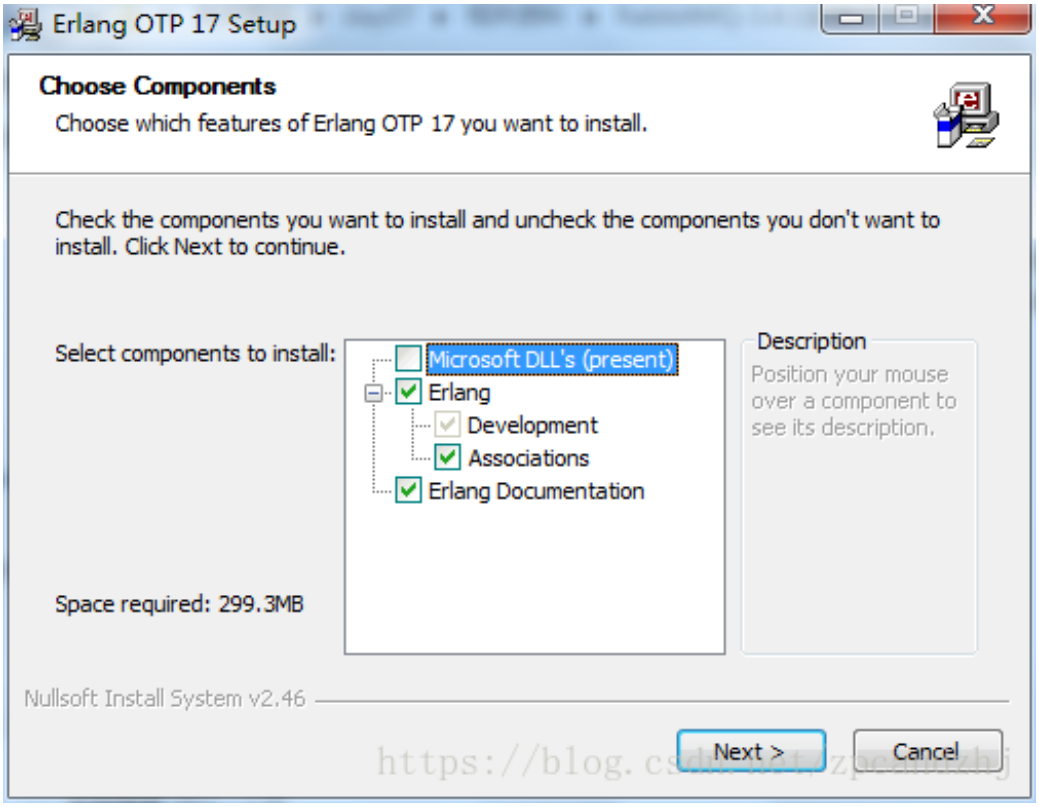

## 3.搭建RabbitMQ环境

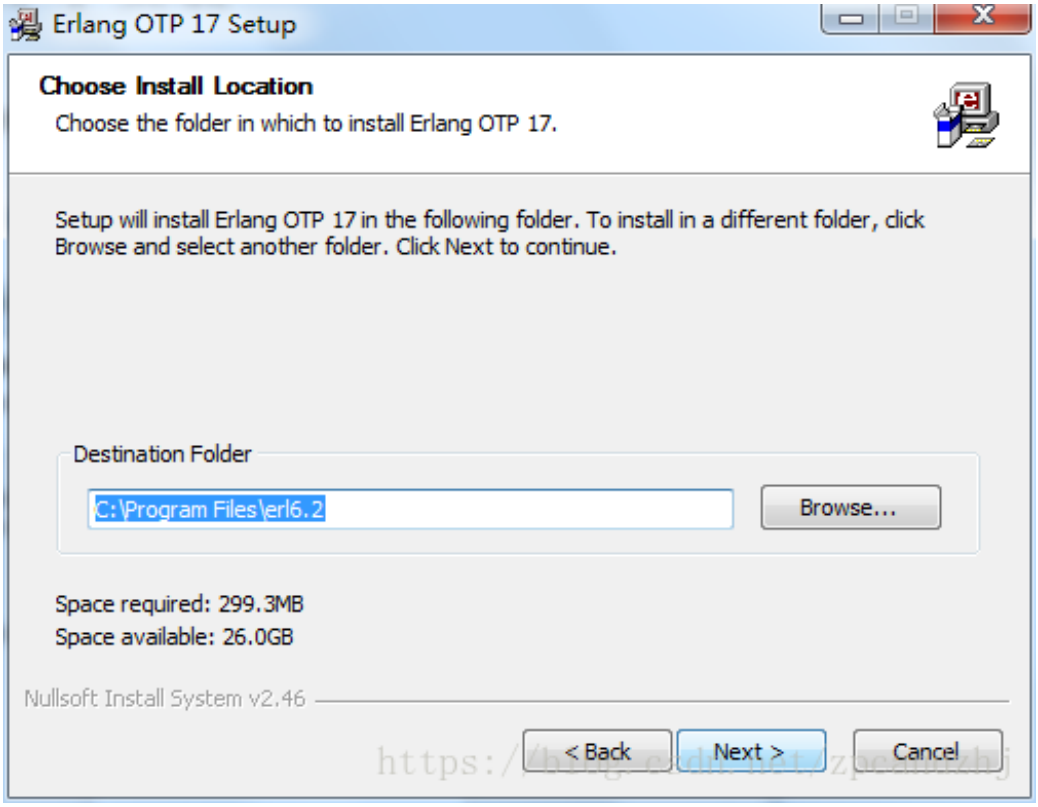
## 3.1.下载

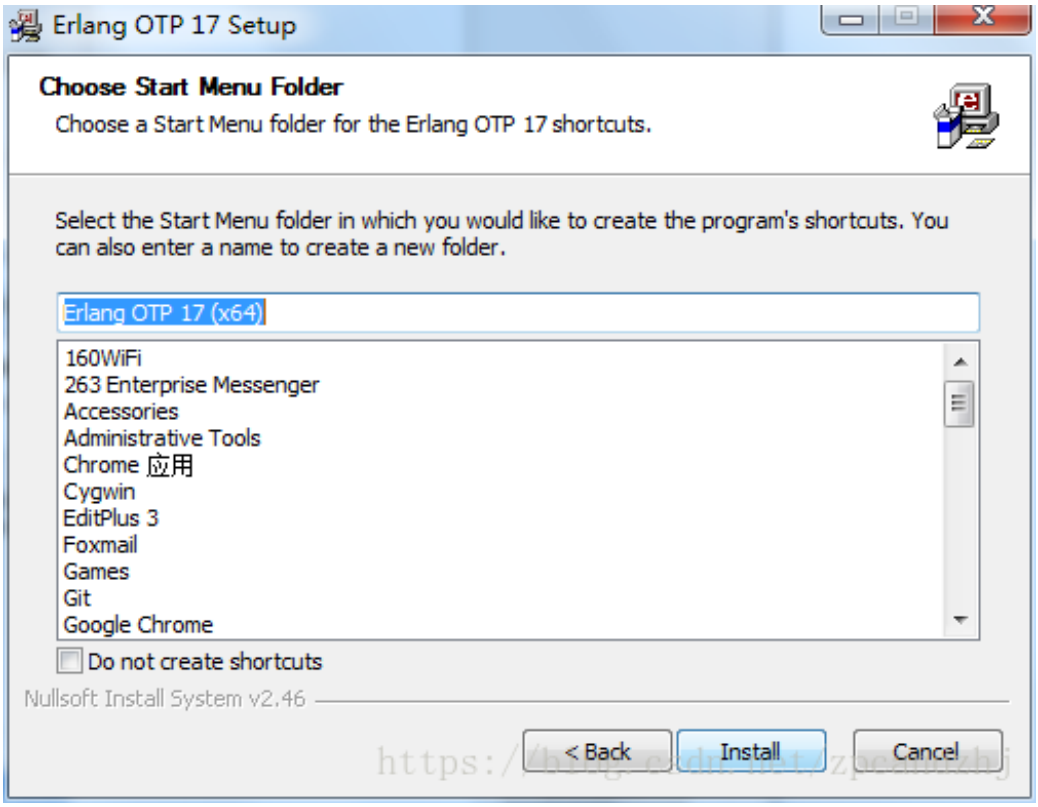
**下载地址：**http://www.rabbitmq.com/download.html

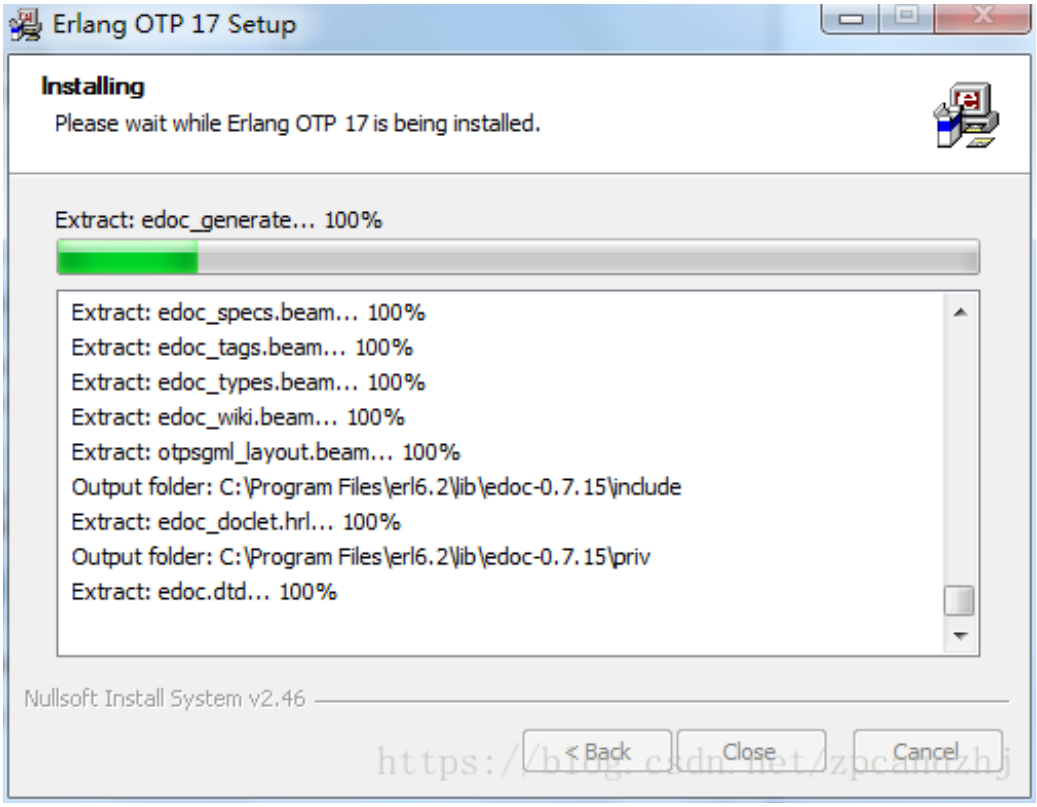
## 3.2.windows下安装

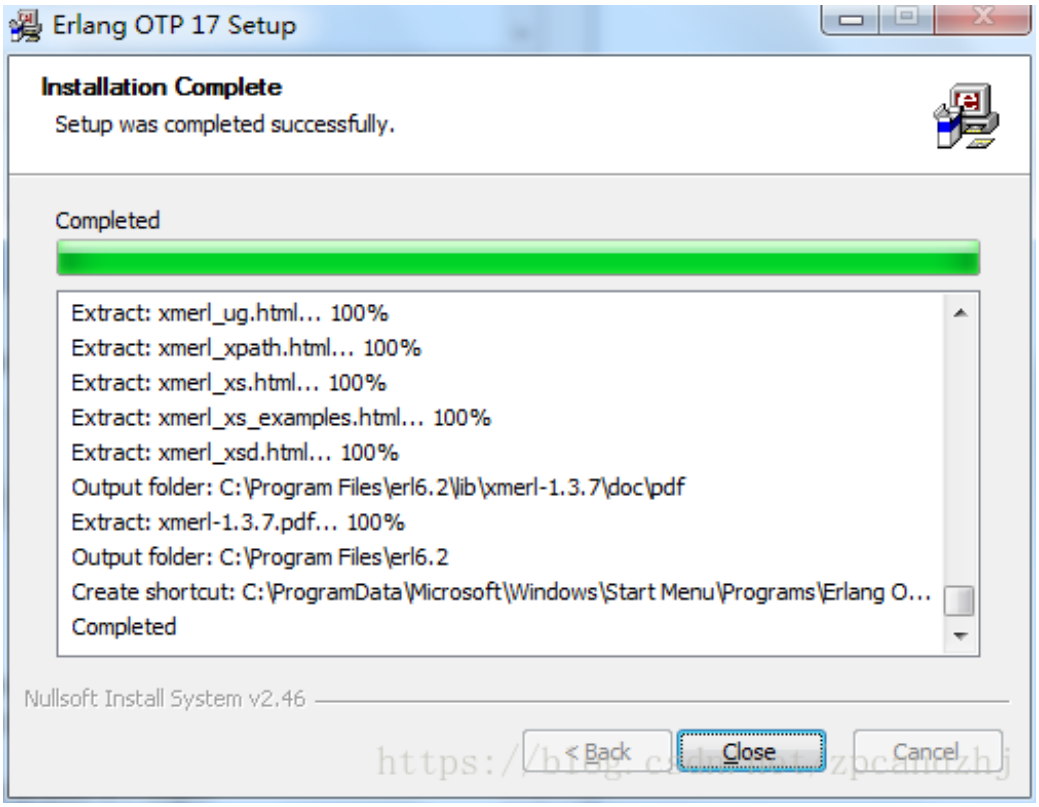
3.2.1.安装Erlang
**下载：**http://www.erlang.org/download/otp_win64_17.3.exe
**安装：**




Erlang OTP 17 Setup
Choose Install Location
Choose the folder in which to install Erlang OTP 17.
Setup will install Erlang OTP 17 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\Program Files\erl6.2
Browse...
Space required: 299.3MB
Space available: 26.0GB
Nullsoft Install System v2.46
< Back
Next >
Cancel

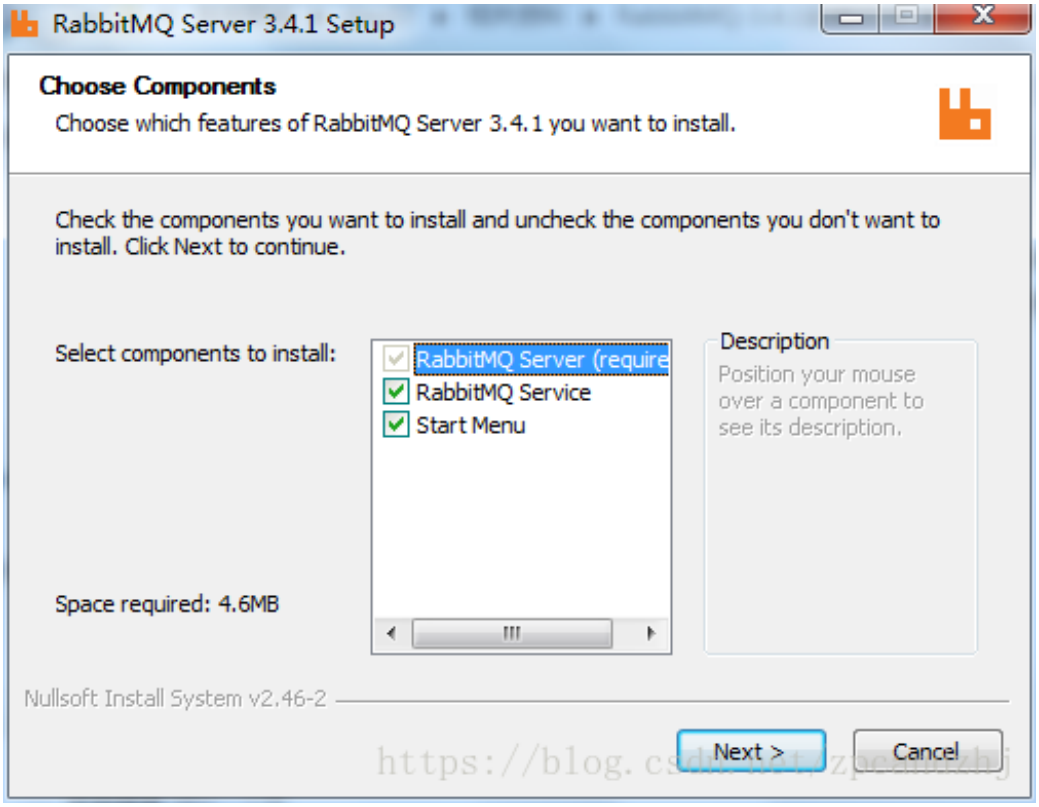

Erlang OTP 17 Setup
Choose Start Menu Folder
Choose a Start Menu folder for the Erlang OTP 17 shortcuts.
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.
Erlang OTP 17 (x64)
160WiFi
263 Enterprise Messenger
Accessories
Administrative Tools
Chrome 应用
Cygwin
EditPlus 3
Foxmail
Games
Git
Google Chrome
Do not create shortcuts
Nullsoft Install System v2.46
< Back
Install
Cancel

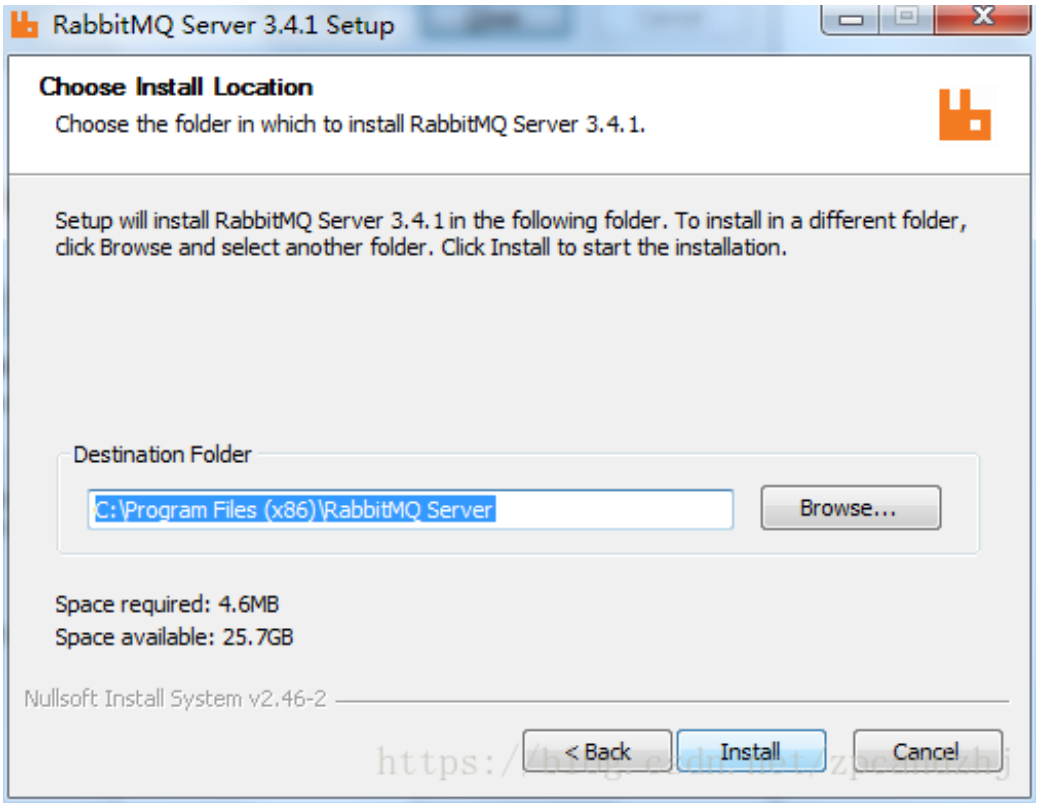


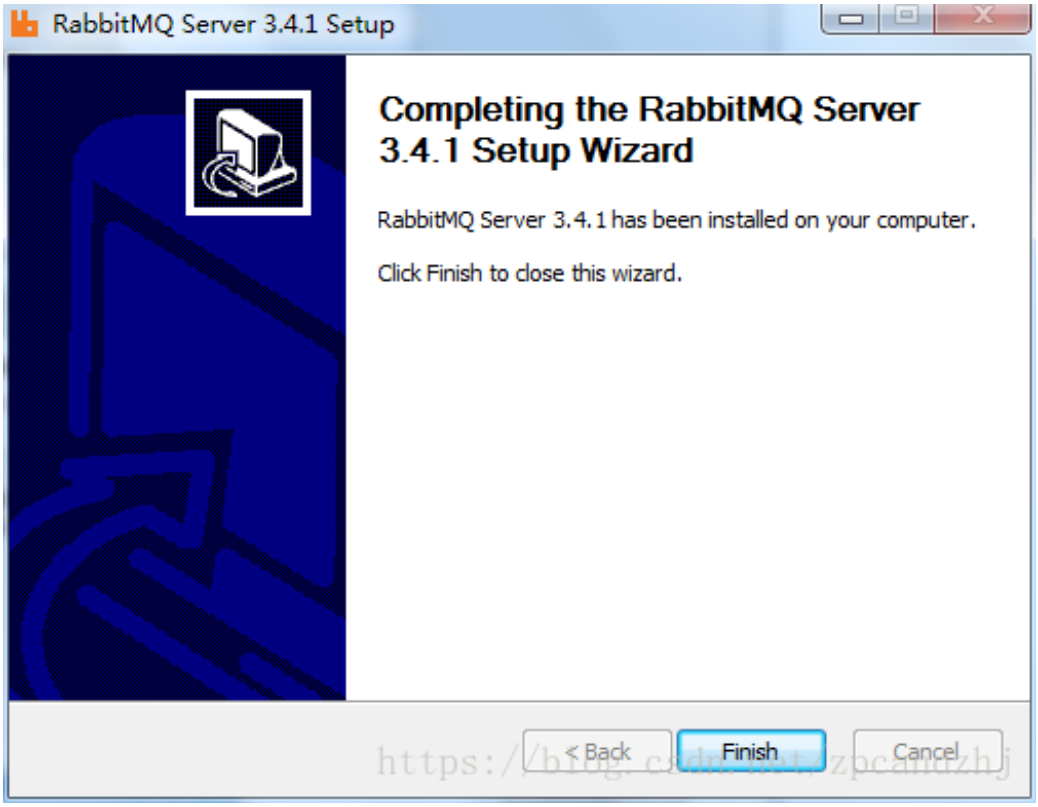




安装完成。

3.2.2.安装RabbitMQ

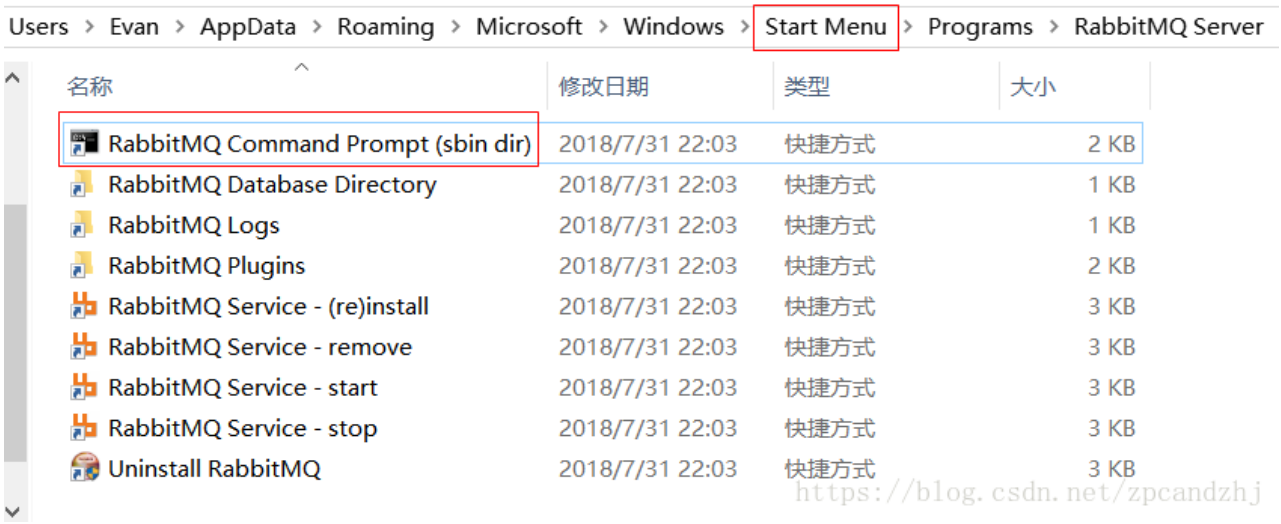

RabbitMQ Server 3.4.1 Setup
Choose Components
Choose which features of RabbitMQ Server 3.4.1 you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
RabbitMQ Server (require
RabbitMQ Service
Start Menu
Description
Position your mouse over a component to see its description.
Space required: 4.6MB
Nullsoft Install System v2.46-2
Next >
Cancel


RabbitMQ Server 3.4.1 Setup
Choose Install Location
Choose the folder in which to install RabbitMQ Server 3.4.1.
Setup will install RabbitMQ Server 3.4.1 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
C:\Program Files (x86)\RabbitMQ Server
Browse...
Space required: 4.6MB
Space available: 25.7GB
Nullsoft Install System v2.46-2
< Back
Install
Cancel




安装完成。

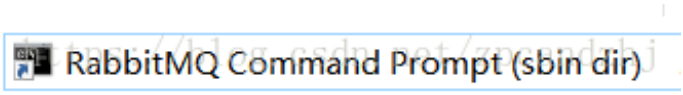
开始菜单里出现如下选项：




启动、停止、重新安装等。

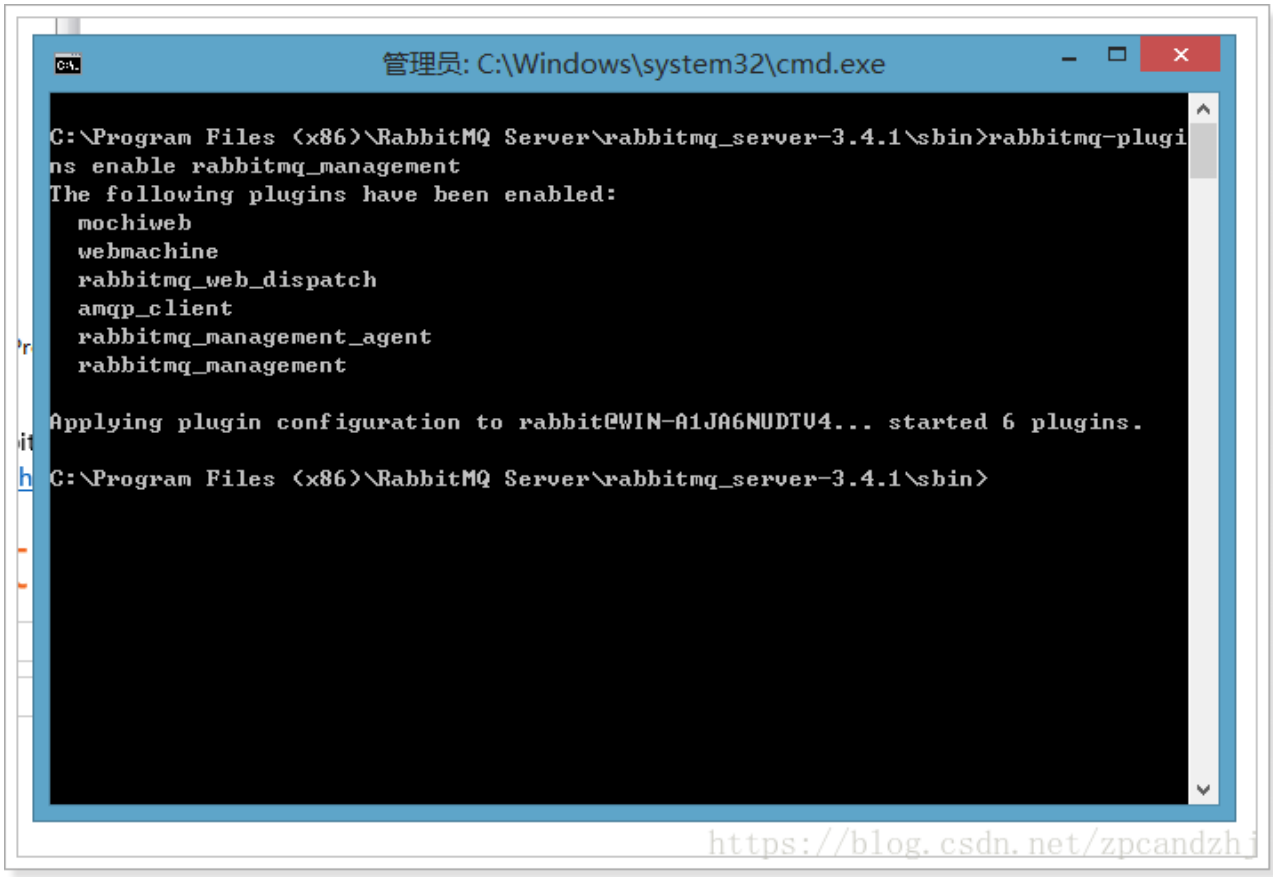
3.2.3.启用管理工具

1、双击




2、进入C:\Program Files (x86)\RabbitMQ Server\rabbitmq_server-3.4.1\sbin输入命令：

rabbitmq-plugins enable rabbitmq_management




这样就启动了管理工具，可以试一下命令：
停止：net stop RabbitMQ
启动：net start RabbitMQ

3、在浏览器中输入地址查看：http://127.0.0.1:15672/

4、使用默认账号登录：guest/ guest


## 3.3.Linux下安装

3.3.1.安装Erlang
3.3.2.添加yum支持
cd /usr/local/src/
mkdir rabbitmq
cd rabbitmq

wget http://packages.erlang-solutions.com/erlang-solutions-1.0-1.noarch.rpm
rpm -Uvh erlang-solutions-1.0-1.noarch.rpm

rpm --import http://packages.erlang-solutions.com/rpm/erlang_solutions.asc

**使用yum安装：**
sudo yum install erlang

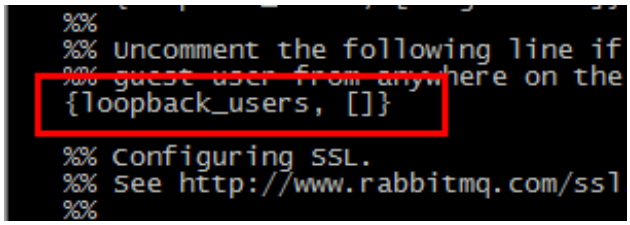
```
总下载量：57 M
Installed size: 136 M
确定吗？[y/N]：y
下载软件包：
(1/60): erlang-17.3-2.el6.x86_64.rpm                                          |  15 kB     00:00
(2/60): erlang-asn1-17.3-2.el6.x86_64.rpm                                     | 1.0 MB     00:17
(3/60): erlang-common_test-17.3-2.el6.x86_64.rpm                              | 957 kB     00:18
(4/60): erlang-compiler-17.3-2.el6.x86_64.rpm                                 | 1.5 MB     00:48
(5/60): erlang-cosEvent-17.3-2.el6.x86_64.rpm                                 | 180 kB     00:05
(6/60): erlang-cosEventDomain-17.3-2.el6.x86_64.rpm                           | 145 kB     00:03
(7/60): erlang-cosFileTransfer-17.3-2.el6.x86_64.rpm                          | 214 kB     00:08
(8/60): erlang-cosNotification-17.3-2.el6.x86_64.rpm                          | 943 kB     00:26
(9/60): erlang-cosProperty-17.3-2.el6.x86_64.rpm                              | 198 kB     00:05
(10/60): erlang-cosTime-17.3-2.el6.x86_64.rpm                                 | 125 kB     00:04
(11/60): erlang-cosTransactions-17.3-2.el6.x86_64.rpm                         | 208 kB     00:10
(12/60): erlang-crypto-17.3-2.el6.x86_64.rpm                                  | 205 kB     00:06
(13/60): erlang-debugger-17.3-2.el6.x86_64.rpm                                | 512 kB     00:19
(14/60): erlang-dialyzer-17.3-2.el6.x86_64.rpm                                | 801 kB     00:08
(15/60): erlang-diameter-17.3-2.el6.x86_64.rpm                                | 892 kB     00:20
(16/60): erlang-edoc-17.3-2.el6.x86_64.rpm                                    | 429 kB     00:19
(17/60): erlang-eldap-17.3-2.el6.x86_64.rpm                                   | 121 kB     00:02
(18/60): erlang-erl_docgen-17.3-2.el6.x86_64.rpm                              | 185 kB     00:04
(19/60): erlang-erl_interface-17.3-2.el6.x86_64.rpm                           | 402 kB     00:14
(20/60): erlang-erts-17.3-2.el6.x86_64.rpm                                    | 3.2 MB     01:34
(21/60): erlang-et-17.3-2.el6.x86_64.rpm                                      | 204 kB     00:21
(22/60): erlang-eunit-17.3-2.el6.x86_64.rpm                                   | 196 kB     00:09
(23/60): erlang-examples-17.3-2.el6.x86_64.rpm                                | 1.2 MB     01:40
(24/60): erlang-gs-17.3-2.el6.x86_64.rpm                                      | 762 kB     00:21
(25/60): erlang-hipe-17.3-2.el6.x86_64.rpm                                    | 3.3 MB     01:08
(26/60): erlang-ic-17.3-2.el6.x86_64.rpm                                      | 1.1 MB     00:18
(27/60): erlang-inets-17.3-2.el6.x86_64.rpm                                   | 1.0 MB     00:19
(28/60): erlang-jinterface-17.3-2.el6.x86_64.rpm                              | 171 kB     00:07
(29/60): erlang-kernel-17.3-2.el6.x86_64.rpm          (36%) 53% [==========================-      ]  27 kB/s | 736 kB     00:23 ETA
```



3.3.3.**安装**RabbitMQ
**上传**rabbitmq-server-3.4.1-1.noarch.rpm**文件到**/usr/local/src/rabbitmq/
**安装：**
rpm -ivh rabbitmq-server-3.4.1-1.noarch.rpm

3.3.4.**启动、停止**
service rabbitmq-server start
service rabbitmq-server stop
service rabbitmq-server restart
3.3.5.**设置开机启动**
chkconfig rabbitmq-server on
3.3.6.**设置配置文件**
cd /etc/rabbitmq
cp /usr/share/doc/rabbitmq-server-3.4.1/rabbitmq.config.example /etc/rabbitmq/

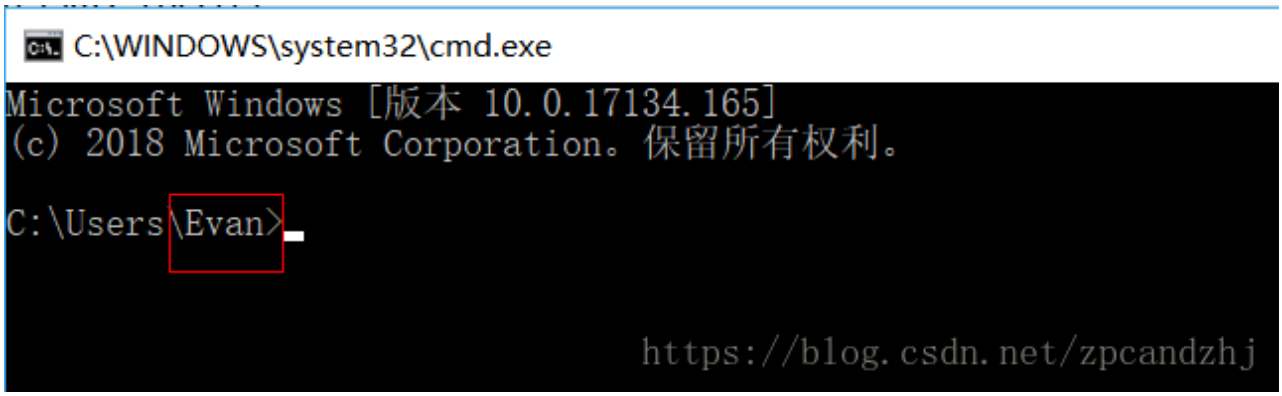
mv rabbitmq.config.example rabbitmq.config
3.3.7.**开启用户远程访问**
vi /etc/rabbitmq/rabbitmq.config




**注意要去掉后面的逗号。**
3.3.8.**开启**web**界面管理工具**
rabbitmq-plugins enable
rabbitmq_management
service rabbitmq-server restart
3.3.9.**防火墙开放**15672**端口**
/sbin/iptables -I INPUT -p tcp --dport 15672 -j ACCEPT
/etc/rc.d/init.d/iptables save

## 3.4.安装的注意事项

1、推荐使用默认的安装路径
2、系统用户名必须是英文
Win10改名字非常麻烦，具体方法百度

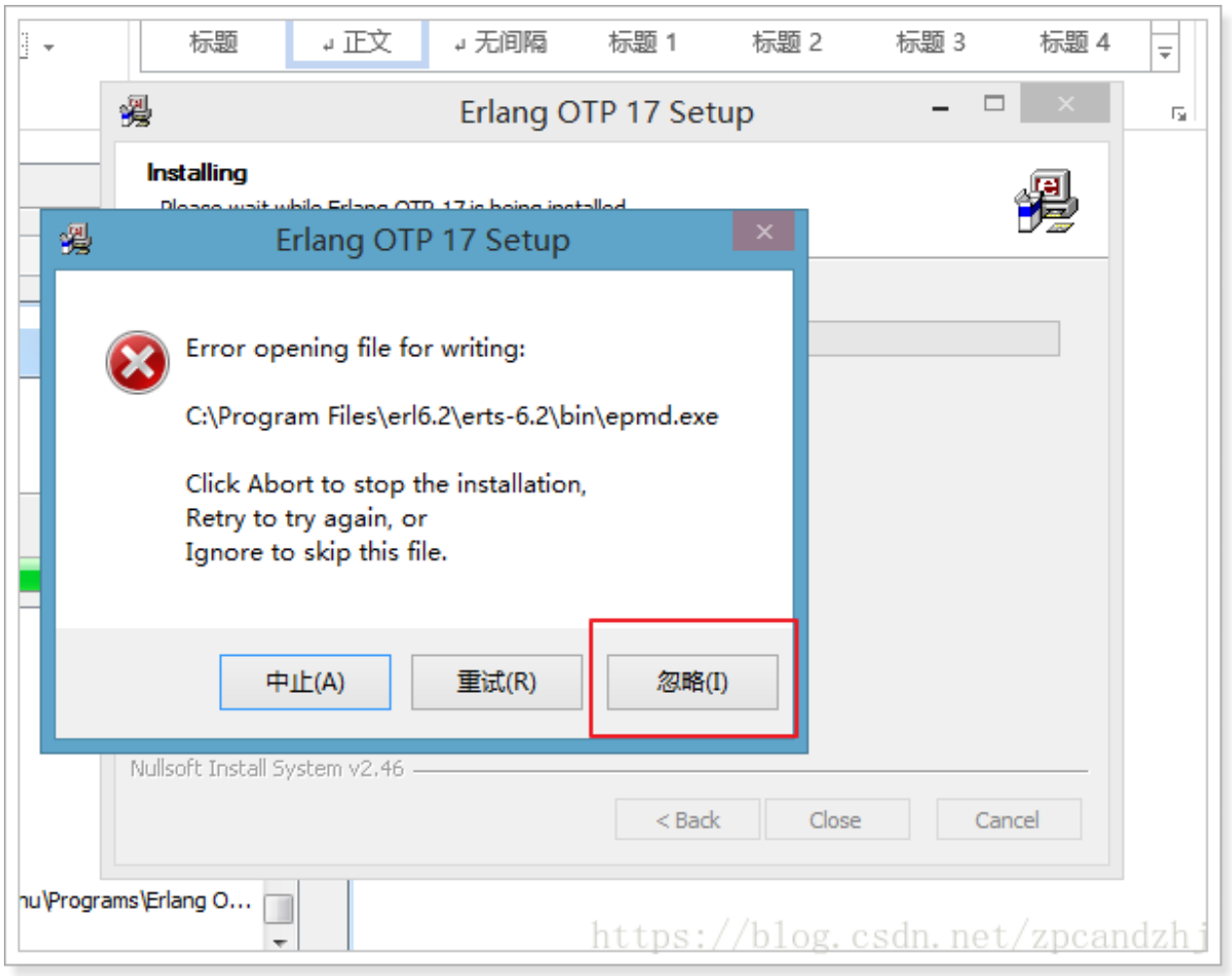



3、计算机名必须是英文


4、系统的用户必须是管理员

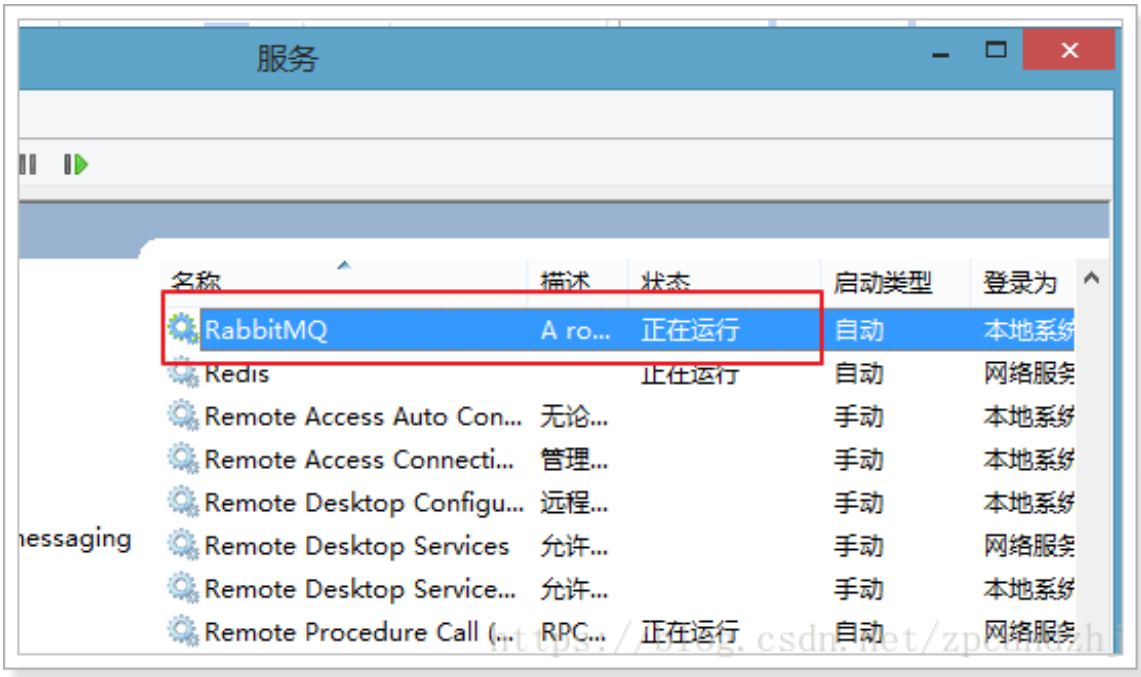
如果安装失败应该如何解决：
1、重装系统 – 不推荐
2、将RabbitMQ安装到linux虚拟机中
a)推荐
3、使用别人安装好的RabbitMQ服务
a)只要给你开通一个账户即可。
b)使用公用的RabbitMQ服务，在192.168.50.22
c)推荐

常见错误：

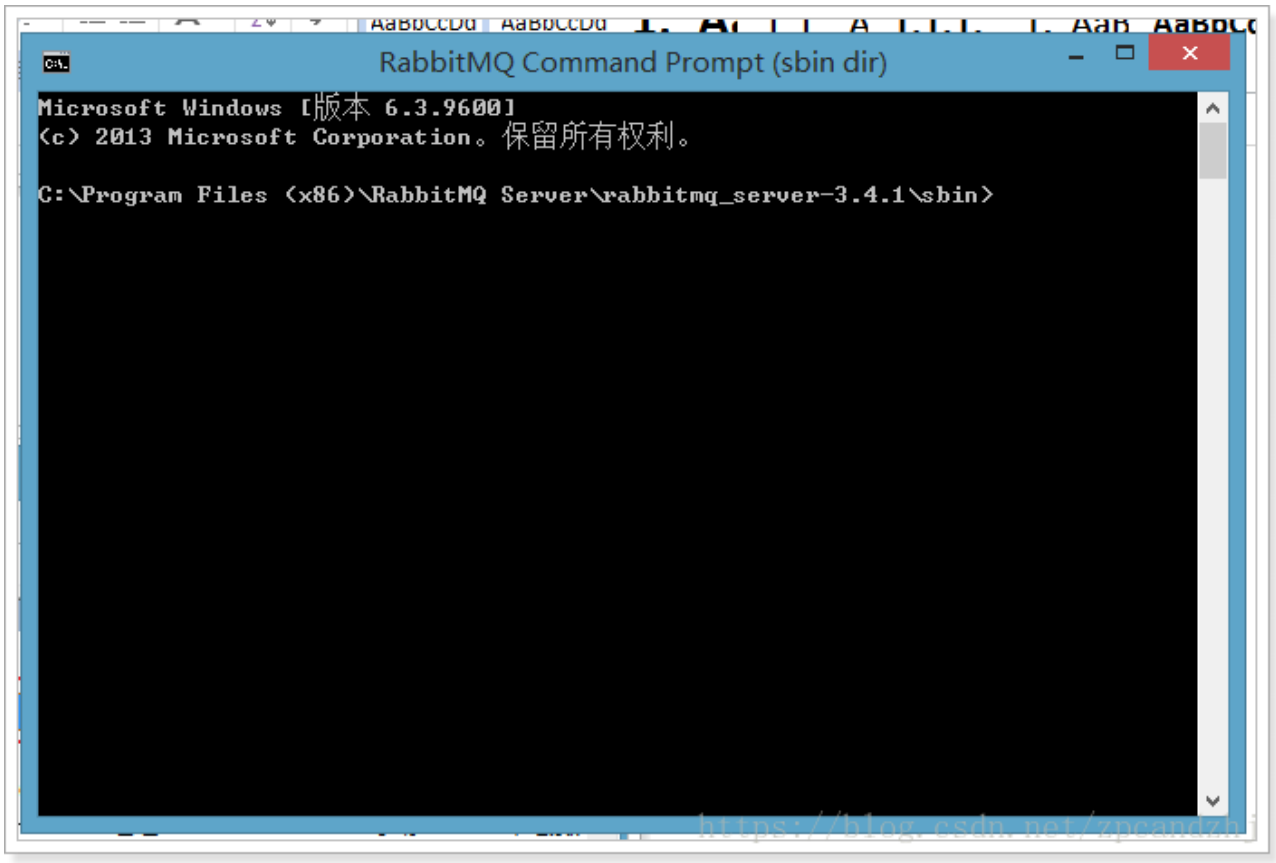



## 3.5.安装完成后操作

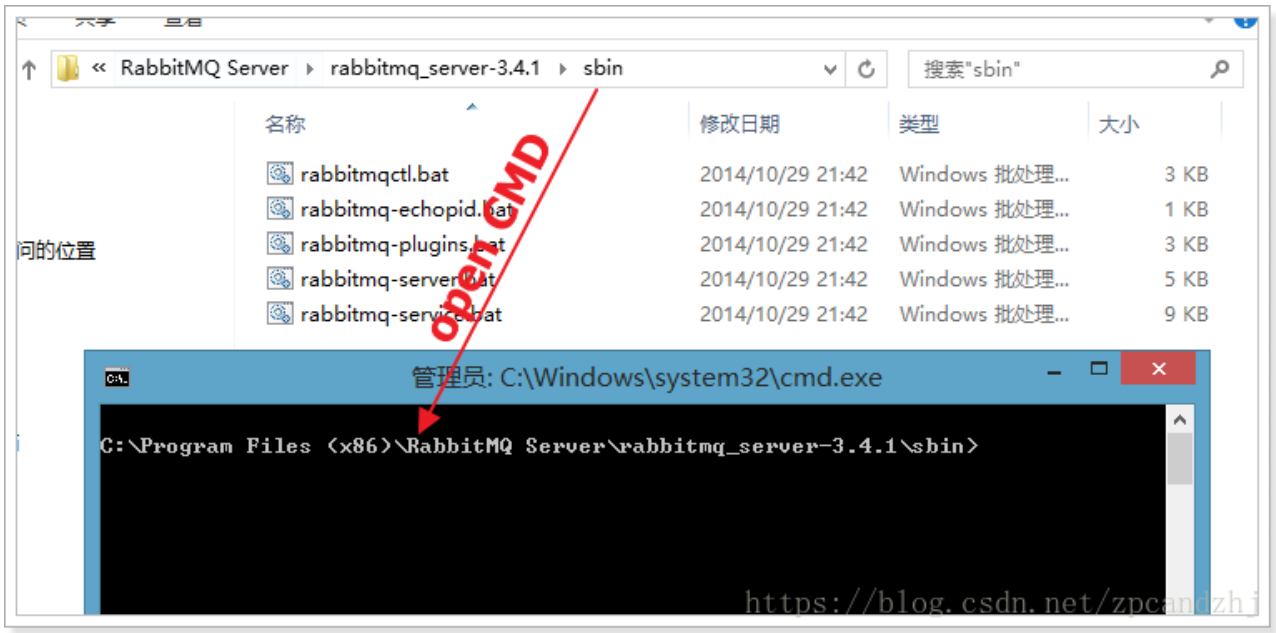
1、系统服务中有RabbitMQ服务，停止、启动、重启



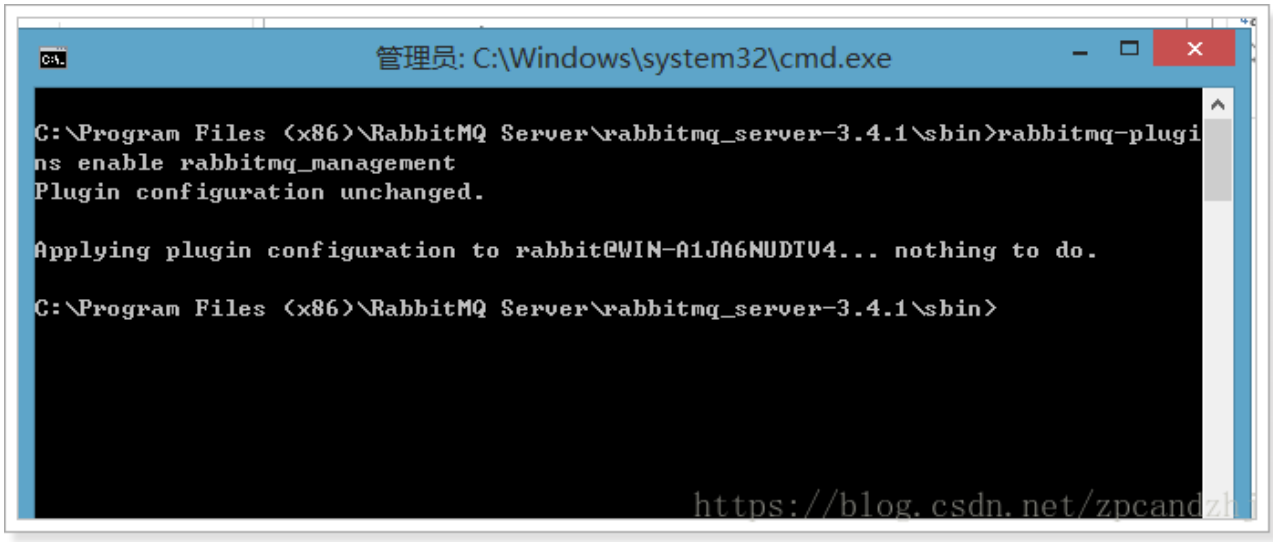
2、打开命令行工具



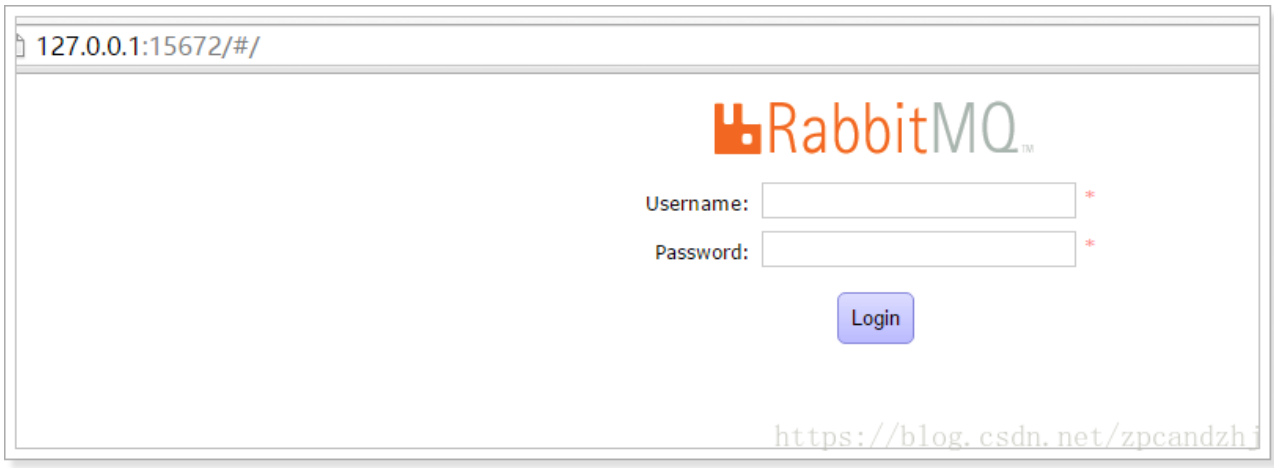

如果找不到命令行工具,直接cd到相应目录：



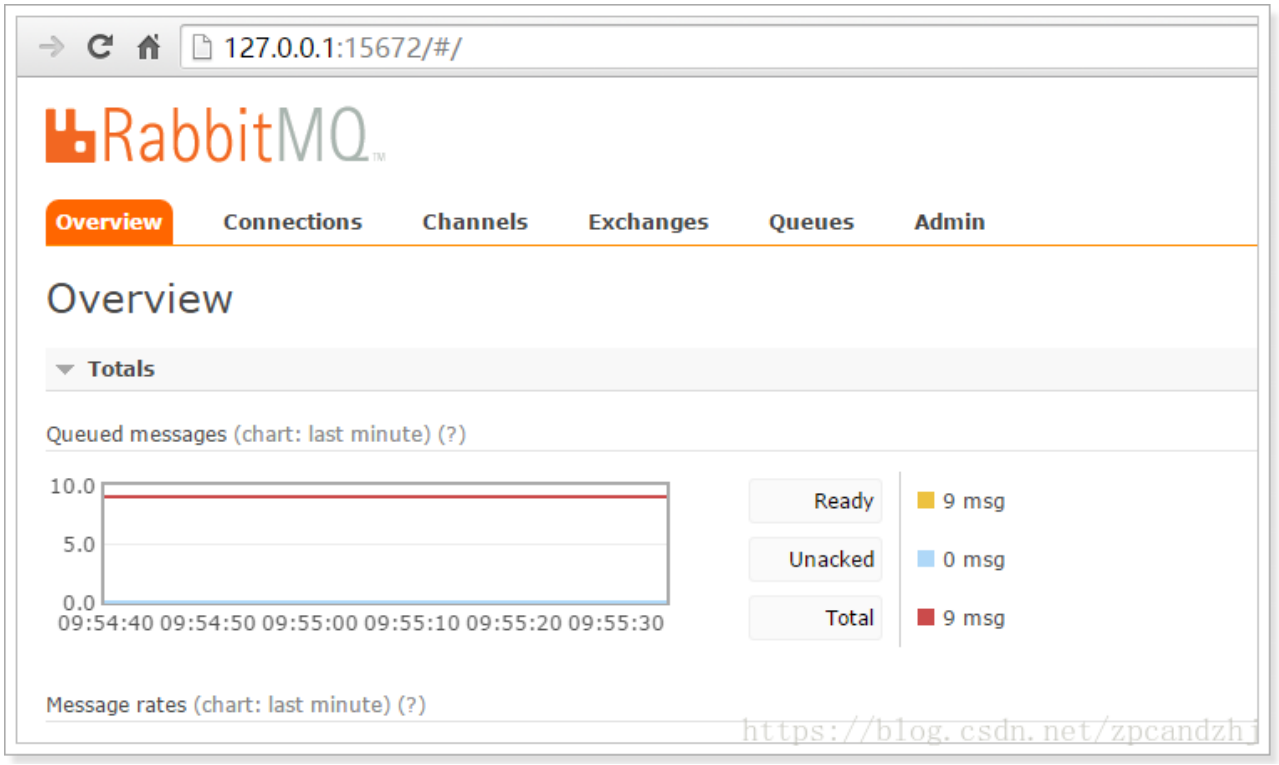

输入命令rabbitmq-plugins enable rabbitmq_management启用管理插件




查看管理页面

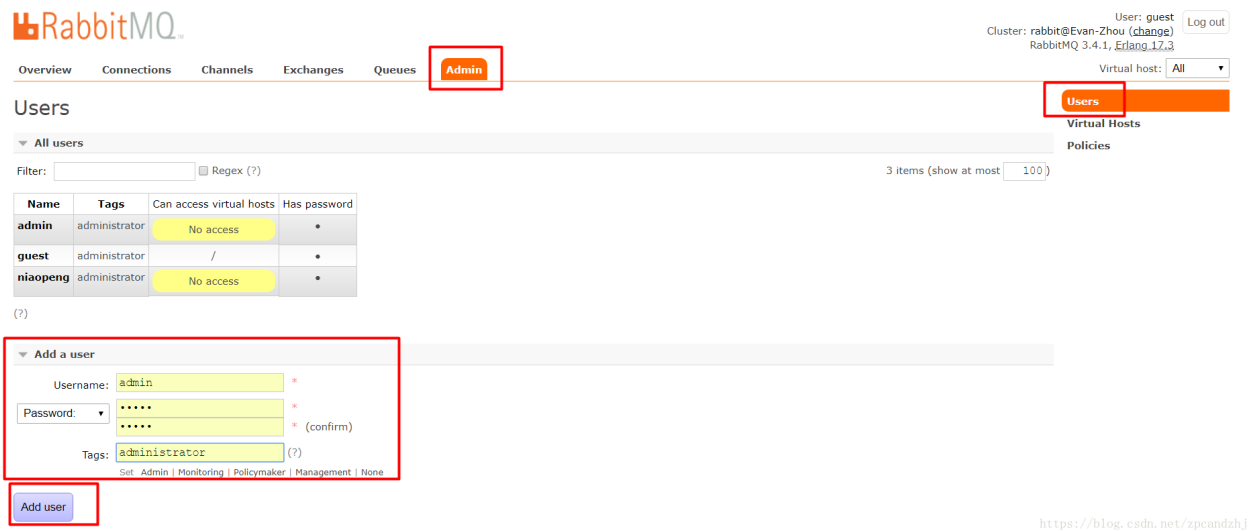



通过默认账户 guest/guest 登录
如果能够登录，说明安装成功。




# 4.添加用户

## 4.1.添加admin用户




## 4.2.用户角色

1、超级管理员(administrator)
可登陆管理控制台，可查看所有的信息，并且可以对用户，策略(policy)进行操作。
2、监控者(monitoring)
可登陆管理控制台，同时可以查看rabbitmq节点的相关信息(进程数，内存使用情况，磁盘使用情况等)
3、策略制定者(policymaker)
可登陆管理控制台, 同时可以对policy进行管理。但无法查看节点的相关信息(上图红框标识的部分)。

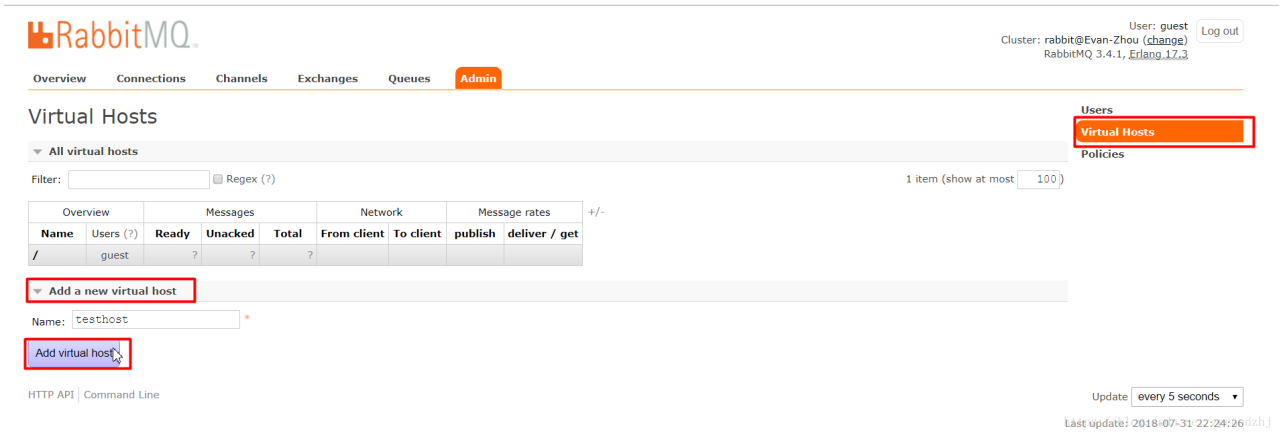
4、普通管理者(management)

仅可登陆管理控制台，无法看到节点信息，也无法对策略进行管理。

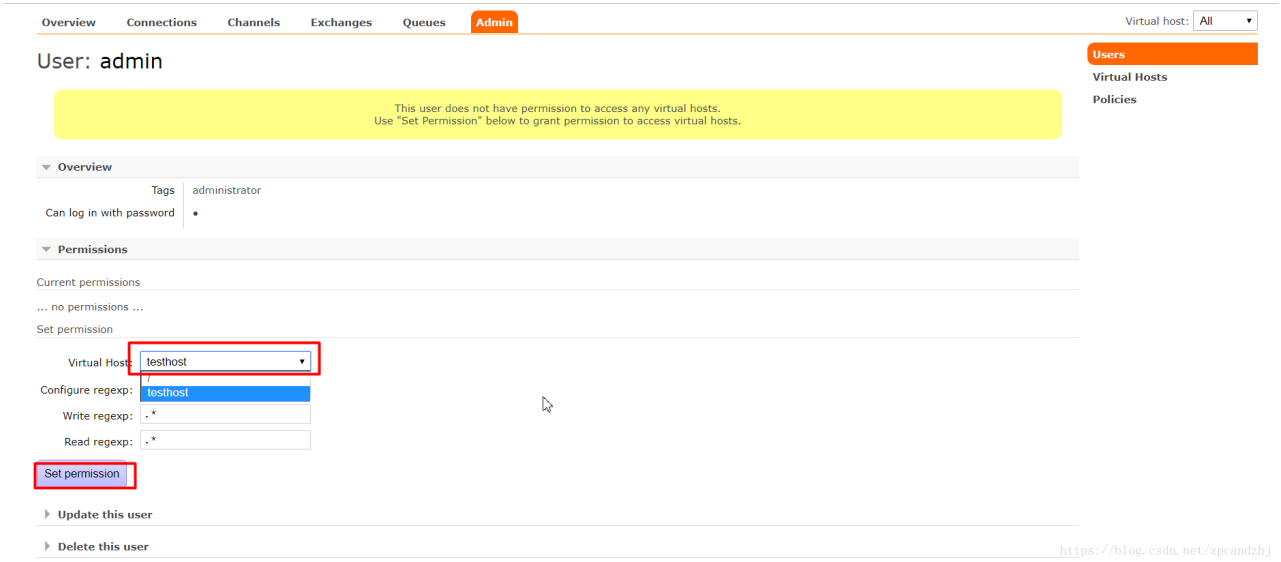
5、其他

无法登陆管理控制台，通常就是普通的生产者和消费者。

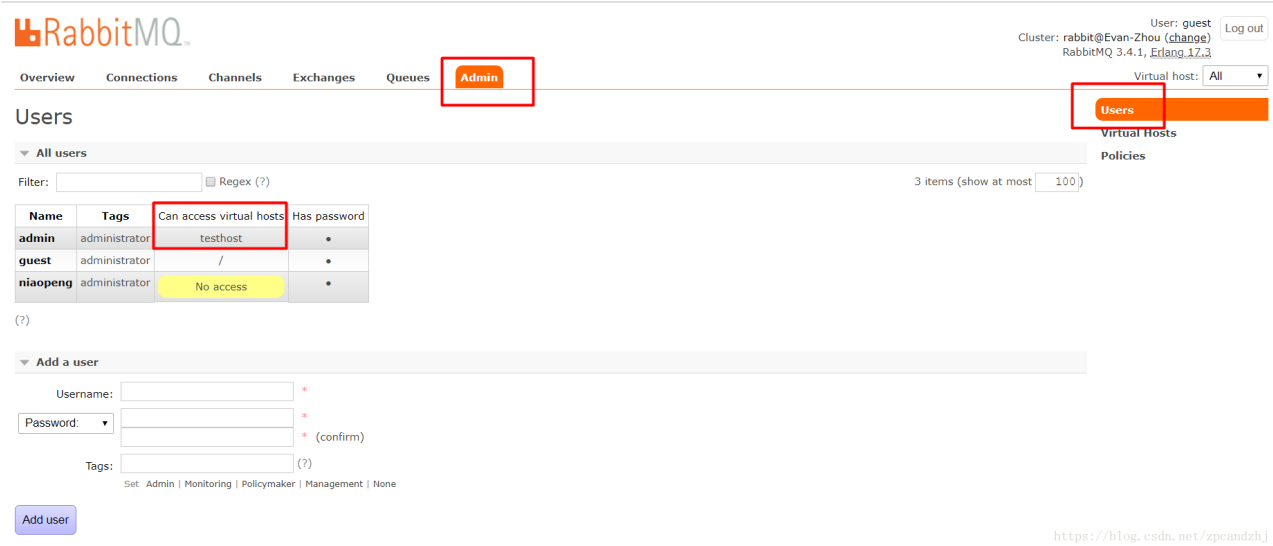
## 4.3.创建Virtual Hosts




选中Admin用户，设置权限：

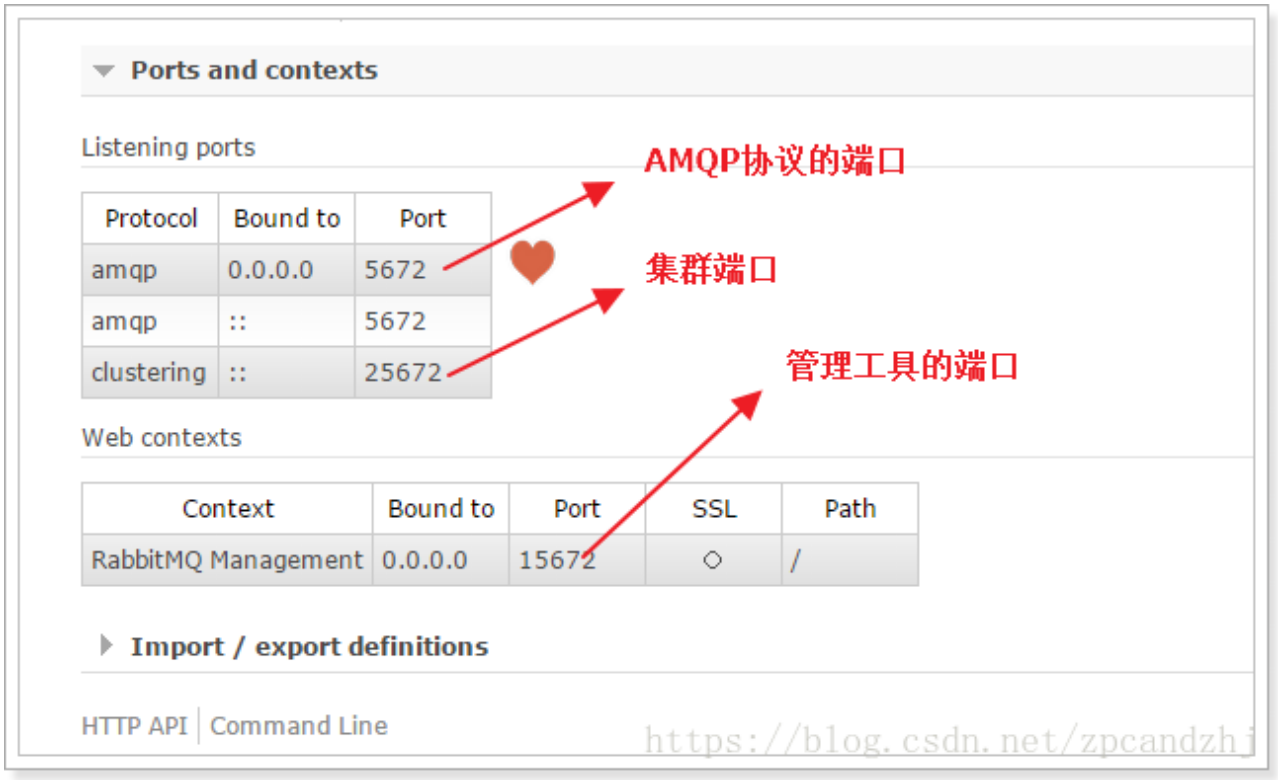



看到权限已加：

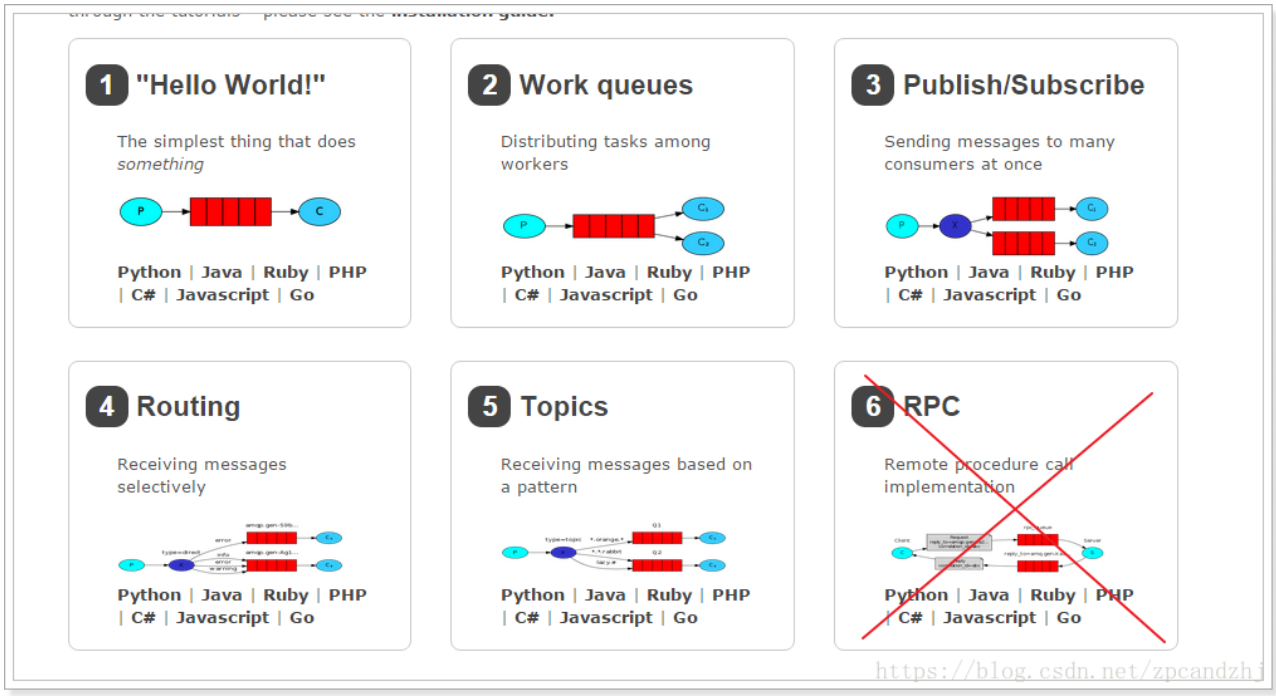

RabbitMQ
User: guest
Cluster: rabbit@Evan-Zhou (change)
RabbitMQ 3.4.1, Erlang 17.3
Log out
Overview
Connections
Channels
Exchanges
Queues
Admin
Virtual host: All
Users
Virtual Hosts
Policies
Users
All users
Filter:
Regex (?)
3 items (show at most 100)
Name
Tags
Can access virtual hosts
Has password
admin
administrator
testhost
guest
administrator
/
niaopeng
administrator
No access
(?)
Add a user
Username:
Password:
(confirm)
Tags:
(?)
Set Admin | Monitoring | Policymaker | Management | None
Add user
https://blog.csdn.net/zpcandzhj

## 4.4.管理界面中的功能


127.0.0.1:15672/#/
RabbitMQ
Overview
Connections
Channels
Exchanges
Queues
Admin
概览信息
连接
通道
交换机
队列
管理
Overview
Totals
Queued messages (chart: last minute) (?)
1.0
0.0
10:18:50 10:19:00 10:19:10 10:19:20 10:19:30 10:19:40
Ready
0 msg
Unacked
0 msg
Total
0 msg
Message rates (chart: last minute) (?)
Currently idle
https://blog.csdn.net/zpcandzhj

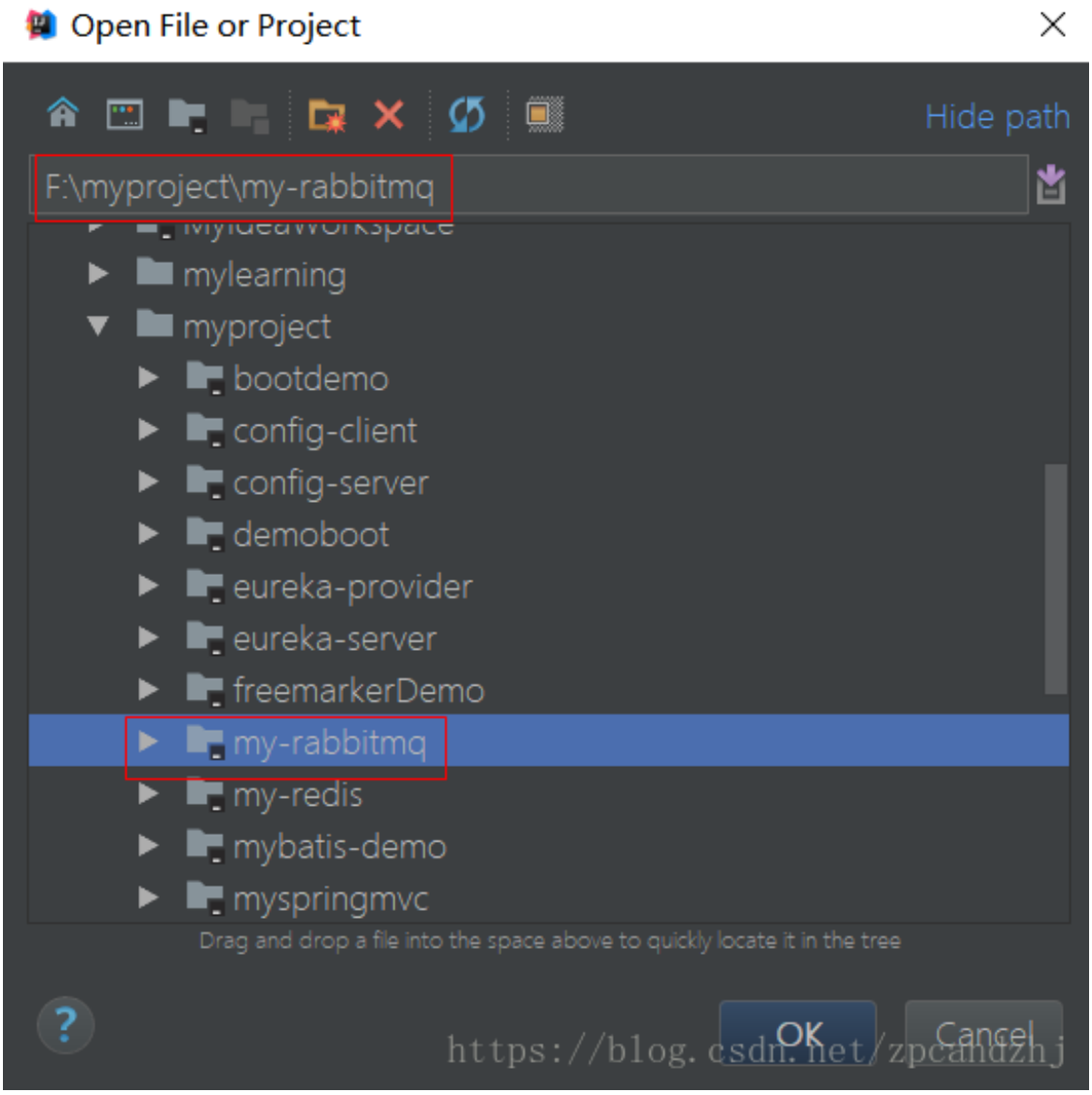



## 5.学习五种队列




### 5.1.导入my-rabbitmq项目

**项目下载地址：**
https://download.csdn.net/download/zpcandzhj/10585077

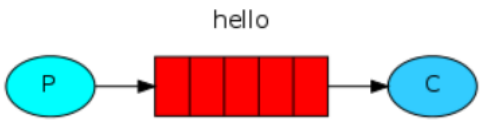



## 5.2.简单队列

### 5.2.1.图示

In the diagram below, "P" is our producer and "C" is our consumer. The box in the middle is a queue - a message buffer that RabbitMQ keeps on behalf of the consumer.

Our overall design will look like:




Producer sends messages to the "hello" queue. The consumer receives messages from that queue.



P：消息的生产者
C：消息的消费者
红色：队列

**生产者将消息发送到队列，消费者从队列中获取消息。**

### 5.2.2.导入RabbitMQ的客户端依赖

```
<dependency>
  <groupId>com.rabbitmq</groupId>
  <artifactId>amqp-client</artifactId>
  <version>3.4.1</version>
</dependency>
```

- 1
- 2
- 3
- 4
- 5

### 5.2.3.获取MQ的连接

```
package com.zpc.rabbitmq.util;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;

public class ConnectionUtil {

    public static Connection getConnection() throws Exception {
        //定义连接工厂
        ConnectionFactory factory = new ConnectionFactory();
        //设置服务地址
        factory.setHost("localhost");
        //端口
        factory.setPort(5672);
        //设置账号信息，用户名、密码、vhost
        factory.setVirtualHost("testhost");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        return connection;
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22

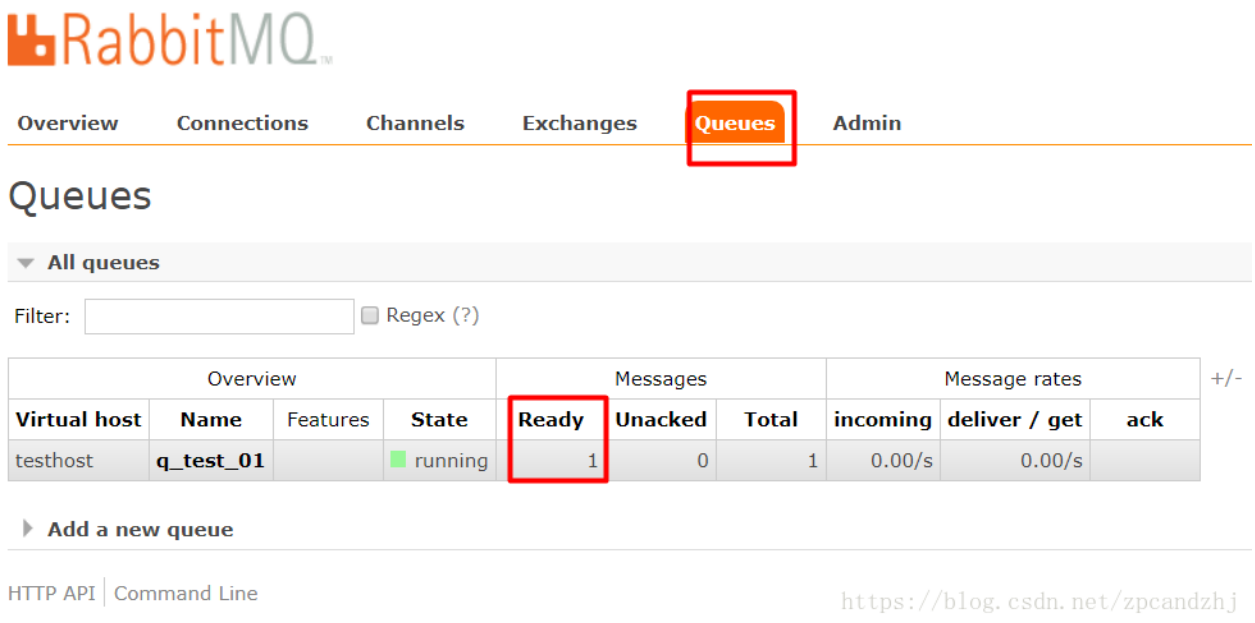
### 5.2.4.生产者发送消息到队列

```
package com.zpc.rabbitmq.simple;

import com.zpc.rabbitmq.util.ConnectionUtil;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
```

```
public class Send {

    private final static String QUEUE_NAME = "q_test_01";

    public static void main(String[] argv) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        // 从连接中创建通道
        Channel channel = connection.createChannel();

        // 声明（创建）队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 消息内容
        String message = "Hello World!";
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
        System.out.println(" [x] Sent '" + message + "'");
        //关闭通道和连接
        channel.close();
        connection.close();
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29

### 5.2.5.管理工具中查看消息




点击上面的队列名称，查询具体的队列中的信息：


### 5.2.6.消费者从队列中获取消息

package com.zpc.rabbitmq.simple;

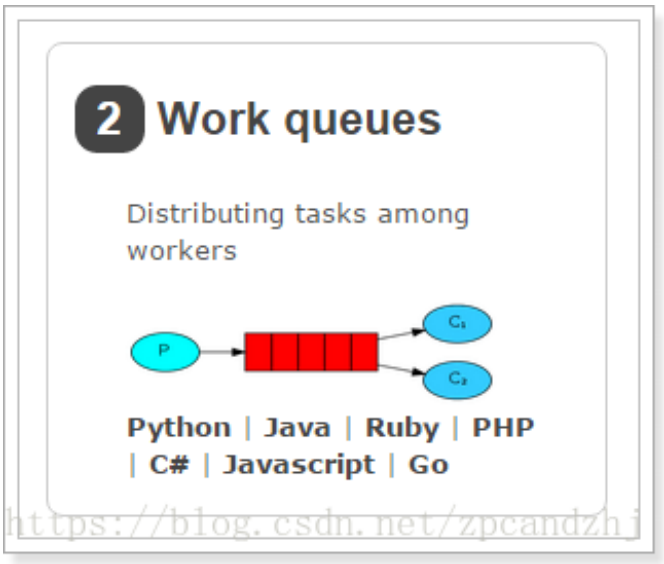
```
import com.zpc.rabbitmq.util.ConnectionUtil;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.QueueingConsumer;

public class Recv {

    private final static String QUEUE_NAME = "q_test_01";

    public static void main(String[] argv) throws Exception {

        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        // 从连接中创建通道
        Channel channel = connection.createChannel();
        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);

        // 监听队列
        channel.basicConsume(QUEUE_NAME, true, consumer);

        // 获取消息
        while (true) {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            System.out.println(" [x] Received '" + message + "'");
        }
    }
}
```

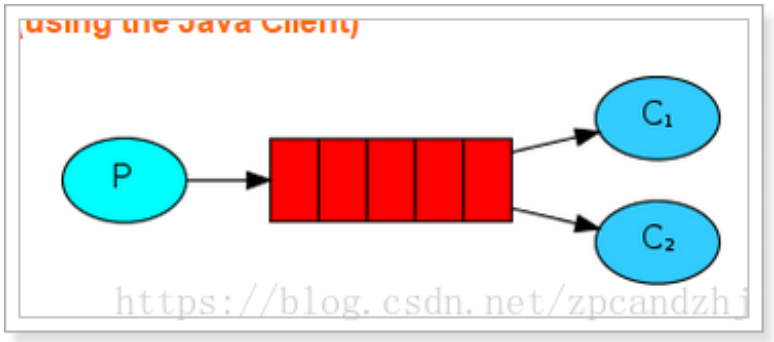
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21

- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35

## 5.3.Work模式

5.3.1.图示




一个生产者、2个消费者。

一个消息只能被一个消费者获取。

5.3.2.消费者1

```
package com.zpc.rabbitmq.work;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
```

```
import com.rabbitmq.client.QueueingConsumer;
import com.zpc.rabbitmq.util.ConnectionUtil;

public class Recv {

    private final static String QUEUE_NAME = "test_queue_work";

    public static void main(String[] argv) throws Exception {

        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 同一时刻服务器只会发一条消息给消费者
        //channel.basicQos(1);

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);
        // 监听队列，false表示手动返回完成状态，true表示自动
        channel.basicConsume(QUEUE_NAME, true, consumer);

        // 获取消息
        while (true) {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            System.out.println(" [y] Received '" + message + "'");
            //休眠
            Thread.sleep(10);
            // 返回确认状态，注释掉表示使用自动确认模式
            //channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
        }
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18

- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40

### 5.3.3.消费者2

```
package com.zpc.rabbitmq.work;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.QueueingConsumer;
import com.zpc.rabbitmq.util.ConnectionUtil;

public class Recv2 {

    private final static String QUEUE_NAME = "test_queue_work";

    public static void main(String[] argv) throws Exception {

        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 同一时刻服务器只会发一条消息给消费者
        //channel.basicQos(1);

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);
        // 监听队列，false表示手动返回完成状态，true表示自动
        channel.basicConsume(QUEUE_NAME, true, consumer);
```

```
        // 获取消息
        while (true) {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            System.out.println(" [x] Received '" + message + "'");
            // 休眠1秒
            Thread.sleep(1000);
            //下面这行注释掉表示使用自动确认模式
            //channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
        }
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40

### 5.3.4.生产者

**向队列中发送100条消息。**

```
package com.zpc.rabbitmq.work;

import com.zpc.rabbitmq.util.ConnectionUtil;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;

public class Send {

    private final static String QUEUE_NAME = "test_queue_work";

    public static void main(String[] argv) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        for (int i = 0; i < 100; i++) {
            // 消息内容
            String message = "" + i;
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
            System.out.println(" [x] Sent '" + message + "'");

            Thread.sleep(i * 10);
        }

        channel.close();
        connection.close();
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19

- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32

5.3.5.测试
测试结果：
1、消费者1和消费者2获取到的消息内容是不同的，同一个消息只能被一个消费者获取。
2、消费者1和消费者2获取到的消息的数量是相同的，一个是消费奇数号消息，一个是偶数。

- 其实，这样是不合理的，因为消费者1线程停顿的时间短。应该是消费者1要比消费者2获取到的消息多才对。
  RabbitMQ 默认将消息顺序发送给下一个消费者，这样，每个消费者会得到相同数量的消息。即轮询（round-robin）分发消息。

- 怎样才能做到按照每个消费者的能力分配消息呢？联合使用 Qos 和 Acknowledge 就可以做到。
  basicQos 方法设置了当前信道最大预获取（prefetch）消息数量为1。消息从队列异步推送给消费者，消费者的 ack 也是异步发送给队列，从队列的视角去看，总是会有一批消息已推送但尚未获得 ack 确认，Qos 的 prefetchCount 参数就是用来限制这批未确认消息数量的。设为1时，队列只有在收到消费者发回的上一条消息 ack 确认后，才会向该消费者发送下一条消息。prefetchCount 的默认值为0，即没有限制，队列会将所有消息尽快发给消费者。

- 2个概念

- 轮询分发 ：使用任务队列的优点之一就是可以轻易的并行工作。如果我们积压了好多工作，我们可以通过增加工作者（消费者）来解决这一问题，使得系统的伸缩性更加容易。在默认情况下，RabbitMQ将逐个发送消息到在序列中的下一个消费者(而不考虑每个任务的时长等等，且是提前一次性分配，并非一个一个分配)。平均每个消费者获得相同数量的消息。这种方式分发消息机制称为Round-Robin（轮询）。

- 公平分发 ：虽然上面的分配法方式也还行，但是有个问题就是：比如：现在有2个消费者，所有的奇数的消息都是繁忙的，而偶数则是轻松的。按照轮询的方式，奇数的任务交给了第一个消费者，所以一直在忙个不停。偶数的任务交给另一个消费者，则立即完成任务，然后闲得不行。而RabbitMQ则是不了解这些的。这是因为当消息进入队列，RabbitMQ就会分派消息。它不看消费者为应答的数目，只是盲目的将消息发给轮询指定的消费者。

为了解决这个问题，我们使用basicQos( prefetchCount = 1)方法，来限制RabbitMQ只发不超过1条的消息给同一个消费者。当消息处理完毕后，有了反馈，才会进行第二次发送。
还有一点需要注意，使用公平分发，必须关闭自动应答，改为手动应答。

## 5.4.Work模式的“能者多劳”

打开上述代码的注释：

```
// 同一时刻服务器只会发一条消息给消费者
channel.basicQos(1);
```

- 1
- 2

```
//开启这行 表示使用手动确认模式
channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
```

- 1
- 2

同时改为手动确认：

```
// 监听队列，false表示手动返回完成状态，true表示自动
channel.basicConsume(QUEUE_NAME, false, consumer);
```

- 1
- 2

测试：
消费者1比消费者2获取的消息更多。

## 5.5.消息的确认模式

消费者从队列中获取消息，服务端如何知道消息已经被消费呢？

模式1：自动确认
只要消息从队列中获取，无论消费者获取到消息后是否成功消息，都认为是消息已经成功消费。
模式2：手动确认
消费者从队列中获取消息后，服务器会将该消息标记为不可用状态，等待消费者的反馈，如果消费者一直没有反馈，那么该消息将一直处于不可用状态。

手动模式：

```java
// 定义队列的消费者
QueueingConsumer consumer = new QueueingConsumer(channel);
// 监听队列，手动返回完成状态
channel.basicConsume(QUEUE_NAME, false, consumer);

// 获取消息
while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
    String message = new String(delivery.getBody());
    System.out.println(" [x] Received '" + message + "'");
    // 休眠1秒
    Thread.sleep(1000);

    //反馈消息的消费状态
    channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
}
```

true：自动确认
false：手动确认



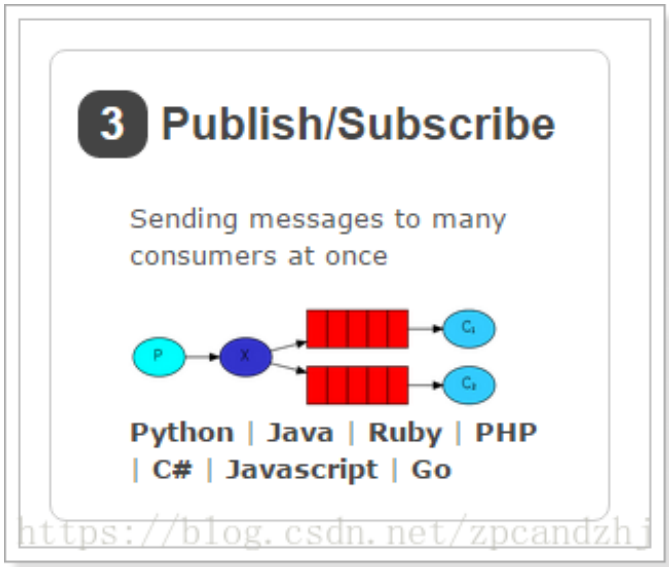
**自动模式：**

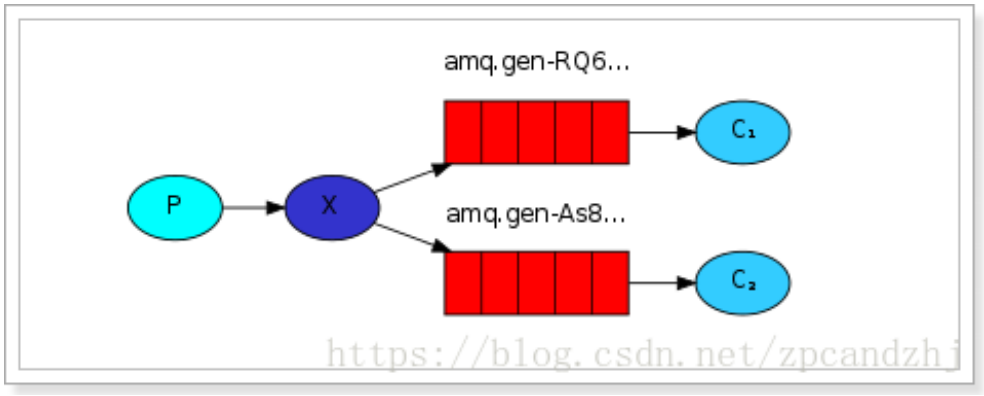
```java
// 定义队列的消费者
QueueingConsumer consumer = new QueueingConsumer(channel);
// 监听队列
channel.basicConsume(QUEUE_NAME, true, consumer);

// 获取消息
while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
    String message = new String(delivery.getBody());
    System.out.println(" [x] Received '" + message + "'");
}
```



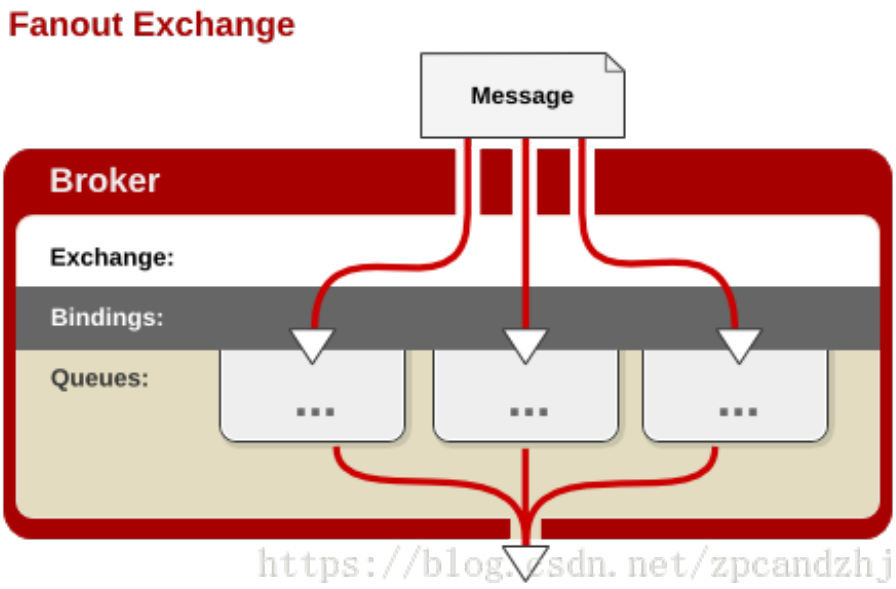
## 5.6.订阅模式

### 5.6.1.图示






解读：
1、1个生产者，多个消费者
2、每一个消费者都有自己的一个队列
3、生产者没有将消息直接发送到队列，而是发送到了交换机
4、每个队列都要绑定到交换机
5、生产者发送的消息，经过交换机，到达队列，实现，一个消息被多个消费者获取的目的
注意：一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费




5.6.2.消息的生产者（看作是后台系统）
向交换机中发送消息。

```
package com.zpc.rabbitmq.subscribe;

import com.zpc.rabbitmq.util.ConnectionUtil;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;

public class Send {

    private final static String EXCHANGE_NAME = "test_exchange_fanout";

    public static void main(String[] argv) throws Exception {
```

```
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明exchange
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

        // 消息内容
        String message = "Hello World!";
        channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
        System.out.println(" [x] Sent '" + message + "'");

        channel.close();
        connection.close();
    }
}
```

**注意：消息发送到没有队列绑定的交换机时，消息将丢失，因为，交换机没有存储消息的能力，消息只能存在在队列中。**

### 5.6.3.消费者1（看作是前台系统）

```
package com.zpc.rabbitmq.subscribe;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
```

```
import com.rabbitmq.client.QueueingConsumer;

import com.zpc.rabbitmq.util.ConnectionUtil;

public class Recv {

    private final static String QUEUE_NAME = "test_queue_work1";

    private final static String EXCHANGE_NAME = "test_exchange_fanout";

    public static void main(String[] argv) throws Exception {

        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定队列到交换机
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");

        // 同一时刻服务器只会发一条消息给消费者
        channel.basicQos(1);

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);
        // 监听队列，手动返回完成
        channel.basicConsume(QUEUE_NAME, false, consumer);

        // 获取消息
        while (true) {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            System.out.println(" [Recv] Received '" + message + "'");
            Thread.sleep(10);

            channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
        }
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13

- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45

### 5.6.4.消费者2（看作是搜索系统）

```
package com.zpc.rabbitmq.subscribe;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.QueueingConsumer;

import com.zpc.rabbitmq.util.ConnectionUtil;

public class Recv2 {

    private final static String QUEUE_NAME = "test_queue_work2";

    private final static String EXCHANGE_NAME = "test_exchange_fanout";

    public static void main(String[] argv) throws Exception {

        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
```

```
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定队列到交换机
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");

        // 同一时刻服务器只会发一条消息给消费者
        channel.basicQos(1);

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);
        // 监听队列，手动返回完成
        channel.basicConsume(QUEUE_NAME, false, consumer);

        // 获取消息
        while (true) {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            System.out.println(" [Recv2] Received '" + message + "'");
            Thread.sleep(10);

            channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
        }
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27

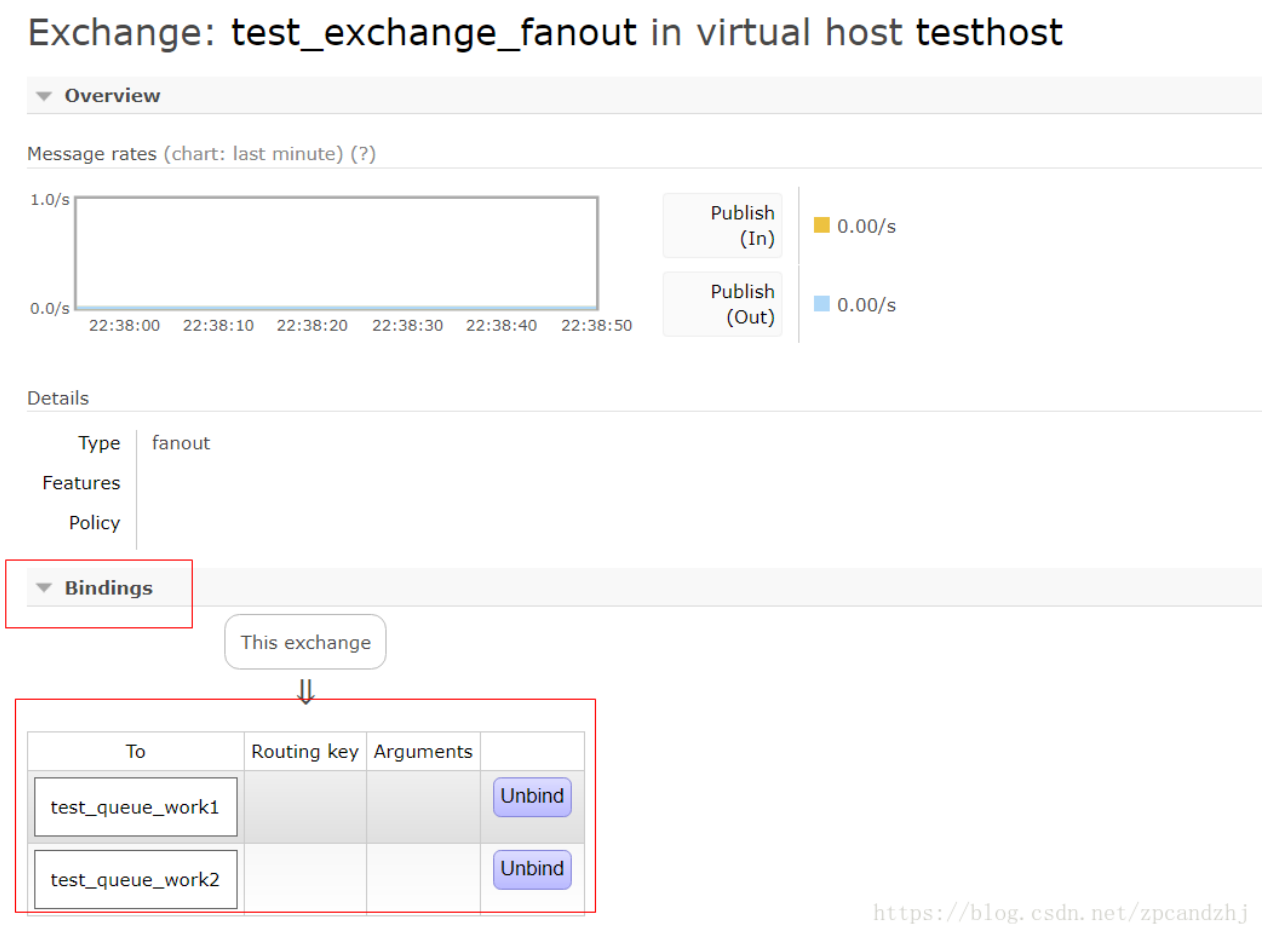
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45

5.6.5.测试

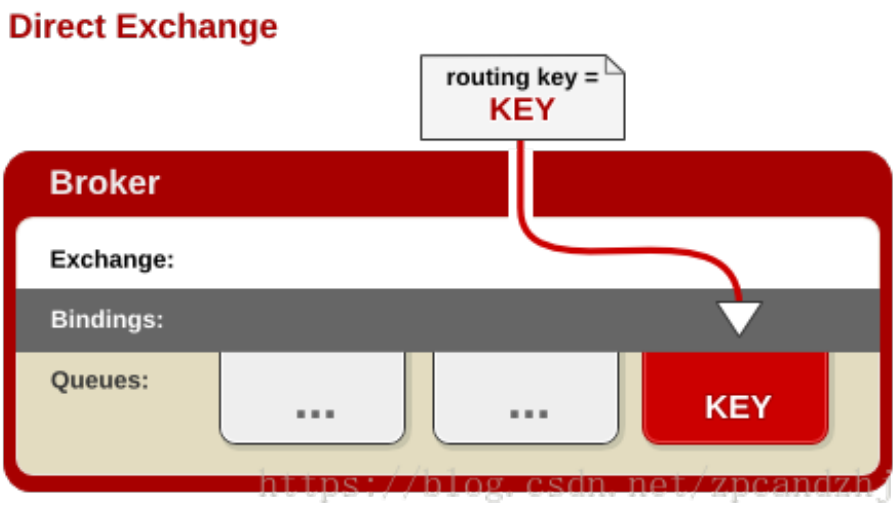
**测试结果：**

**同一个消息被多个消费者获取。一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费到消息。**

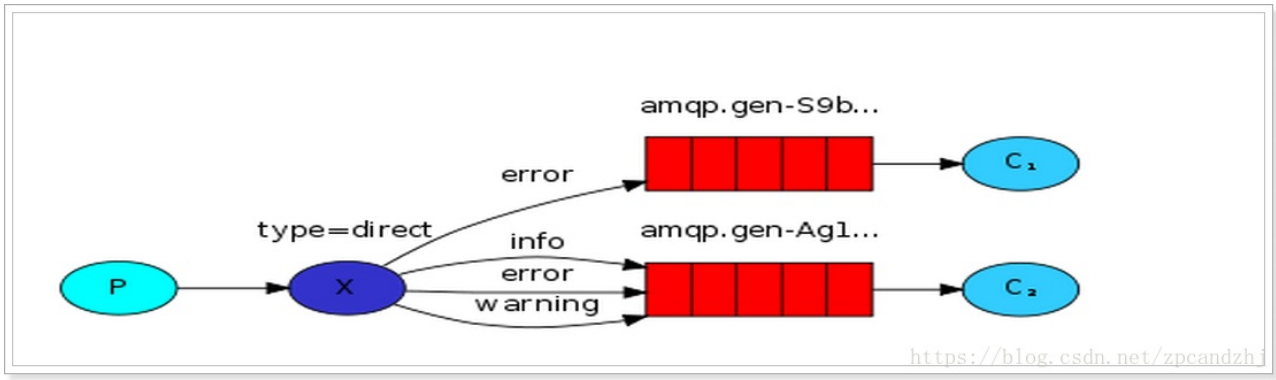
**在管理工具中查看队列和交换机的绑定关系：**



## 5.7.路由模式




### 5.7.1.图示




### 5.7.2.生产者

```
private final static String EXCHANGE_NAME = "test_exchange_direct";

public static void main(String[] argv) throws Exception {
    // 获取到连接以及mq通道
    Connection connection = ConnectionUtil.getConnection();
    Channel channel = connection.createChannel();

    // 声明exchange
    channel.exchangeDeclare(EXCHANGE_NAME, "direct");

    // 消息内容
    String message = "删除商品";
    channel.basicPublish(EXCHANGE_NAME, "delete", null, message.getBytes());
    System.out.println(" [x] Sent '" + message + "'");

    channel.close();
    connection.close();
}
}
```

交换机类型

消息KEY



### 5.7.3.消费者1(假设是前台系统)

```java
private final static String QUEUE_NAME = "test_queue_direct_1";

private final static String EXCHANGE_NAME = "test_exchange_direct";

public static void main(String[] argv) throws Exception {

    // 获取到连接以及mq通道
    Connection connection = ConnectionUtil.getConnection();
    Channel channel = connection.createChannel();

    // 声明队列
    channel.queueDeclare(QUEUE_NAME, false, false, false, null);

    // 绑定队列到交换机
    channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "update");
    channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "delete");

    // 同一时刻服务器只会发一条消息给消费者
    channel.basicQos(1);

    // 定义队列的消费者
    QueueingConsumer consumer = new QueueingConsumer(channel);
```



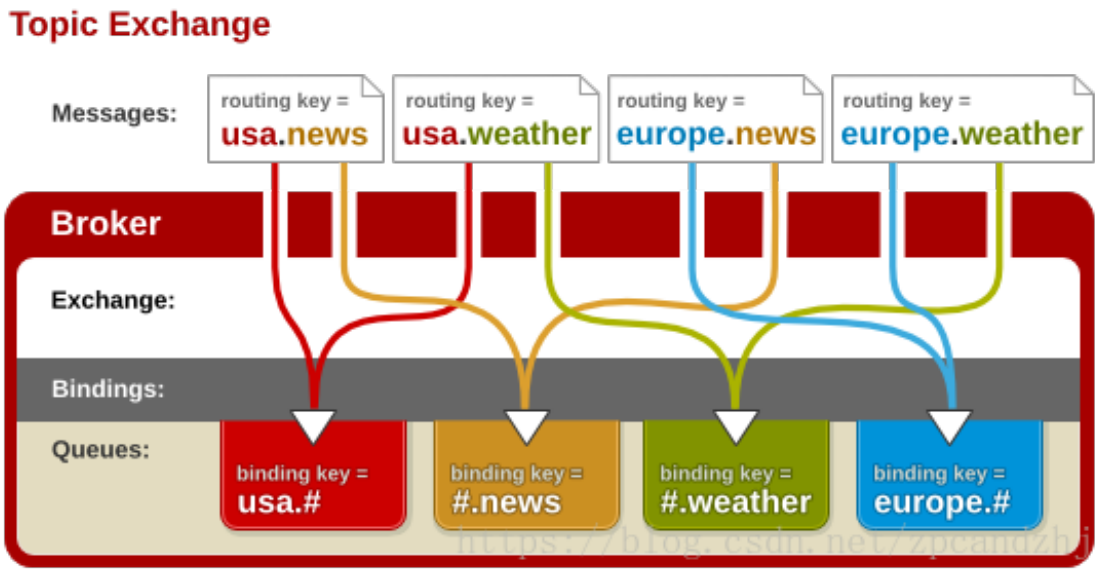
### 5.7.4.消费2（假设是搜索系统）

```java
private final static String QUEUE_NAME = "test_queue_direct_2";

private final static String EXCHANGE_NAME = "test_exchange_direct";

public static void main(String[] argv) throws Exception {

    // 获取到连接以及mq通道
    Connection connection = ConnectionUtil.getConnection();
    Channel channel = connection.createChannel();

    // 声明队列
    channel.queueDeclare(QUEUE_NAME, false, false, false, null);

    // 绑定队列到交换机
    channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "insert");
    channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "update");
    channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "delete");

    // 同一时刻服务器只会发一条消息给消费者
    channel.basicQos(1);

    // 定义队列的消费者
```

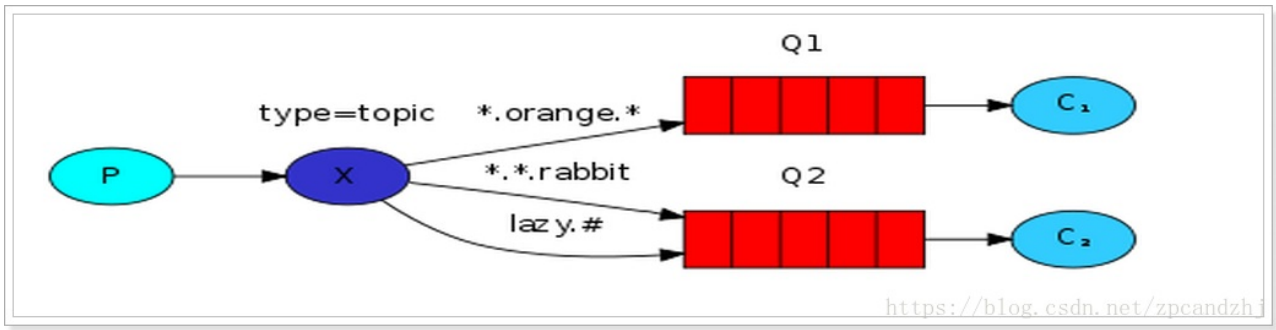


## 5.8.主题模式（通配符模式）

**Topic Exchange** – 将路由键和某模式进行匹配。此时队列需要绑定要一个模式上。符号“#”匹配一个或多个词，符号“*”匹配不多不少一个词。因此“audit.#”能够匹配到“audit.irs.corporate”，但是“audit.*” 只会匹配到“audit.irs”。我在 RedHat 的朋友做了一张不错的图，来表明 topic 交换机是如何工作的：




5.8.1.图示




**同一个消息被多个消费者获取。一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费到消息。**

5.8.2.生产者

```
package com.zpc.rabbitmq.topic;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;

import com.zpc.rabbitmq.util.ConnectionUtil;

public class Send {

    private final static String EXCHANGE_NAME = "test_exchange_topic";

    public static void main(String[] argv) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明exchange
        channel.exchangeDeclare(EXCHANGE_NAME, "topic");
```

```
        // 消息内容
        String message = "Hello World!!";
        channel.basicPublish(EXCHANGE_NAME, "routekey.1", null, message.getBytes());
        System.out.println(" [x] Sent '" + message + "'");

        channel.close();
        connection.close();
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28

### 5.8.3.消费者1（前台系统）

```
package com.zpc.rabbitmq.topic;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.QueueingConsumer;

import com.zpc.rabbitmq.util.ConnectionUtil;

public class Recv {

    private final static String QUEUE_NAME = "test_queue_topic_work_1";

    private final static String EXCHANGE_NAME = "test_exchange_topic";
```

```
    public static void main(String[] argv) throws Exception {

        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定队列到交换机
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "routekey.*");

        // 同一时刻服务器只会发一条消息给消费者
        channel.basicQos(1);

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);
        // 监听队列，手动返回完成
        channel.basicConsume(QUEUE_NAME, false, consumer);

        // 获取消息
        while (true) {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            System.out.println(" [Recv_x] Received '" + message + "'");
            Thread.sleep(10);

            channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
        }
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22

- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45

### 5.8.4.消费者2（搜索系统）

```
package com.zpc.rabbitmq.topic;

import com.zpc.rabbitmq.util.ConnectionUtil;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.QueueingConsumer;

public class Recv2 {

    private final static String QUEUE_NAME = "test_queue_topic_work_2";

    private final static String EXCHANGE_NAME = "test_exchange_topic";

    public static void main(String[] argv) throws Exception {

        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定队列到交换机
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "*.*");

        // 同一时刻服务器只会发一条消息给消费者
```

```
        channel.basicQos(1);

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);
        // 监听队列，手动返回完成
        channel.basicConsume(QUEUE_NAME, false, consumer);

        // 获取消息
        while (true) {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            System.out.println(" [Recv2_x] Received '" + message + "'");
            Thread.sleep(10);

            channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
        }
    }
}
```

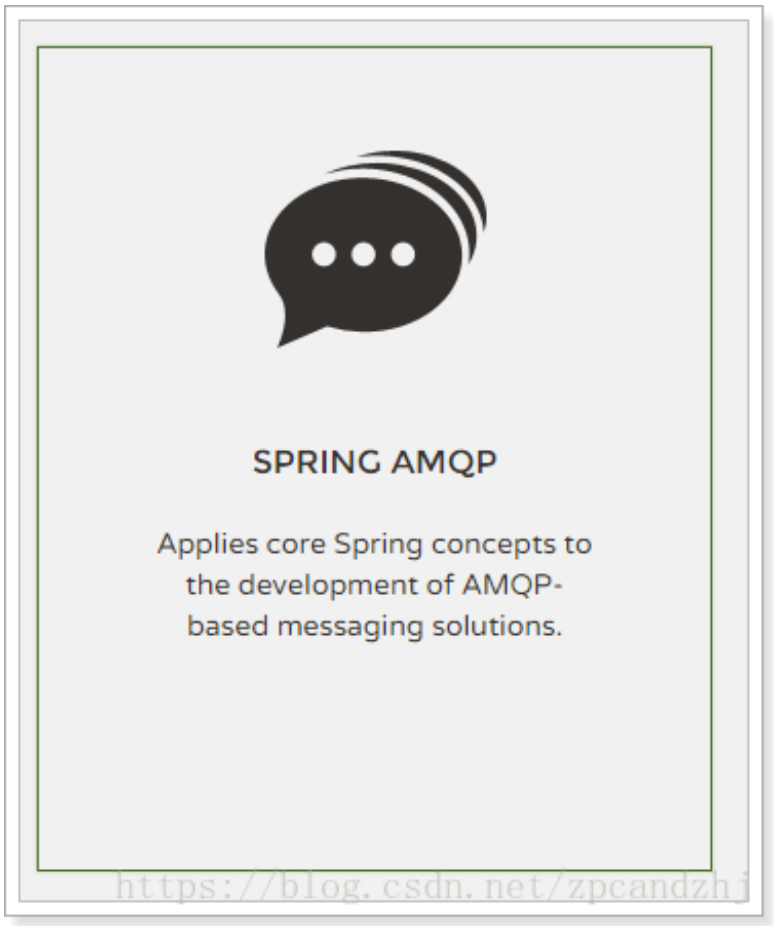
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36

- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45

# 6.Spring-Rabbit

## 6.1.Spring项目

http://spring.io/projects




## 6.2.简介

The project consists of two parts; `spring-amqp` is the base abstraction, and `spring-rabbit` is the RabbitMQ implementation.

### Features

- Listener container for asynchronous processing of inbound messages
- RabbitTemplate for sending and receiving messages
- RabbitAdmin for automatically declaring queues, exchanges and bindings



# Spring-AMQP

- Spring对AMQP做了支持，目前只是做了RabbitMQ的实现。
- http://projects.spring.io/spring-amqp/



## 6.3.使用

### 6.3.1.消费者

```
package com.zpc.rabbitmq.spring;

/**
 * 消费者
 *
 * @author Evan
 */
public class Foo {

    //具体执行业务的方法
    public void listen(String foo) {
        System.out.println("\n消费者： " + foo + "\n");
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14

### 6.3.2.生产者

```
package com.zpc.rabbitmq.spring;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringMain {
    public static void main(final String... args) throws Exception {
        AbstractApplicationContext ctx = new ClassPathXmlApplicationContext(
                "classpath:spring/rabbitmq-context.xml");
        //RabbitMQ模板
        RabbitTemplate template = ctx.getBean(RabbitTemplate.class);
        //发送消息
        template.convertAndSend("Hello, 鸟鹏!");
        Thread.sleep(1000);// 休眠1秒
        ctx.destroy(); //容器销毁
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18

### 6.3.3.配置文件

### 1、定义连接工厂

```
<!-- 定义RabbitMQ的连接工厂 -->
<rabbit:connection-factory id="connectionFactory"
  host="127.0.0.1" port="5672" username="admin" password="admin"
  virtual-host="testhost" />
```

- 1
- 2
- 3
- 4

### 2、定义模板（可以指定交换机或队列）

```
<rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
exchange="fanoutExchange" />
```

1

## 3、定义队列、交换机、以及完成队列和交换机的绑定

```
<!-- 定义队列，自动声明 -->
<rabbit:queue name="zpcQueue" auto-declare="true"/>

<!-- 定义交换器，把Q绑定到交换机，自动声明 -->
<rabbit:fanout-exchange name="fanoutExchange" auto-declare="true">
  <rabbit:bindings>
    <rabbit:binding queue="zpcQueue"/>
  </rabbit:bindings>
</rabbit:fanout-exchange>
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

## 4、定义监听

```
<rabbit:listener-container connection-factory="connectionFactory">
  <rabbit:listener ref="foo" method="listen" queue-names="zpcQueue" />
</rabbit:listener-container>

<bean id="foo" class="com.zpc.rabbitmq.spring.Foo" />
```

- 1
- 2
- 3
- 4
- 5

## 5、定义管理，用于管理队列、交换机等：

```
<!-- MQ的管理，包括队列、交换器等 -->
<rabbit:admin connection-factory="connectionFactory" />
```

- 1
- 2

## 完整配置文件rabbitmq-context.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rabbit="http://www.springframework.org/schema/rabbit"
  xsi:schemaLocation="http://www.springframework.org/schema/rabbit
```

```
  http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">

  <!-- 定义RabbitMQ的连接工厂 -->
  <rabbit:connection-factory id="connectionFactory"
    host="127.0.0.1" port="5672" username="admin" password="admin"
    virtual-host="testhost" />

  <!-- 定义Rabbit模板，指定连接工厂以及定义exchange -->
  <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
exchange="fanoutExchange" />
  <!-- <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
    exchange="fanoutExchange" routing-key="foo.bar" /> -->

  <!-- MQ的管理，包括队列、交换器等 -->
  <rabbit:admin connection-factory="connectionFactory" />

  <!-- 定义队列，自动声明 -->
  <rabbit:queue name="zpcQueue" auto-declare="true"/>

  <!-- 定义交换器，把Q绑定到交换机，自动声明 -->
  <rabbit:fanout-exchange name="fanoutExchange" auto-declare="true">
    <rabbit:bindings>
      <rabbit:binding queue="zpcQueue"/>
    </rabbit:bindings>
  </rabbit:fanout-exchange>

<!--   <rabbit:topic-exchange name="myExchange">
    <rabbit:bindings>
      <rabbit:binding queue="myQueue" pattern="foo.*" />
    </rabbit:bindings>
  </rabbit:topic-exchange> -->

  <!-- 队列监听 -->
  <rabbit:listener-container connection-factory="connectionFactory">
    <rabbit:listener ref="foo" method="listen" queue-names="zpcQueue" />
  </rabbit:listener-container>

  <bean id="foo" class="com.zpc.rabbitmq.spring.Foo" />

</beans>
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12

- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44

## 6.4.持久化交换机和队列

## Exchanges

All exchanges

Filter: Regex (?)

| Virtual host | Name | Type | Features | Message rate in | Message rate out | +/- |
|---|---|---|---|---|---|---|
| testhost | (AMQP default) | direct | D | 0.00/s | 0.00/s | |
| testhost | amq.direct | direct | D | | | |
| testhost | amq.fanout | fanout | D | | | |
| testhost | amq.headers | headers | D | | | |
| testhost | amq.match | headers | D | | | |
| testhost | amq.rabbitmq.trace | topic | D I | | | |
| testhost | amq.topic | topic | D | | | |
| testhost | fanoutExchange | fanout | D | 0.00/s | 0.00/s | |
| testhost | test_exchange_direct | direct | | 0.00/s | 0.00/s | |
| testhost | test_exchange_fanout | fanout | | 0.00/s | 0.00/s | |
| testhost | test_exchange_topic | topic | | 0.00/s | 0.00/s | |

持久化

非持久化



持久化：将交换机或队列的数据保存到磁盘，服务器宕机或重启之后依然存在。
非持久化：将交换机或队列的数据保存到内存，服务器宕机或重启之后将不存在。

非持久化的性能高于持久化。

如何选择持久化？非持久化？ – 看需求。

欢迎关注公众号「程猿薇茑」

## 7.Spring集成RabbitMQ一个完整案例

创建三个系统A,B,C
A作为生产者，B、C作为消费者(B,C作为web项目启动)
项目下载地址：https://download.csdn.net/download/zpcandzhj/10585077

## 7.1.在A系统中发送消息到交换机

### 7.1.1.导入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.zpc</groupId>
  <artifactId>myrabbitA</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>myrabbit</name>

  <dependencies>
    <dependency>
      <groupId>org.springframework.amqp</groupId>
      <artifactId>spring-rabbit</artifactId>
      <version>1.4.0.RELEASE</version>
    </dependency>

    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>fastjson</artifactId>
      <version>1.2.47</version>
    </dependency>
  </dependencies>
</project>
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

7.1.2.队列和交换机的绑定关系

实现：

1、在配置文件中将队列和交换机完成绑定

2、可以在管理界面中完成绑定

a)绑定关系如果发生变化，需要修改配置文件，并且服务需要重启

b)管理更加灵活

c)更容易对绑定关系的权限管理，流程管理

本例选择第2种方式

7.1.3.配置

rabbitmq-context.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rabbit="http://www.springframework.org/schema/rabbit"
  xsi:schemaLocation="http://www.springframework.org/schema/rabbit
  http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">

  <!-- 定义RabbitMQ的连接工厂 -->
  <rabbit:connection-factory id="connectionFactory"
    host="127.0.0.1" port="5672" username="admin" password="admin"
    virtual-host="testhost" />

  <!-- MQ的管理，包括队列、交换器等 -->
  <rabbit:admin connection-factory="connectionFactory" />

  <!-- 定义交换器，暂时不把Q绑定到交换机，在管理界面去绑定 -->
  <!--<rabbit:topic-exchange name="topicExchange" auto-declare="true" ></rabbit:topic-
exchange>-->
  <rabbit:direct-exchange name="directExchange" auto-declare="true" ></rabbit:direct-exchange>
  <!--<rabbit:fanout-exchange name="fanoutExchange" auto-declare="true" ></rabbit:fanout-
exchange>-->

  <!-- 定义Rabbit模板，指定连接工厂以及定义exchange(exchange要和上面的一致) -->
  <!--<rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
exchange="topicExchange" />-->
  <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
exchange="directExchange" />
  <!--<rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
exchange="fanoutExchange" />-->
</beans>
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

### 7.1.4.消息内容

**方案：**

**1、消息内容使用对象做json序列化发送**

**a)数据大**

**b)有些数据其他人是可能用不到的**

**2、发送特定的业务字段，如id、操作类型**

### 7.1.5.实现

**生产者MsgSender.java：**

```
package com.zpc.myrabbit;

import com.alibaba.fastjson.JSON;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;


/**
 * 消息生产者
 */
public class MsgSender {
    public static void main(String[] args) throws Exception {
        AbstractApplicationContext ctx = new ClassPathXmlApplicationContext(
                "classpath:spring/rabbitmq-context.xml");
        //RabbitMQ模板
        RabbitTemplate template = ctx.getBean(RabbitTemplate.class);

        String date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());//24小时制
        //发送消息
        Map<String, Object> msg = new HashMap<String, Object>();
```

```
        msg.put("type", "1");
        msg.put("date", date);
        template.convertAndSend("type2", JSON.toJSONString(msg));
        Thread.sleep(1000);// 休眠1秒
        ctx.destroy(); //容器销毁
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33

## 7.2.在B系统接收消息

### 7.2.1.导入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.zpc</groupId>
```

```
    <artifactId>myrabbitB</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>myrabbit</name>
    <properties>
        <spring.version>4.1.3.RELEASE</spring.version>
        <fastjson.version>1.2.46</fastjson.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.rabbitmq</groupId>
            <artifactId>amqp-client</artifactId>
            <version>3.4.1</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.amqp</groupId>
            <artifactId>spring-rabbit</artifactId>
            <version>1.4.0.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.47</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${project.artifactId}</finalName>
        <plugins>
            <!-- web层需要配置Tomcat插件 -->
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <configuration>
                    <path>/testRabbit</path>
                    <uriEncoding>UTF-8</uriEncoding>
                    <port>8081</port>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

- 1
- 2
- 3
- 4
- 5

- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50
- 51
- 52
- 53
- 54
- 55

### 7.2.2.配置

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rabbit="http://www.springframework.org/schema/rabbit"
  xsi:schemaLocation="http://www.springframework.org/schema/rabbit
  http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">

  <!-- 定义RabbitMQ的连接工厂 -->
  <rabbit:connection-factory id="connectionFactory"
    host="127.0.0.1" port="5672" username="admin" password="admin"
    virtual-host="testhost" />

  <!-- MQ的管理，包括队列、交换器等 -->
  <rabbit:admin connection-factory="connectionFactory" />

  <!-- 定义B系统需要监听的队列，自动声明 -->
  <rabbit:queue name="q_topic_testB" auto-declare="true"/>

  <!-- 队列监听 -->
  <rabbit:listener-container connection-factory="connectionFactory">
    <rabbit:listener ref="myMQlistener" method="listen" queue-names="q_topic_testB" />
  </rabbit:listener-container>

  <bean id="myMQlistener" class="com.zpc.myrabbit.listener.Listener" />
</beans>
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

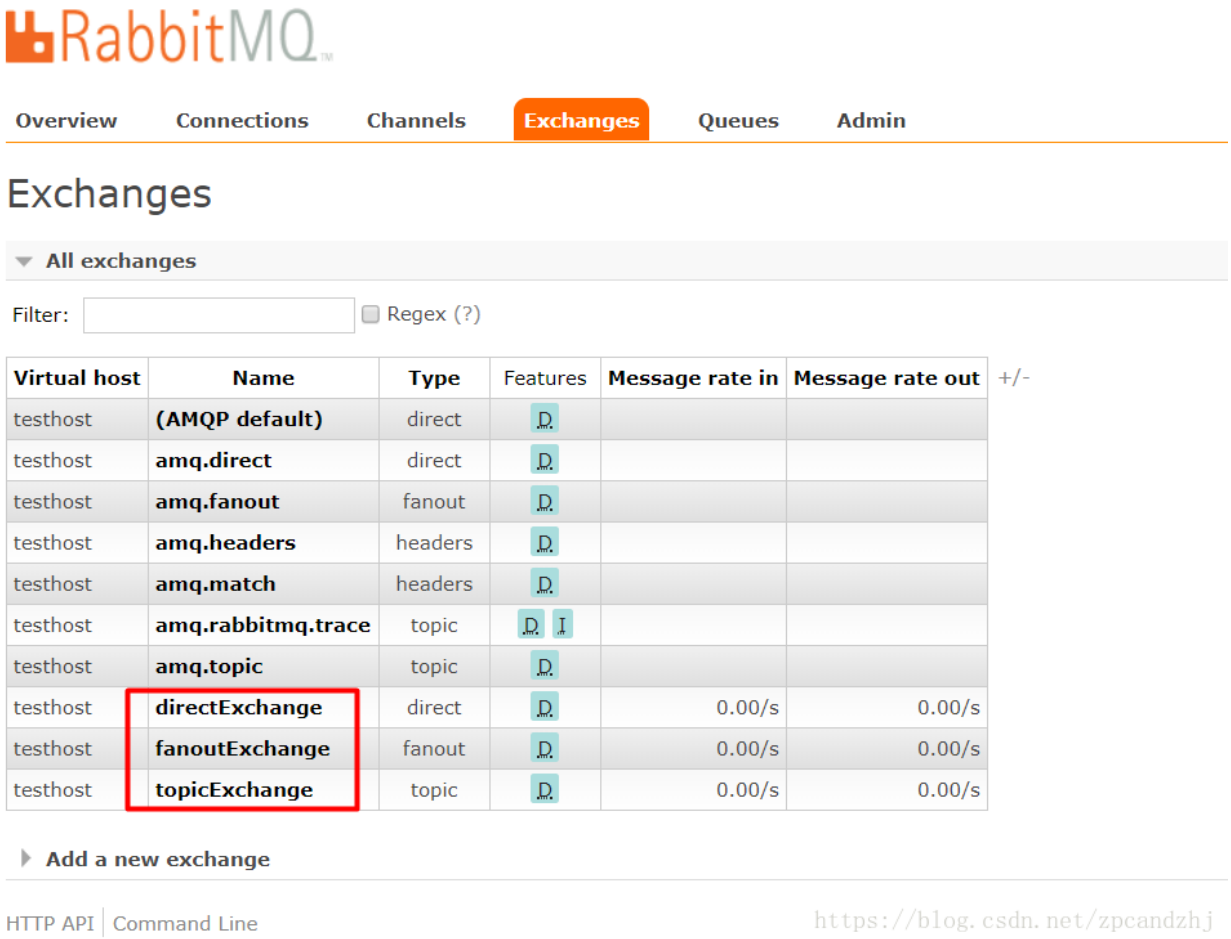
### 7.2.3.具体处理逻辑

```
public class Listener {
    //具体执行业务的方法
    public void listen(String msg) {
        System.out.println("\n消费者B开始处理消息： " + msg + "\n");
    }
}
```

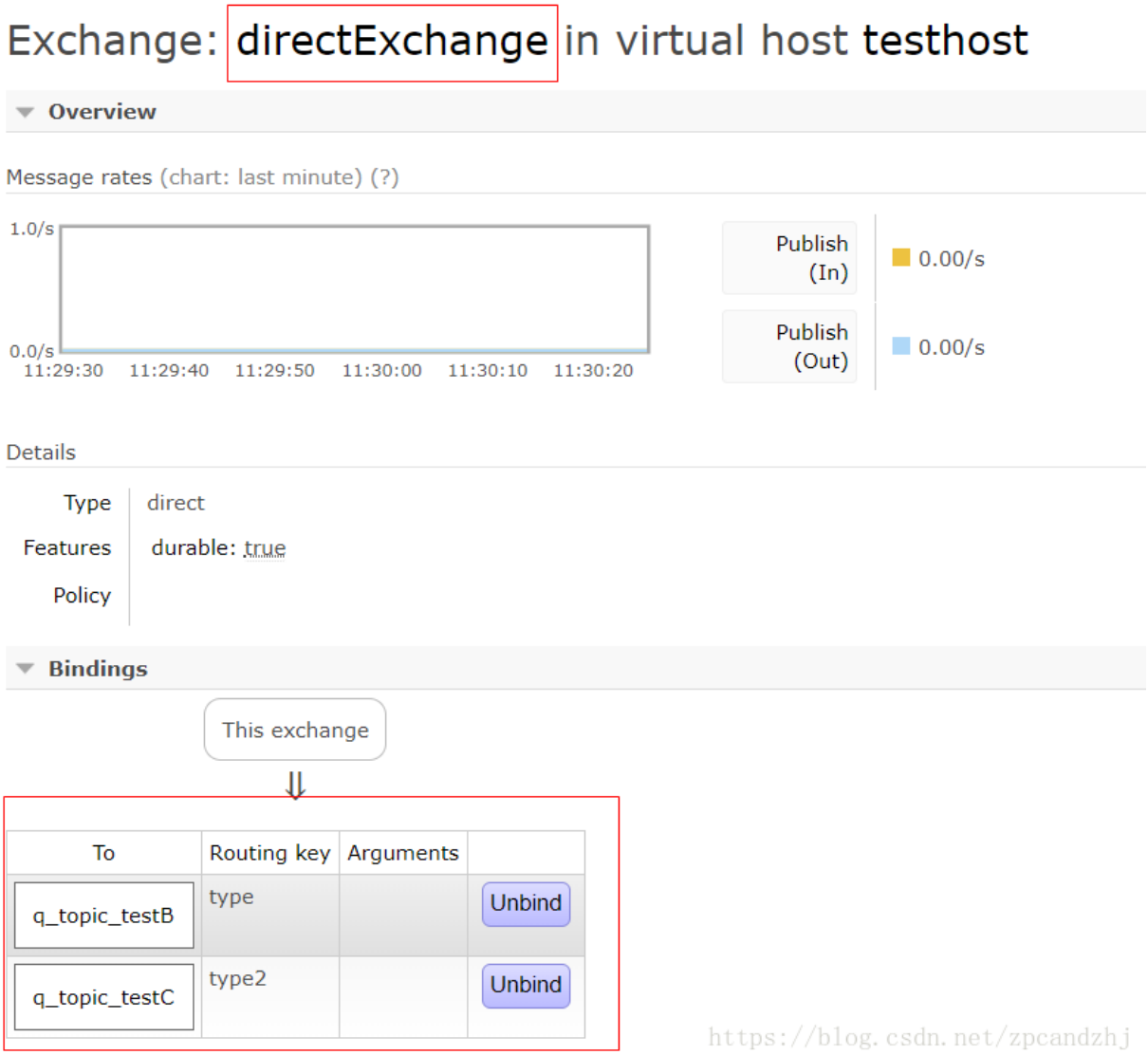
- 1
- 2
- 3
- 4
- 5
- 6

### 7.2.4.在界面管理工具中完成绑定关系

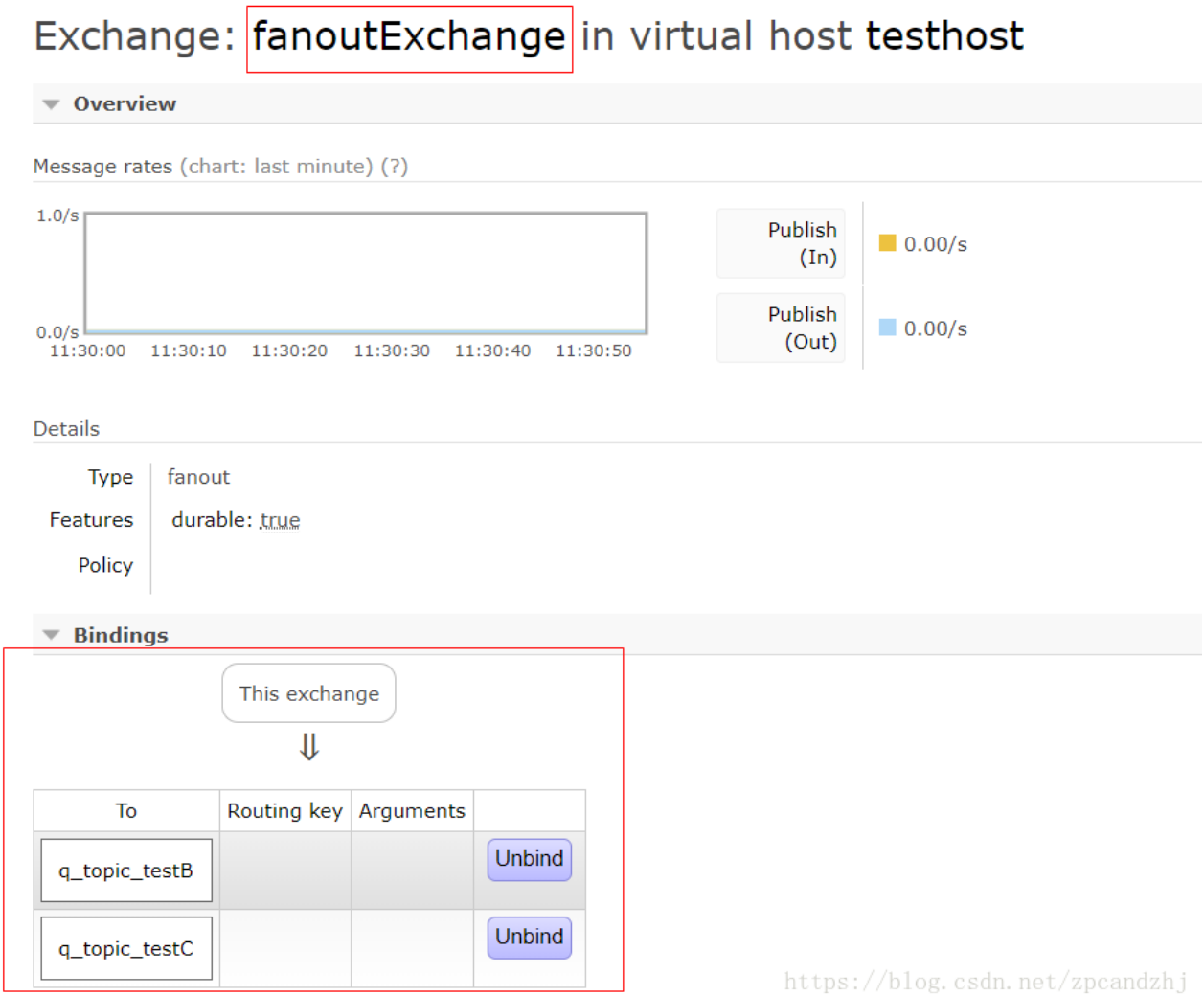
选中定义好的交换机(exchange)



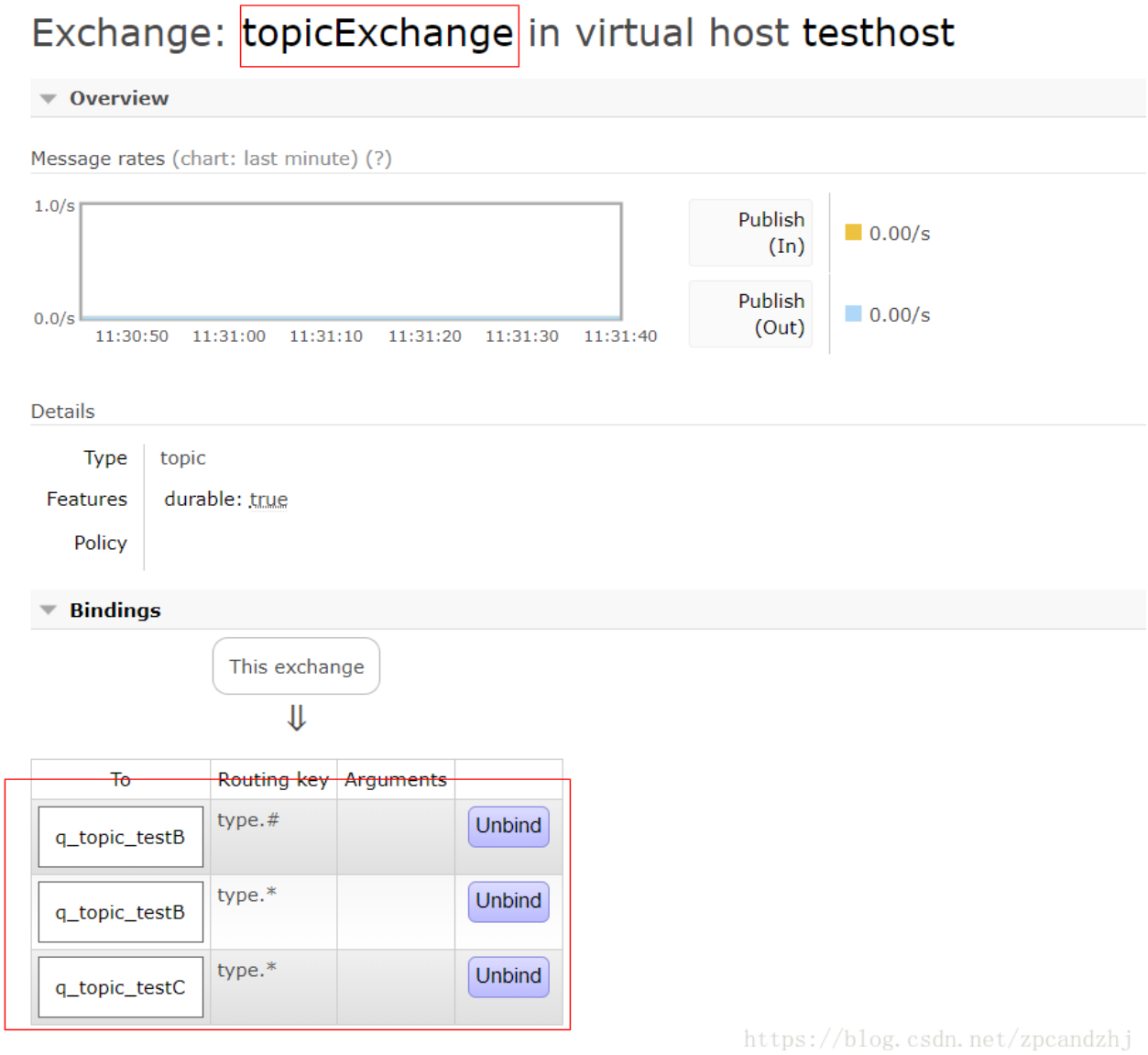

1）direct


Exchange: directExchange in virtual host testhost
Overview
Message rates (chart: last minute) (?)
1.0/s
0.0/s
11:29:30
11:29:40
11:29:50
11:30:00
11:30:10
11:30:20
Publish (In)
0.00/s
Publish (Out)
0.00/s
Details
Type
direct
Features
durable: true
Policy
Bindings
This exchange
To
Routing key
Arguments
q_topic_testB
type
Unbind
q_topic_testC
type2
Unbind
https://blog.csdn.net/zpcandzhj

2）fanout


Exchange: fanoutExchange in virtual host testhost
Overview
Message rates (chart: last minute) (?)
1.0/s
0.0/s
11:30:00
11:30:10
11:30:20
11:30:30
11:30:40
11:30:50
Publish (In)
0.00/s
Publish (Out)
0.00/s
Details
Type
fanout
Features
durable: true
Policy
Bindings
This exchange
To
Routing key
Arguments
q_topic_testB
Unbind
q_topic_testC
Unbind
https://blog.csdn.net/zpcandzhj

3) topic




## 7.3.在C系统中接收消息

（和B系统配置差不多，无非是Q名和Q对应的处理逻辑变了）

### 7.3.1.配置

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rabbit="http://www.springframework.org/schema/rabbit"
  xsi:schemaLocation="http://www.springframework.org/schema/rabbit
  http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">

  <!-- 定义RabbitMQ的连接工厂 -->
  <rabbit:connection-factory id="connectionFactory"
    host="127.0.0.1" port="5672" username="admin" password="admin"
    virtual-host="testhost" />

  <!-- MQ的管理，包括队列、交换器等 -->
  <rabbit:admin connection-factory="connectionFactory" />

  <!-- 定义C系统需要监听的队列，自动声明 -->
  <rabbit:queue name="q_topic_testC" auto-declare="true"/>

  <!-- 队列监听 -->
  <rabbit:listener-container connection-factory="connectionFactory">
    <rabbit:listener ref="myMQlistener" method="listen" queue-names="q_topic_testC" />
  </rabbit:listener-container>

  <bean id="myMQlistener" class="com.zpc.myrabbit.listener.Listener" />
</beans>
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

### 7.3.2.处理业务逻辑

```
public class Listener {

    //具体执行业务的方法
    public void listen(String msg) {
        System.out.println("\n消费者C开始处理消息： " + msg + "\n");
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7

7.3.3.在管理工具中绑定队列和交换机
见7.2.4

7.3.4.测试
分别启动B,C两个web应用，然后运行A的MsgSender主方法发送消息，分别测试fanout、direct、topic三种类型

# 8.Springboot集成RabbitMQ

> springboot集成RabbitMQ非常简单，如果只是简单的使用配置非常少，springboot提供了spring-boot-starter-amqp对消息各种支持。
> 代码下载地址：https://download.csdn.net/download/zpcandzhj/10585077

## 8.1.简单队列

1、配置pom文件，主要是添加spring-boot-starter-amqp的支持

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

- 1
- 2
- 3
- 4

2、配置application.properties文件
配置rabbitmq的安装地址、端口以及账户信息

```
spring.application.name=spirng-boot-rabbitmq
spring.rabbitmq.host=127.0.0.1
spring.rabbitmq.port=5672
spring.rabbitmq.username=admin
spring.rabbitmq.password=admin
```

- 1
- 2
- 3
- 4
- 5

## 3、配置队列

```
package com.zpc.rabbitmq;

import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitConfig {
    @Bean
    public Queue queue() {
        return new Queue("q_hello");
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13

## 4、发送者

```
package com.zpc.rabbitmq;

import org.springframework.amqp.core.AmqpTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.text.SimpleDateFormat;
import java.util.Date;

@Component
public class HelloSender {
    @Autowired
    private AmqpTemplate rabbitTemplate;

    public void send() {
        String date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());//24小时制
        String context = "hello " + date;
        System.out.println("Sender : " + context);
        //简单对列的情况下routingKey即为Q名
        this.rabbitTemplate.convertAndSend("q_hello", context);
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22

## 5、接收者

```
package com.zpc.rabbitmq;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RabbitListener(queues = "q_hello")
public class HelloReceiver {

    @RabbitHandler
    public void process(String hello) {
        System.out.println("Receiver  : " + hello);
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

## 6、测试

```
package com.zpc.rabbitmq;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class RabbitMqHelloTest {

    @Autowired
    private HelloSender helloSender;

    @Test
    public void hello() throws Exception {
        helloSender.send();
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20

## 8.2.多对多使用（Work模式）

**注册两个**Receiver:

```
package com.zpc.rabbitmq;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RabbitListener(queues = "q_hello")
public class HelloReceiver2 {

    @RabbitHandler
    public void process(String hello) {
        System.out.println("Receiver2  : " + hello);
    }

}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16

```
@Test
public void oneToMany() throws Exception {
    for (int i=0;i<100;i++){
        helloSender.send(i);
        Thread.sleep(300);
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7

```
public void send(int i) {
    String date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());//24小时制
    String context = "hello " + i + " " + date;
    System.out.println("Sender : " + context);
    //简单对列的情况下routingKey即为Q名
    this.rabbitTemplate.convertAndSend("q_hello", context);
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7

## 8.3.Topic Exchange（主题模式）

topic 是RabbitMQ中最灵活的一种方式，可以根据routing_key自由的绑定不同的队列

**首先对topic规则配置，这里使用两个队列(消费者)来演示。**
**1)配置队列，绑定交换机**

```
package com.zpc.rabbitmq.topic;

import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.core.TopicExchange;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class TopicRabbitConfig {

    final static String message = "q_topic_message";
    final static String messages = "q_topic_messages";

    @Bean
    public Queue queueMessage() {
        return new Queue(TopicRabbitConfig.message);
    }

    @Bean
    public Queue queueMessages() {
        return new Queue(TopicRabbitConfig.messages);
    }

    /**
     * 声明一个Topic类型的交换机
     * @return
     */
    @Bean
    TopicExchange exchange() {
```

```
        return new TopicExchange("mybootexchange");
    }

    /**
     * 绑定Q到交换机,并且指定routingKey
     * @param queueMessage
     * @param exchange
     * @return
     */
    @Bean
    Binding bindingExchangeMessage(Queue queueMessage, TopicExchange exchange) {
        return BindingBuilder.bind(queueMessage).to(exchange).with("topic.message");
    }

    @Bean
    Binding bindingExchangeMessages(Queue queueMessages, TopicExchange exchange) {
        return BindingBuilder.bind(queueMessages).to(exchange).with("topic.#");
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35

- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50

### 2)创建2个消费者
### q_topic_message 和q_topic_messages

```
package com.zpc.rabbitmq.topic;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RabbitListener(queues = "q_topic_message")
public class Receiver1 {

    @RabbitHandler
    public void process(String hello) {
        System.out.println("Receiver1  : " + hello);
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

```
package com.zpc.rabbitmq.topic;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RabbitListener(queues = "q_topic_messages")
public class Receiver2 {

    @RabbitHandler
    public void process(String hello) {
        System.out.println("Receiver2 : " + hello);
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

### 3)消息发送者（生产者）

```
package com.zpc.rabbitmq.topic;

import org.springframework.amqp.core.AmqpTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MsgSender {

    @Autowired
    private AmqpTemplate rabbitTemplate;

    public void send1() {
        String context = "hi, i am message 1";
        System.out.println("Sender : " + context);
        this.rabbitTemplate.convertAndSend("mybootexchange", "topic.message", context);
    }


    public void send2() {
        String context = "hi, i am messages 2";
        System.out.println("Sender : " + context);
        this.rabbitTemplate.convertAndSend("mybootexchange", "topic.messages", context);
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

send1方法会匹配到topic.#和topic.message，两个Receiver都可以收到消息，发送send2只有topic.#可以匹配所有只有Receiver2监听到消息。
4)测试

```
package com.zpc.rabbitmq.topic;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class RabbitTopicTest {

    @Autowired
    private MsgSender msgSender;

    @Test
    public void send1() throws Exception {
        msgSender.send1();
    }

    @Test
    public void send2() throws Exception {
        msgSender.send2();
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

## 8.4.Fanout Exchange（订阅模式）

Fanout 就是我们熟悉的广播模式或者订阅模式，给Fanout交换机发送消息，绑定了这个交换机的所有队列都收到这个消息。

1)配置队列，绑定交换机

```
package com.zpc.rabbitmq.fanout;

import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.FanoutExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class FanoutRabbitConfig {

    @Bean
    public Queue aMessage() {
        return new Queue("q_fanout_A");
    }

    @Bean
    public Queue bMessage() {
        return new Queue("q_fanout_B");
    }

    @Bean
    public Queue cMessage() {
        return new Queue("q_fanout_C");
    }

    @Bean
    FanoutExchange fanoutExchange() {
        return new FanoutExchange("mybootfanoutExchange");
    }

    @Bean
    Binding bindingExchangeA(Queue aMessage, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(aMessage).to(fanoutExchange);
    }

    @Bean
    Binding bindingExchangeB(Queue bMessage, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(bMessage).to(fanoutExchange);
    }

    @Bean
    Binding bindingExchangeC(Queue cMessage, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(cMessage).to(fanoutExchange);
    }
}
```

- 1
- 2
- 3

- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47

2）创建3个消费者

```
package com.zpc.rabbitmq.fanout;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RabbitListener(queues = "q_fanout_A")
public class ReceiverA {

    @RabbitHandler
    public void process(String hello) {
        System.out.println("AReceiver  : " + hello + "/n");
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

```
package com.zpc.rabbitmq.fanout;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RabbitListener(queues = "q_fanout_B")
public class ReceiverB {

    @RabbitHandler
    public void process(String hello) {
        System.out.println("BReceiver  : " + hello + "/n");
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

```
package com.zpc.rabbitmq.fanout;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RabbitListener(queues = "q_fanout_C")
public class ReceiverC {

    @RabbitHandler
    public void process(String hello) {
        System.out.println("CReceiver  : " + hello + "/n");
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

### 3）生产者

```
package com.zpc.rabbitmq.fanout;

import org.springframework.amqp.core.AmqpTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MsgSenderFanout {

    @Autowired
    private AmqpTemplate rabbitTemplate;

    public void send() {
        String context = "hi, fanout msg ";
        System.out.println("Sender : " + context);
        this.rabbitTemplate.convertAndSend("mybootfanoutExchange","", context);
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18

### 4）测试

```
package com.zpc.rabbitmq.fanout;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class RabbitFanoutTest {

    @Autowired
    private MsgSenderFanout msgSender;

    @Test
    public void send1() throws Exception {
        msgSender.send();
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20

**结果如下，三个消费者都收到消息：**
AReceiver : hi, fanout msg
CReceiver : hi, fanout msg
BReceiver : hi, fanout msg

## 9.总结

- **使用MQ实现商品数据的同步优势：**
  **1、降低系统间耦合度**
  **2、便于管理数据的同步（数据一致性）**

- 推荐阅读
  《RabbitMQ详解》
  《大型网站技术架构：核心原理与案例分析》

推荐springCloud教程：
https://blog.csdn.net/hellozpc/article/details/83692496

推荐Springboot2.0教程：
https://blog.csdn.net/hellozpc/article/details/82531834

文章摘自可爱的小军老师。

**欢迎关注公众号【程猿薇茑】**
**微信扫一扫**