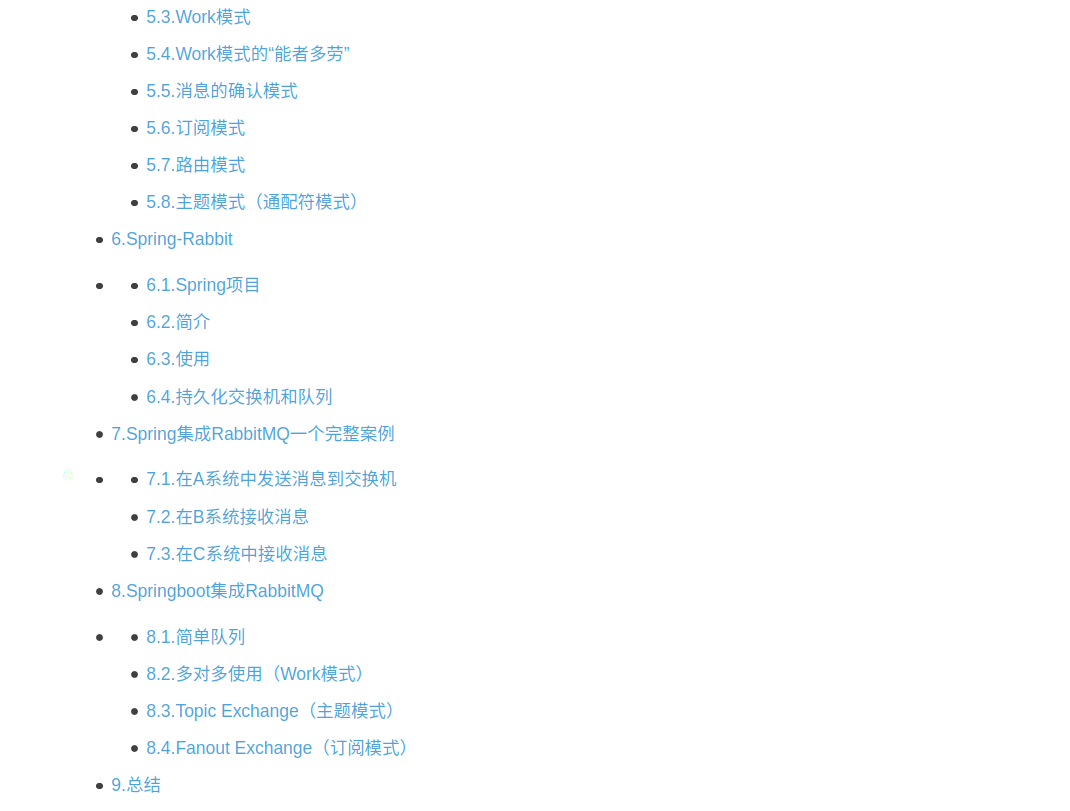
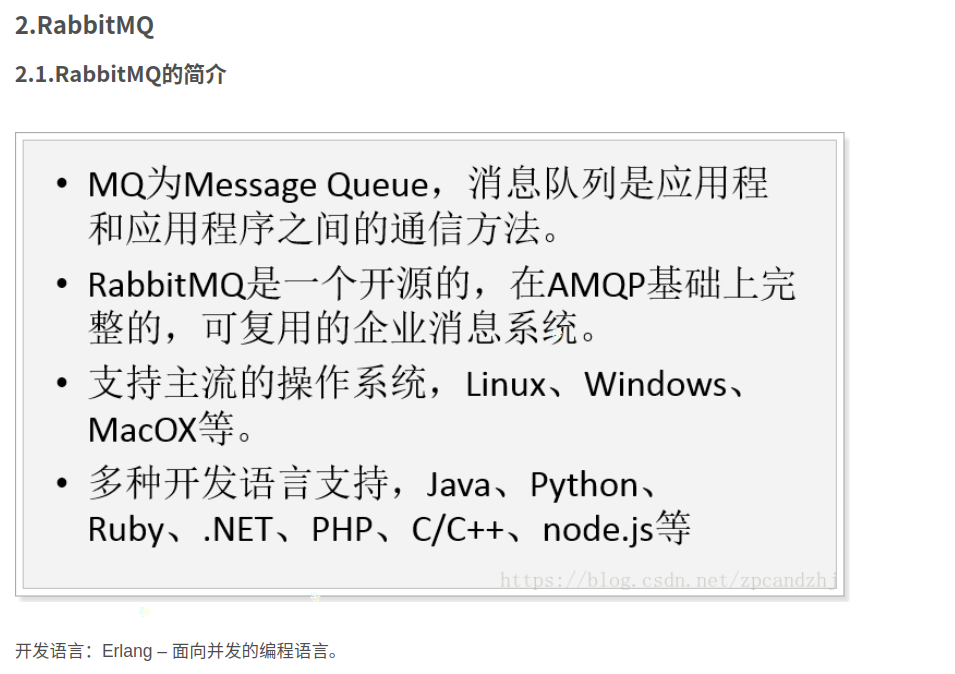
RabbitMQ学习



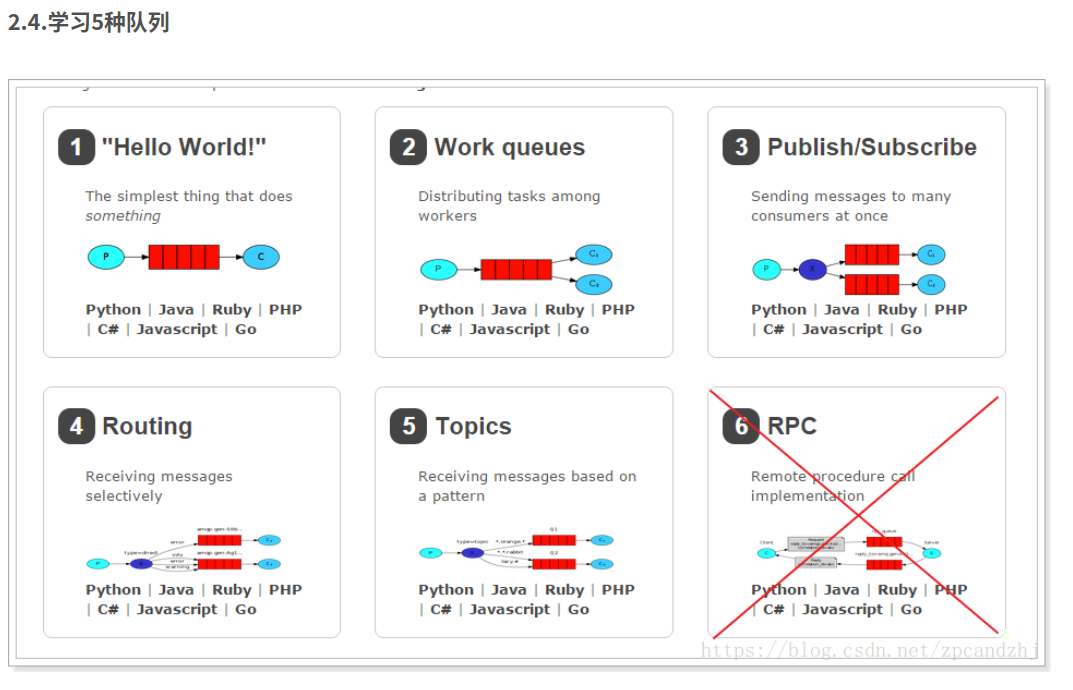






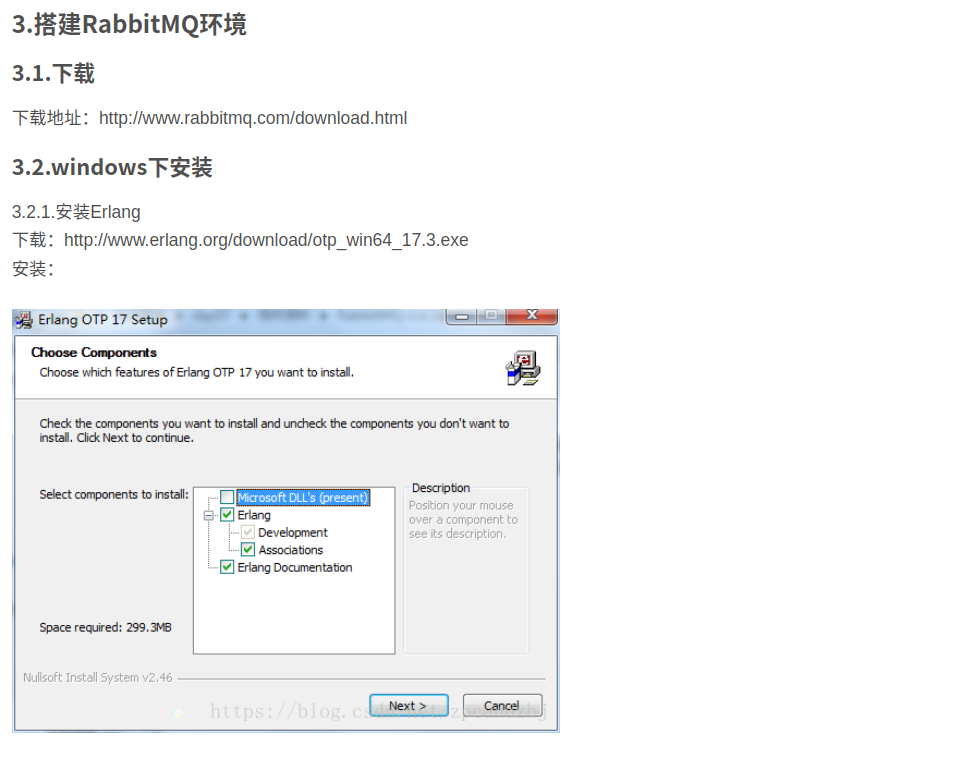


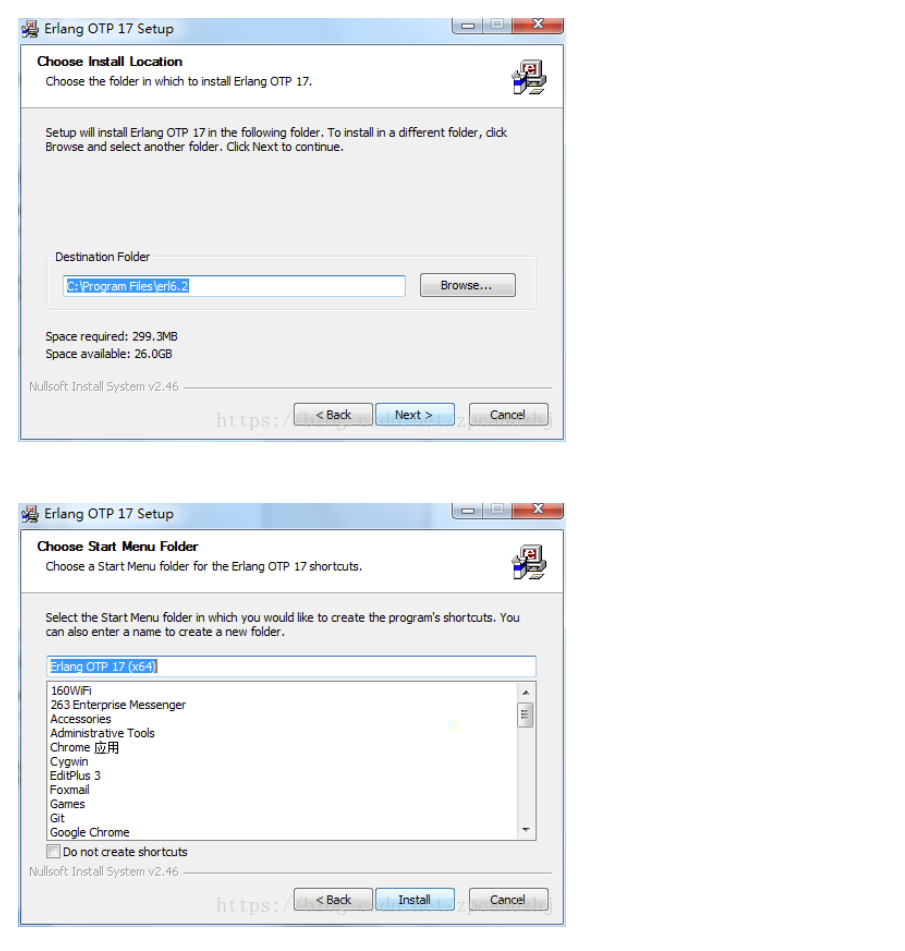


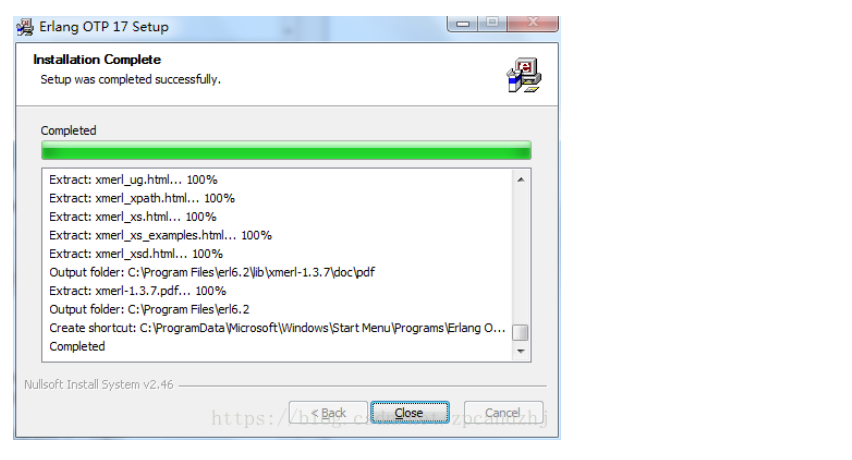


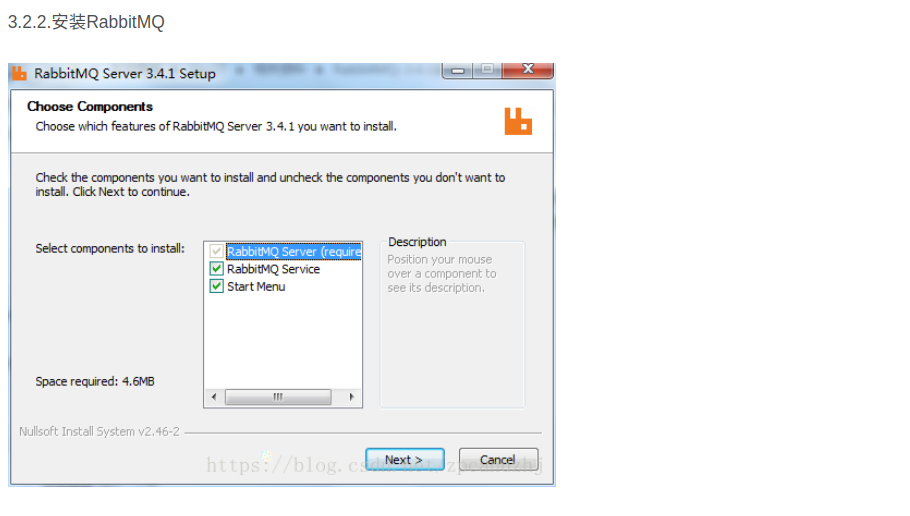
安装RabbtiMQ

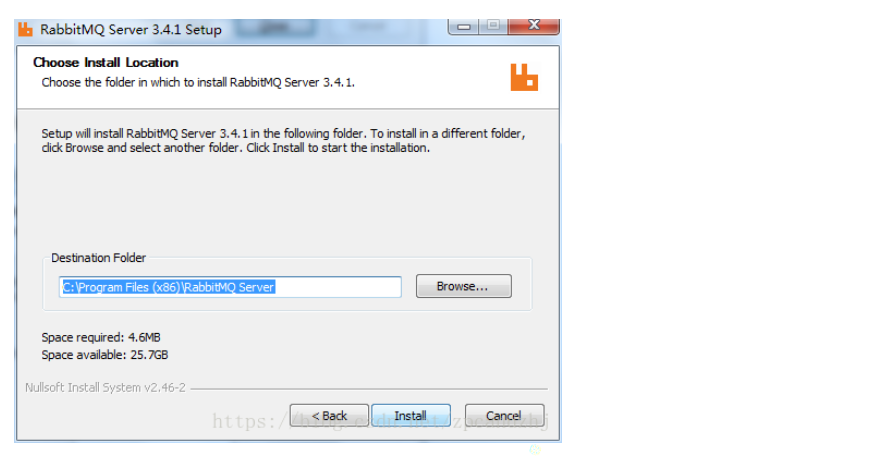




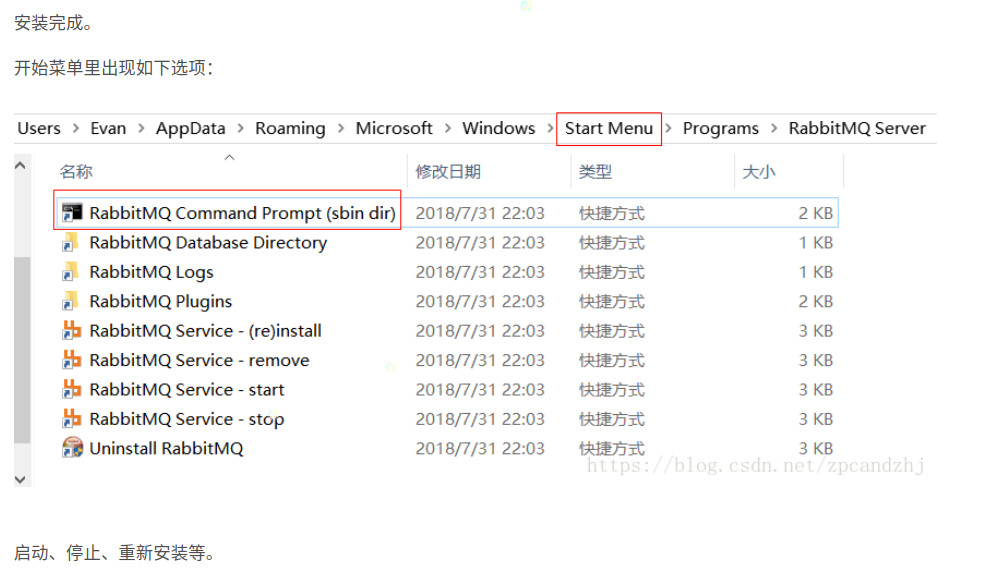


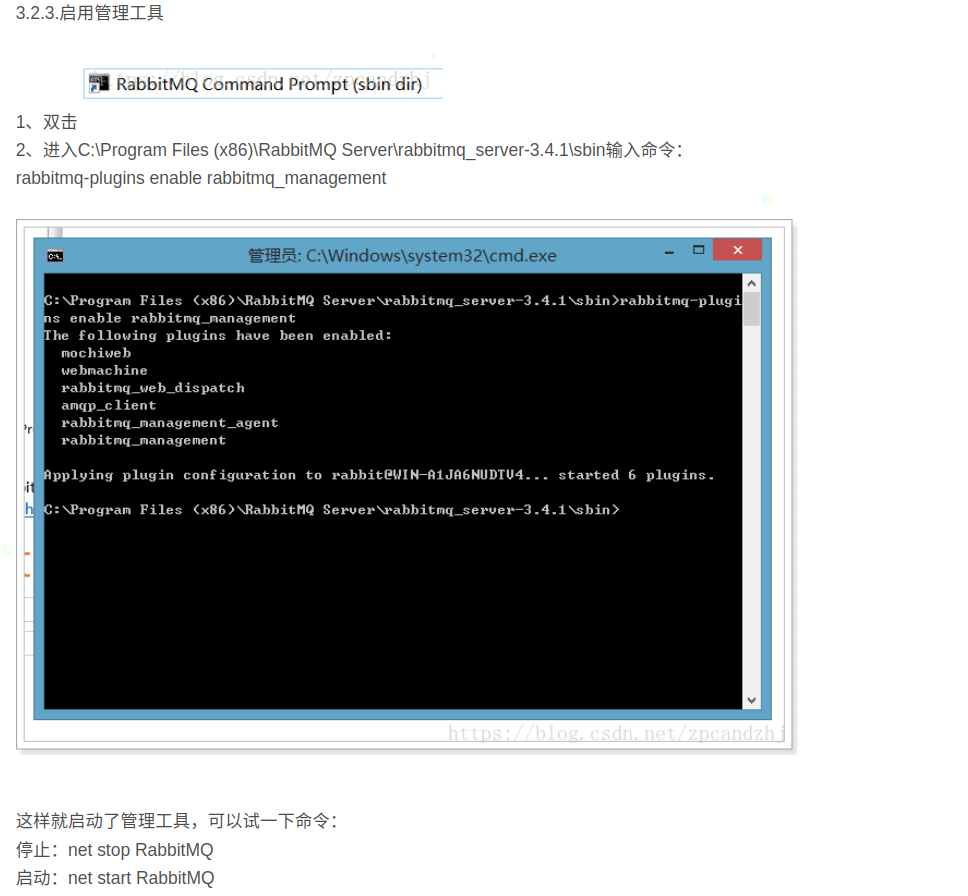














Linux 下安装



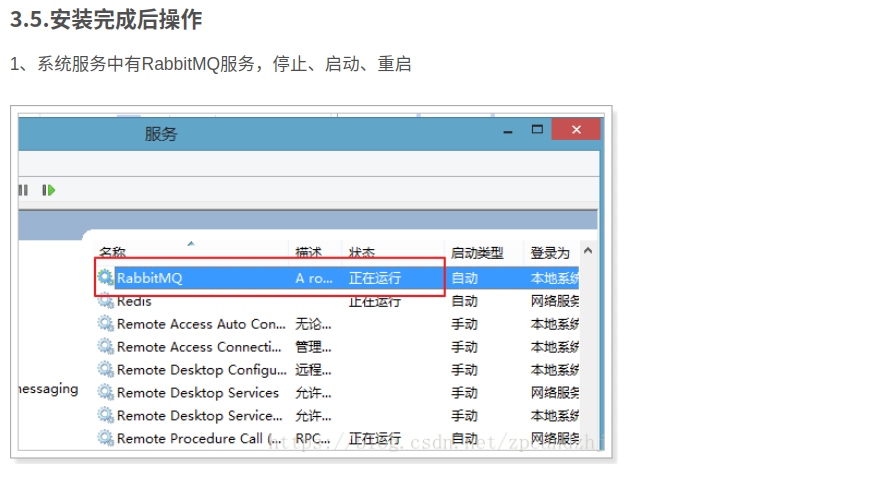


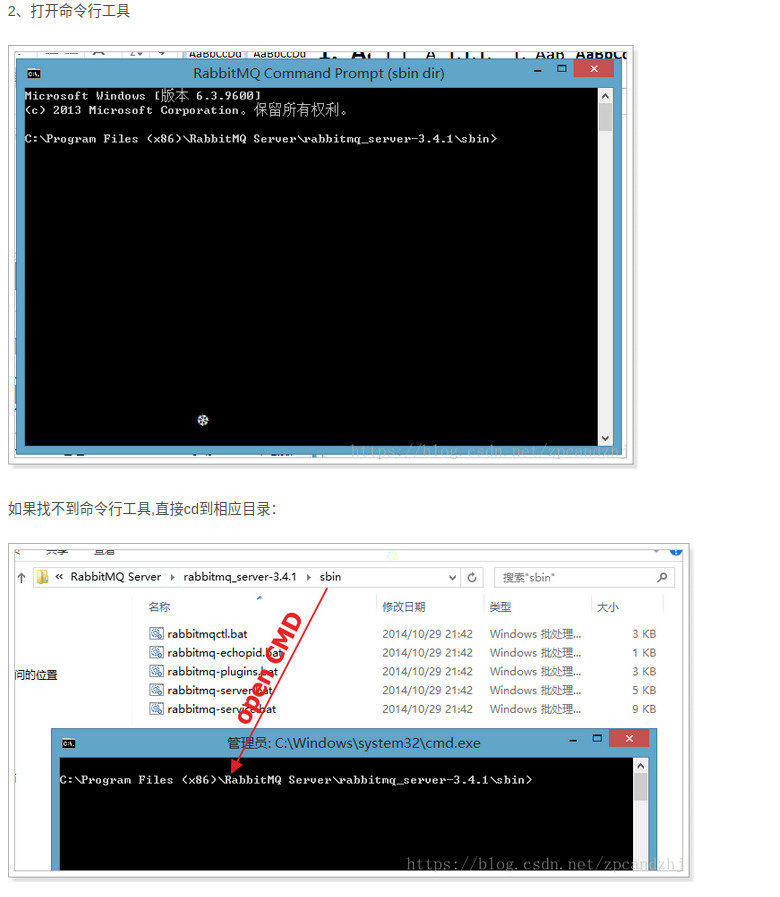
注意事项

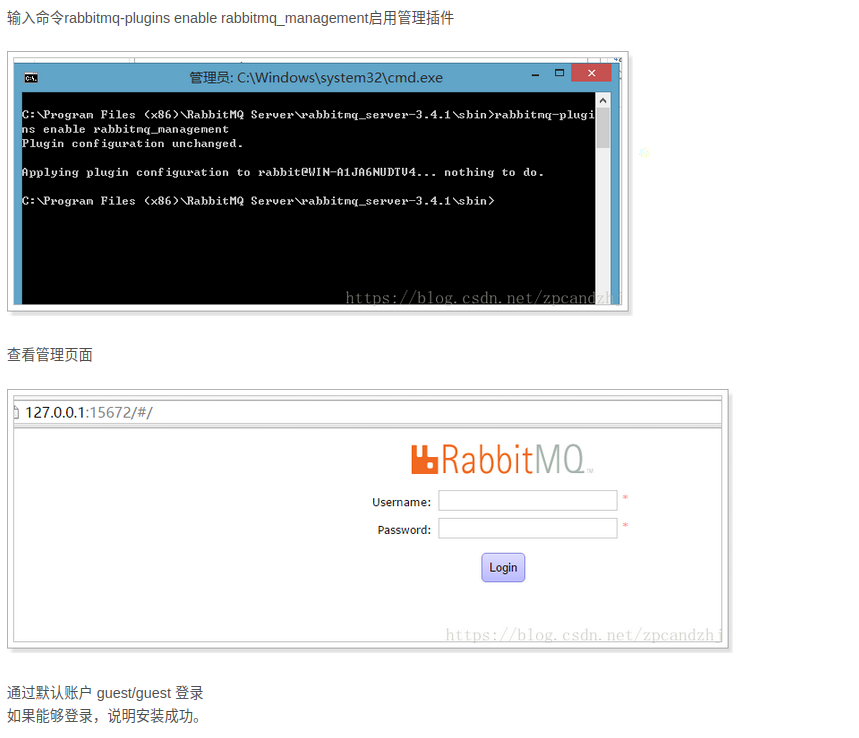




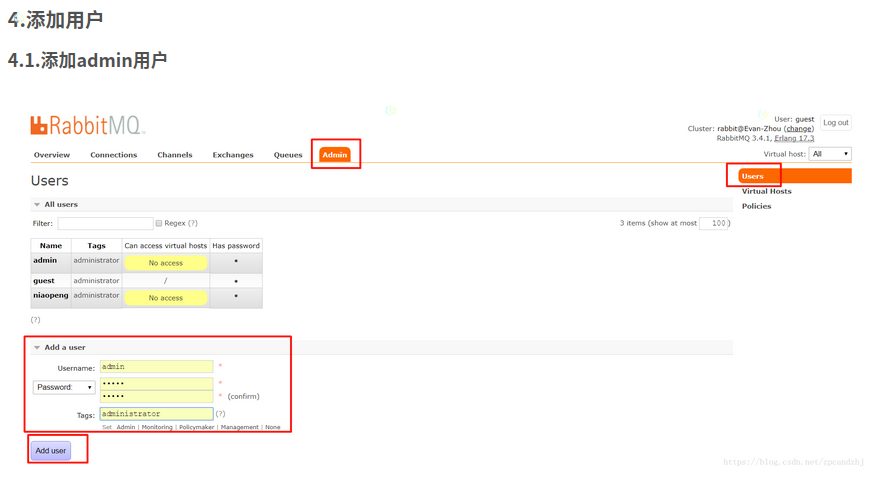
#### 安装完成后操作

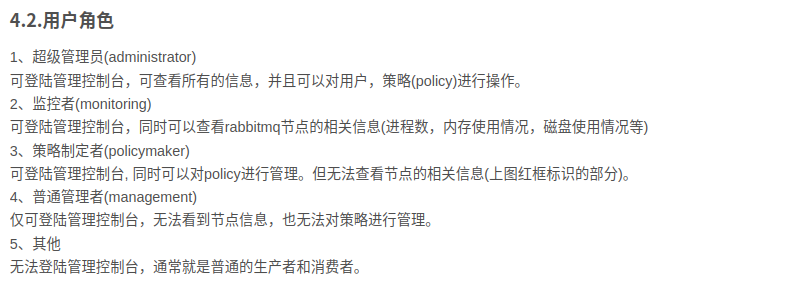




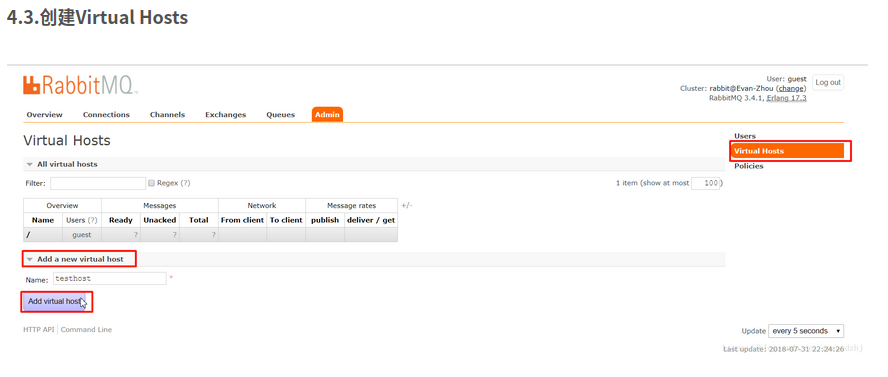


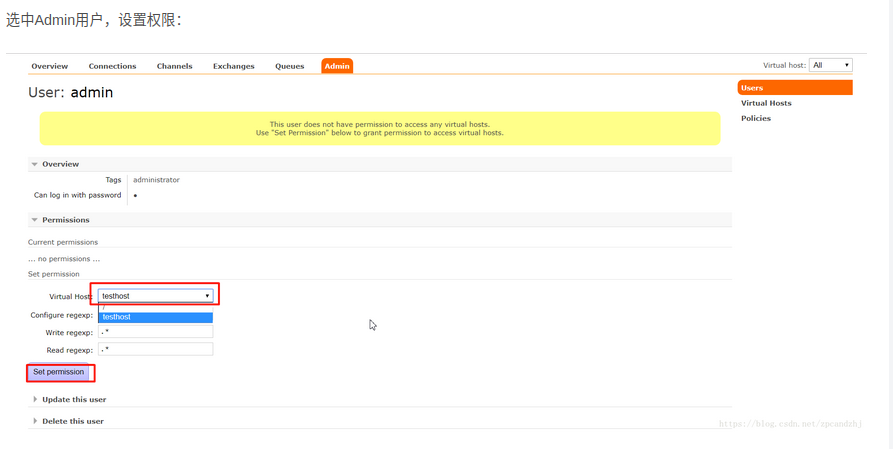
RabbitMQ 页面管理使用

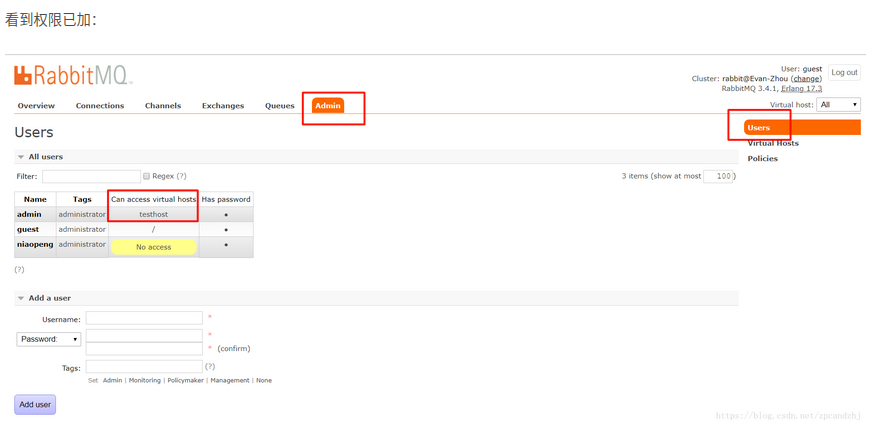




创建虚拟主机

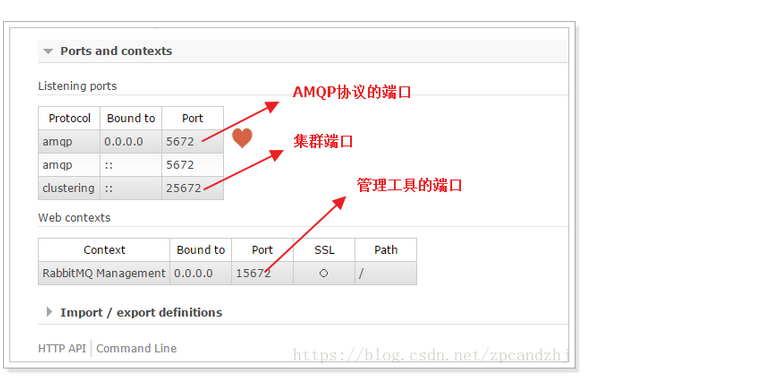




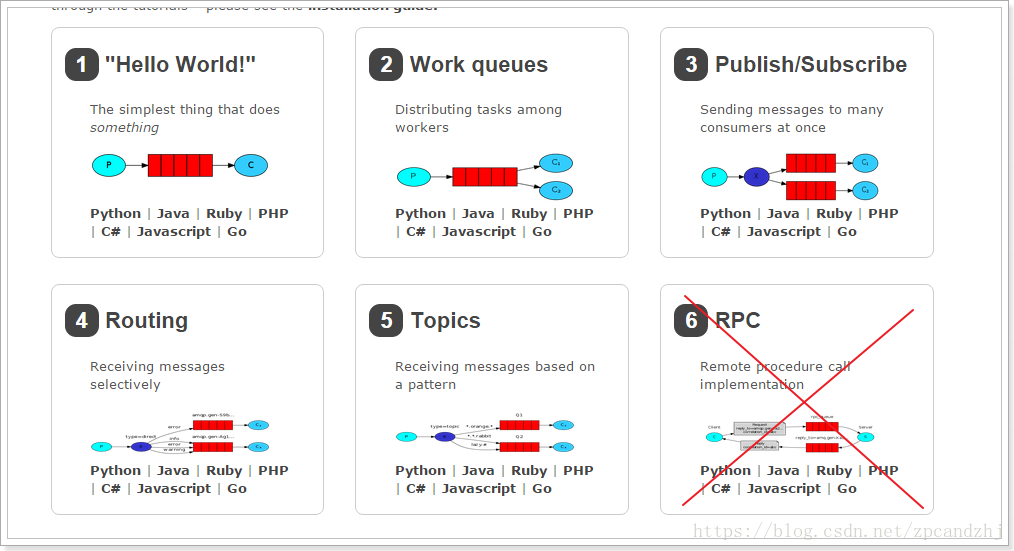


管理界面功能

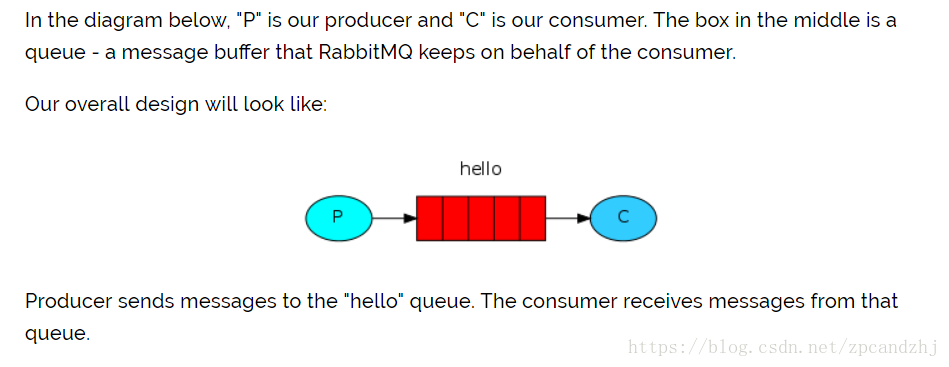




### 5.学习五种队列



#### 5.2.简单队列

5.2.1.图示  


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>

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;

}

}

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 + "'");

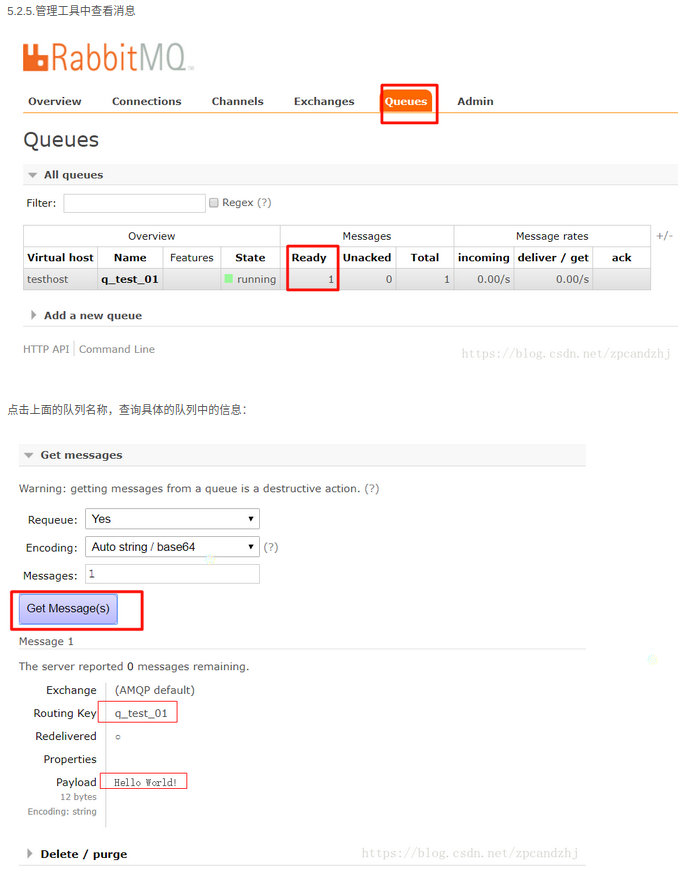
//关闭通道和连接

channel.close();

connection.close();

}

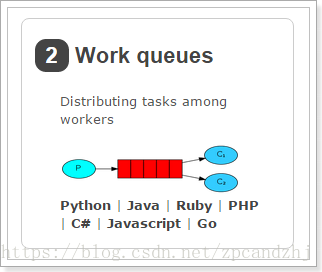
}

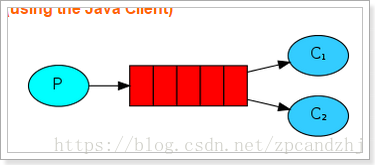


5.2.6.消费者从队列中获取消息

package com.example.demo2.rabbitMQ;  
import com.rabbitmq.client.\*;  
import java.io.Console;  
import java.io.IOException;  
  
public class RabbitRecv {  
*//接收者 rabbitMQ消息的接收者*  
private final static String *QUEUE\_NAME* = "q\_test\_01"; *//消息接收的通道名*  
public static void main(String[] args) throws IOException {  
 *// 获取到连接以及mq通道*  
Connection connection = null;  
 try {  
 connection = ConnectionUtil.*getConnection*();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 *// 从连接中创建通道*  
Channel channel = connection.createChannel();  
  
 *// 声明（创建）队列 --- 队列的声明必须与生产者保持一致*  
channel.queueDeclare(*QUEUE\_NAME*, false, false, false, null);  
  
 channel.basicConsume(*QUEUE\_NAME*, false, "/",  
 new DefaultConsumer(channel) {  
 @Override  
 public void handleDelivery(String consumerTag,Envelope envelope,AMQP.BasicProperties properties,byte[] body)  
 throws IOException  
 {  
 String routingKey = envelope.getRoutingKey();  
 String contentType = properties.getContentType();  
 String message =new String(body);  
 long deliveryTag = envelope.getDeliveryTag();  
 *// (process the message components here ...)*  
  
System.*out*.println("routingKey="+routingKey+" message="+message+" contentType="+contentType +" deliveryTag="+deliveryTag);  
 channel.basicAck(deliveryTag, false);  
  
 }  
 });  
 }  
  
}

#### 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);

}

}

}

1234567891011121314151617181920212223242526272829303132333435363738394041

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);

}

}

}

1234567891011121314151617181920212223242526272829303132333435363738394041

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();

}

}

123456789101112131415161718192021222324252627282930313233

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);

123

//开启这行 表示使用手动确认模式

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

123

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);

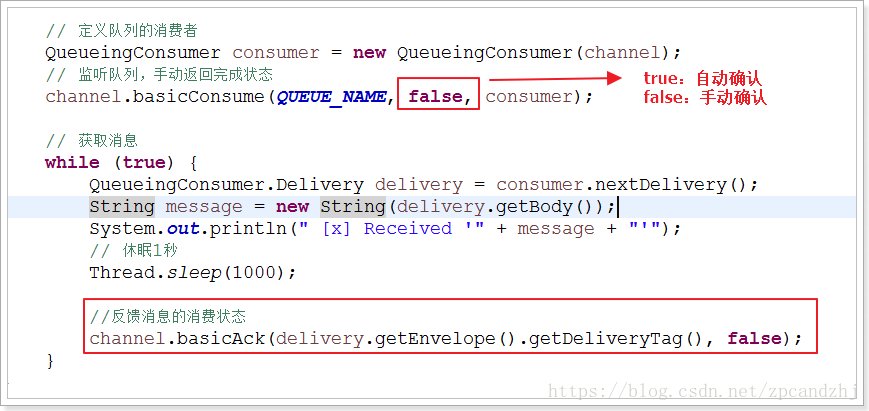
123

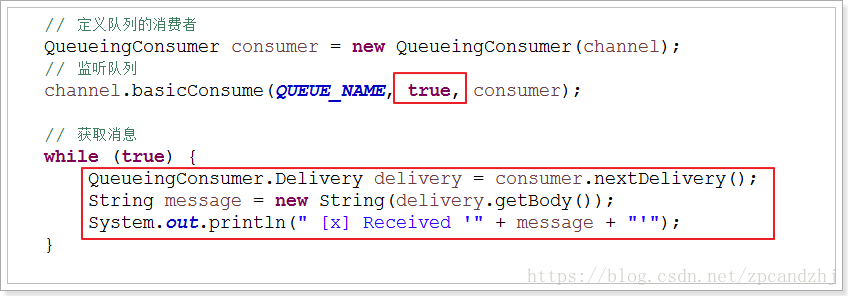
测试：  
消费者1比消费者2获取的消息更多。

#### 5.5.消息的确认模式

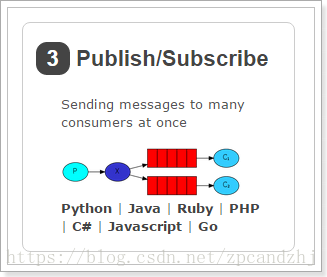
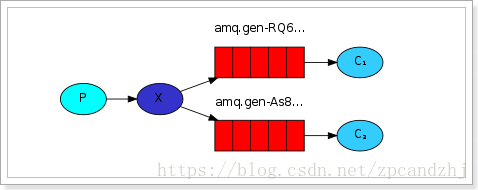
消费者从队列中获取消息，服务端如何知道消息已经被消费呢？

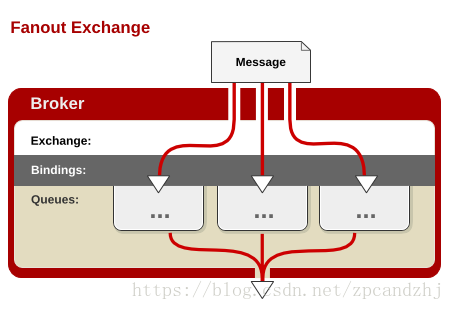
模式1：自动确认  
只要消息从队列中获取，无论消费者获取到消息后是否成功消息，都认为是消息已经成功消费。  
模式2：手动确认  
消费者从队列中获取消息后，服务器会将该消息标记为不可用状态，等待消费者的反馈，如果消费者一直没有反馈，那么该消息将一直处于不可用状态。

手动模式：  


自动模式：  


#### 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();

}

}

1234567891011121314151617181920212223242526272829

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);

}

}

}

12345678910111213141516171819202122232425262728293031323334353637383940414243444546

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);

}

}

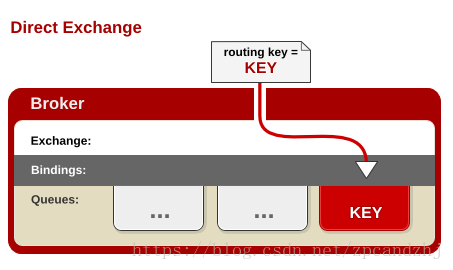
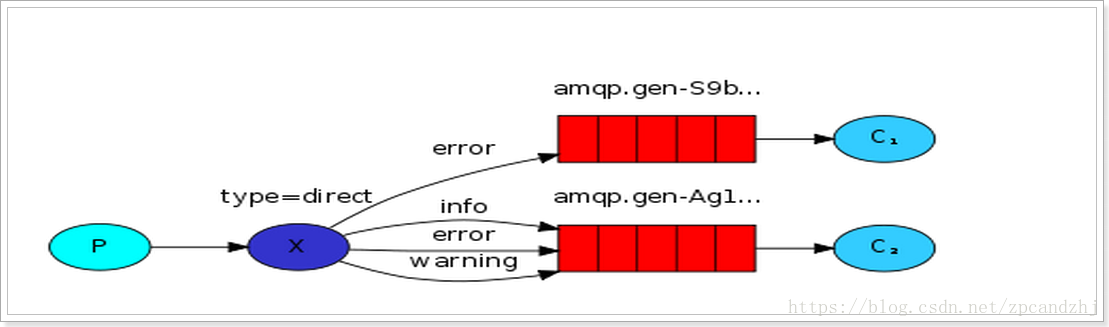
}

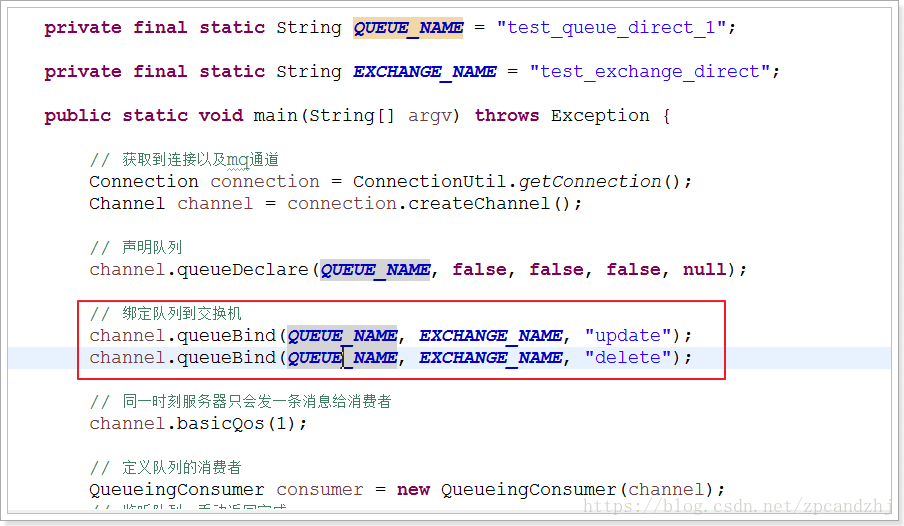
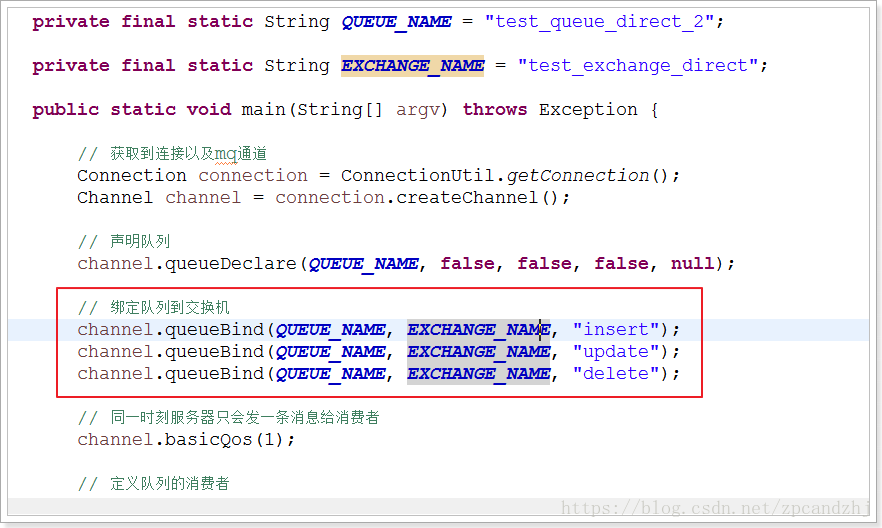
12345678910111213141516171819202122232425262728293031323334353637383940414243444546

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。

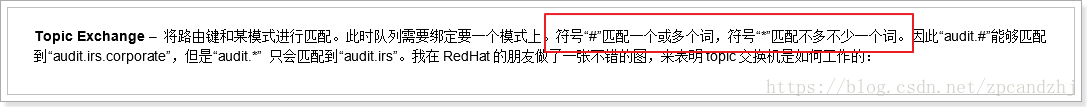
在管理工具中查看队列和交换机的绑定关系：

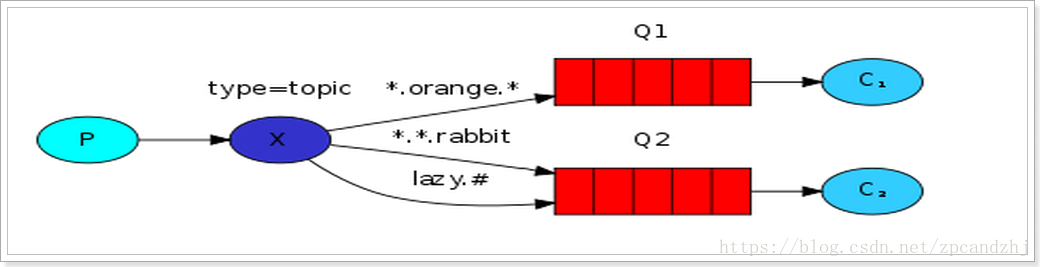
#### 5.7.路由模式

  
5.7.1.图示  


5.7.2.生产者  
  
5.7.3.消费者1(假设是前台系统)  
  
5.7.4.消费2（假设是搜索系统）  


#### 5.8.主题模式（通配符模式）



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();

}

}

1234567891011121314151617181920212223242526272829

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);

}

}

}

12345678910111213141516171819202122232425262728293031323334353637383940414243444546

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);

}

}

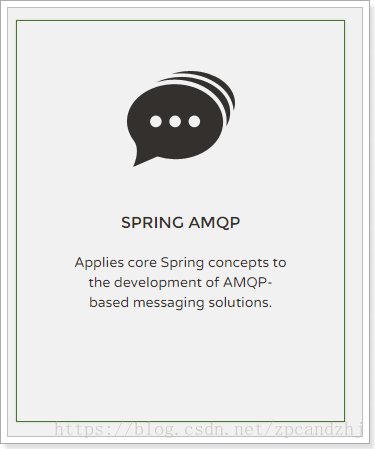
}

12345678910111213141516171819202122232425262728293031323334353637383940414243444546

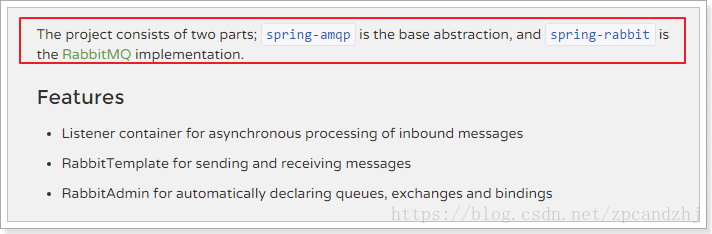
### 6.Spring-Rabbit

#### 6.1.Spring项目

http://spring.io/projects



#### 6.2.简介


#### 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");

}

}

123456789101112131415

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

}

}

12345678910111213141516171819

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" />

12345

2、定义模板（可以指定交换机或队列）

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

12

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>

12345678910

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" />

123456

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" />

123

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" />

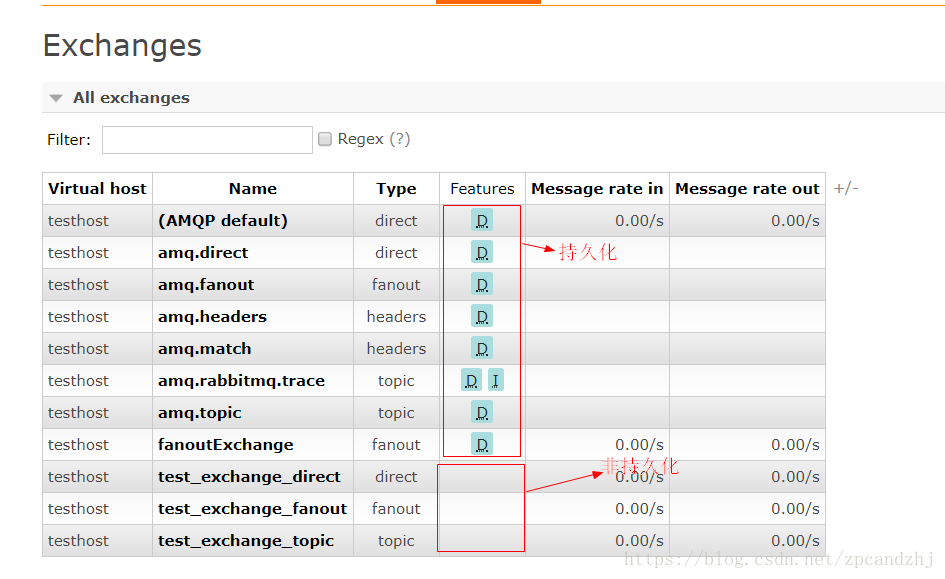
</rabbit:listener-container>

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

</beans>

123456789101112131415161718192021222324252627282930313233343536373839404142434445

#### 6.4.持久化交换机和队列



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。

### 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>

1234567891011121314151617181920212223242526

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>

1234567891011121314151617181920212223242526

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

}

}

12345678910111213141516171819202122232425262728293031323334

#### 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>

1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556

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>

1234567891011121314151617181920212223242526

7.2.3.具体处理逻辑

public class Listener {

//具体执行业务的方法

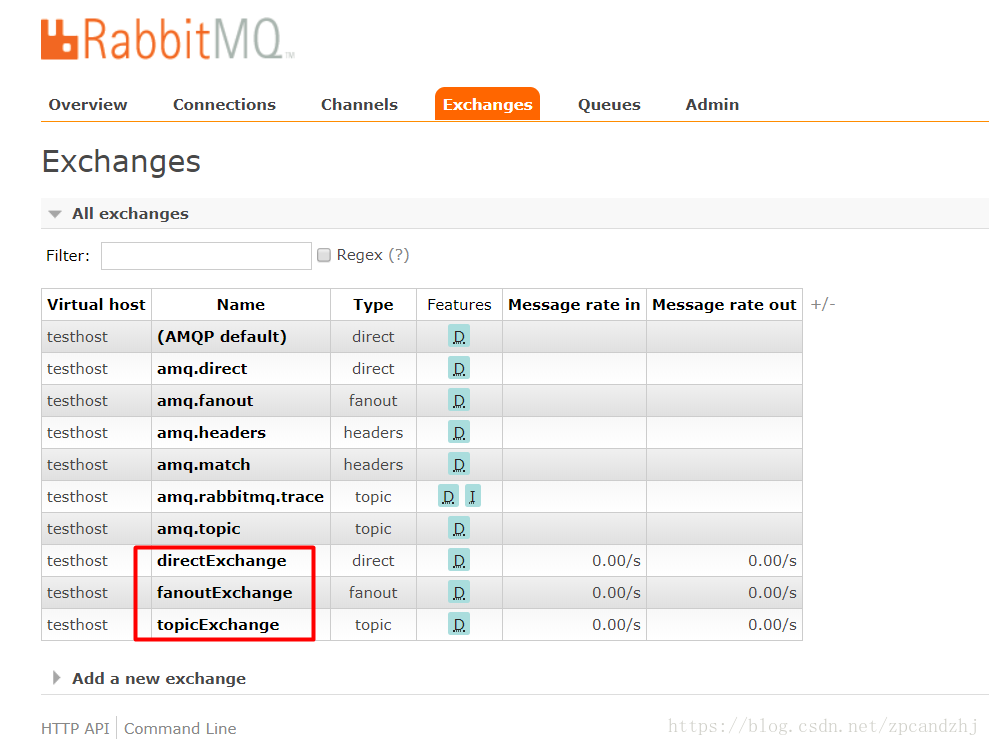
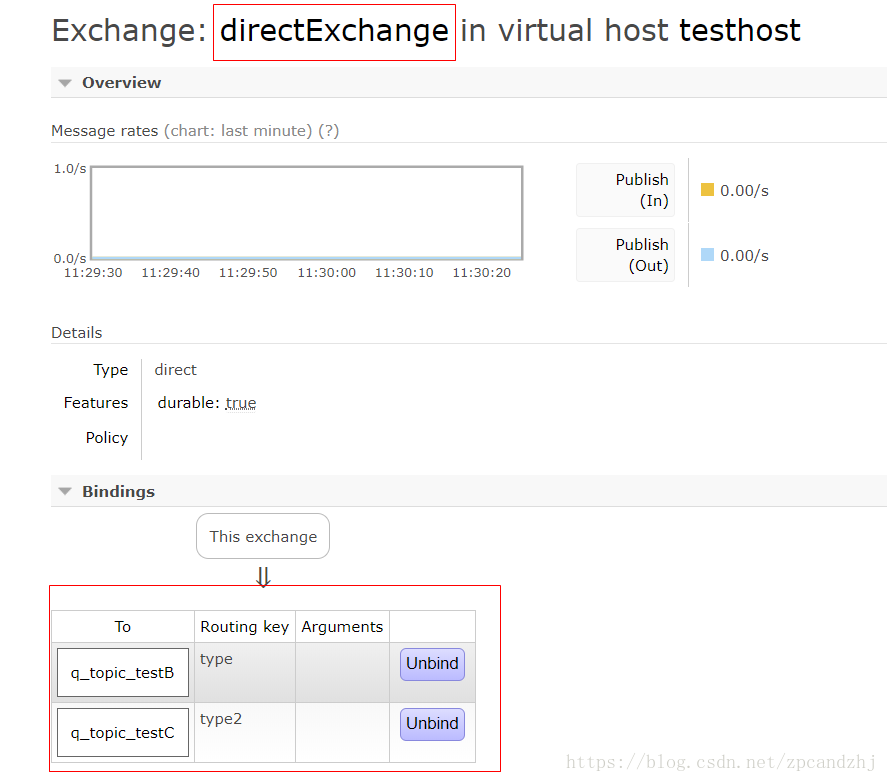
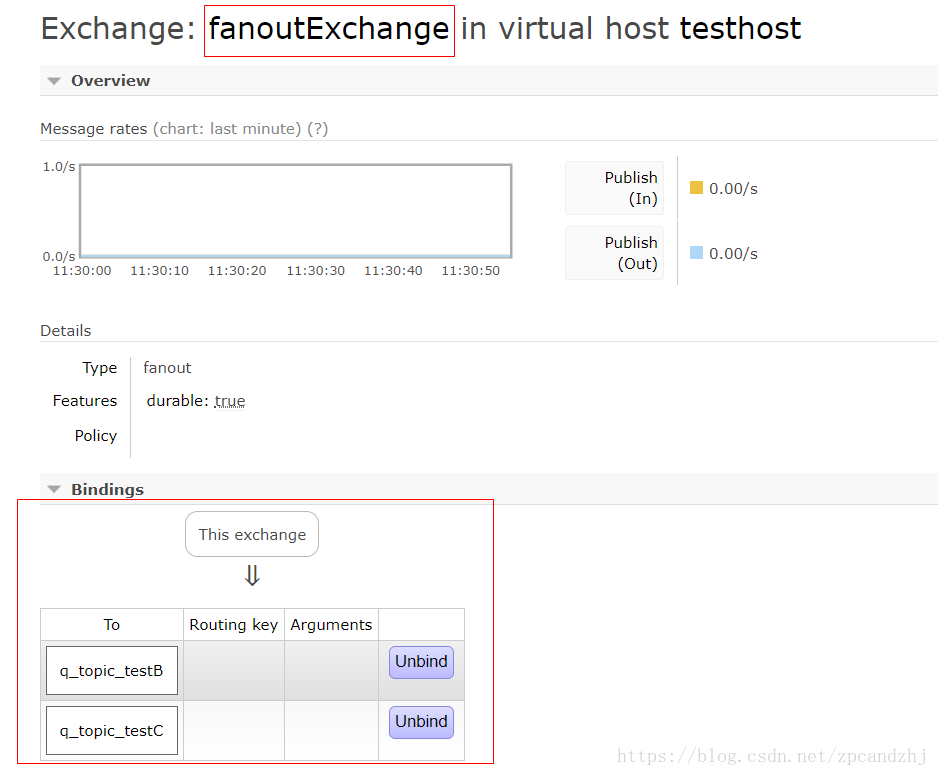
public void listen(String msg) {

System.out.println("\n消费者B开始处理消息： " + msg + "\n");

}

}

1234567

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  
  
2）fanout  
  
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>

1234567891011121314151617181920212223242526

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");

}

}

12345678

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>

12345

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

123456

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");

}

}

1234567891011121314

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);

}

}

1234567891011121314151617181920212223

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);

}

}

12345678910111213141516

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();

}

}

123456789101112131415161718192021

#### 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);

}

}

1234567891011121314151617

@Test

public void oneToMany() throws Exception {

for (int i=0;i<100;i++){

helloSender.send(i);

Thread.sleep(300);

}

}

12345678

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);

}

12345678

#### 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.#");

}

}

123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051

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);

}

}

12345678910111213141516

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);

}

}

12345678910111213141516

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);

}

}

1234567891011121314151617181920212223242526

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();

}

}

1234567891011121314151617181920212223242526

#### 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);

}

}

123456789101112131415161718192021222324252627282930313233343536373839404142434445464748

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");

}

}

12345678910111213141516

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");

}

}

12345678910111213141516

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");

}

}

12345678910111213141516

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);

}

}

12345678910111213141516171819

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();

}

}

123456789101112131415161718192021

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）