**RabbitMQ 消息队列六种模式**

[[](https://www.kuangstudy.com/user/f31d39f29a684914a0fb26e438f902e1) 全栈自学社区](https://www.kuangstudy.com/user/f31d39f29a684914a0fb26e438f902e1)分类：[学习笔记](https://www.kuangstudy.com/bbs?cid=4) 浏览：96 [评论：3](https://www.kuangstudy.com/bbs/1352890836446158849#comments)[**收藏**](javascript:void(0);)最后修改于： 2021/01/23 20:54:37

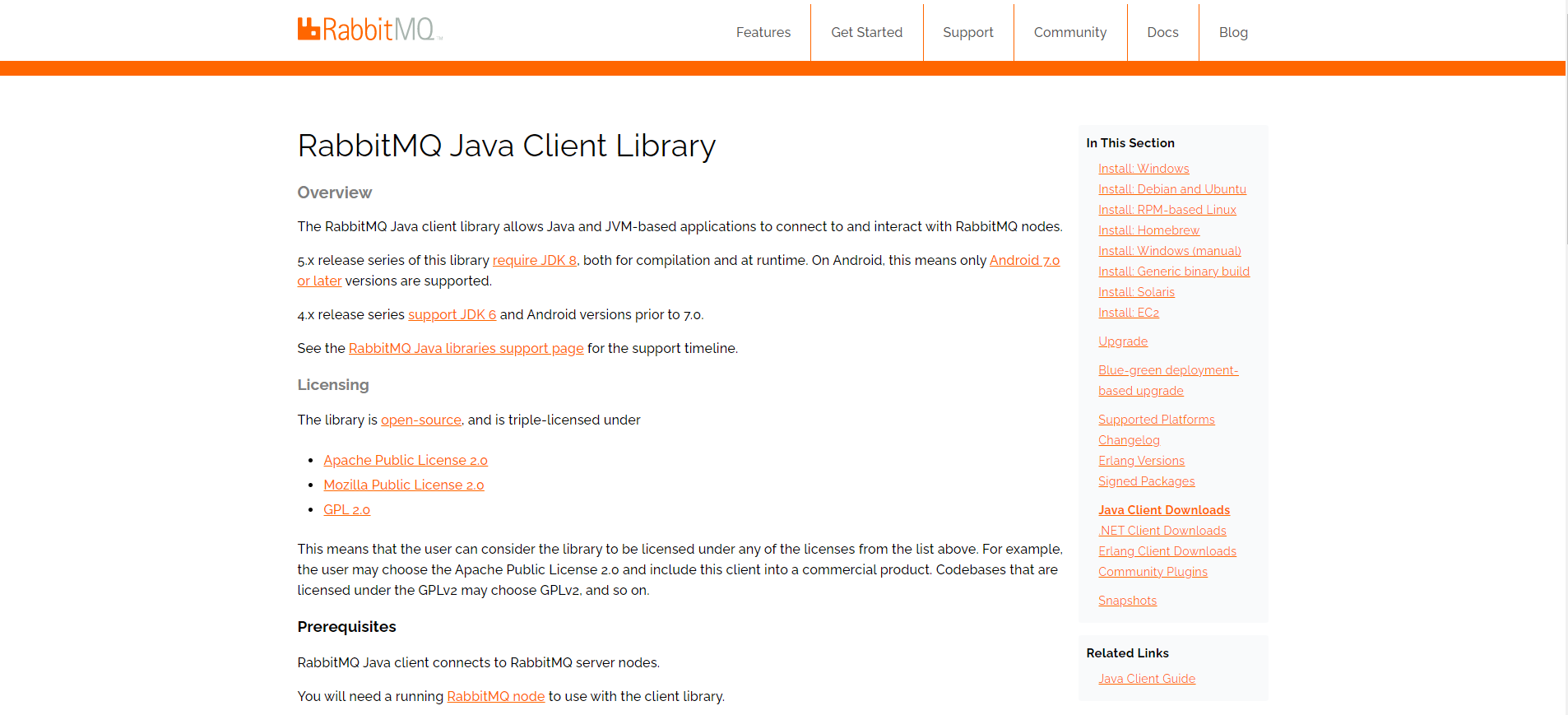
[展开目录+](javascript:void(0);)

**RabbitMQ 消息队列六种模式**

**搭建环境**

**java client**

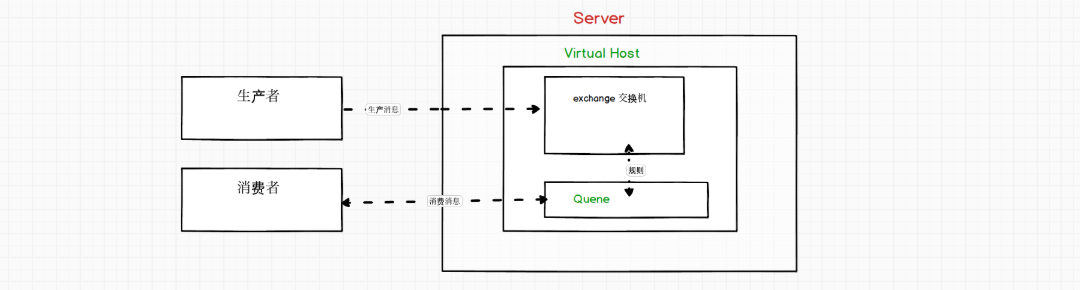
生产者和消费者都属于客户端, rabbitMQ的java客户端如下



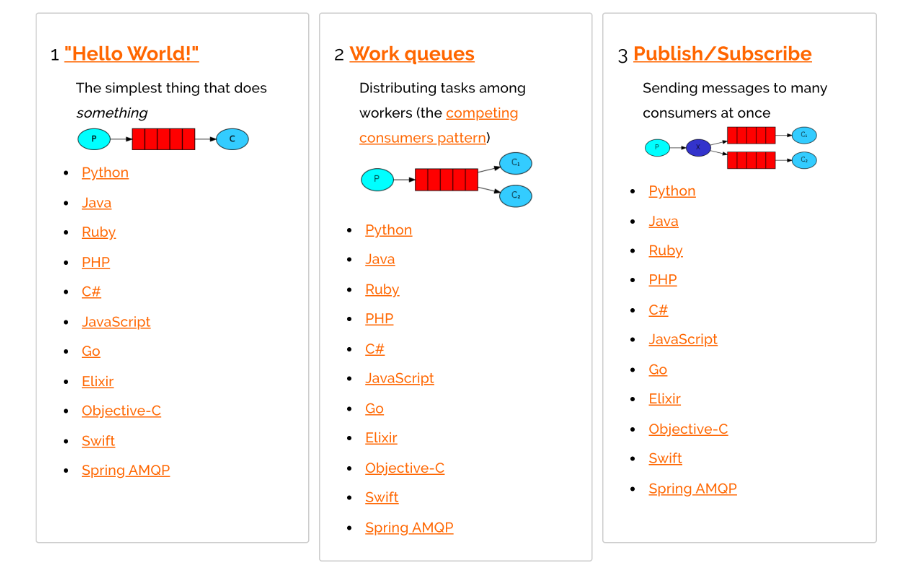
**创建 maven 工程**

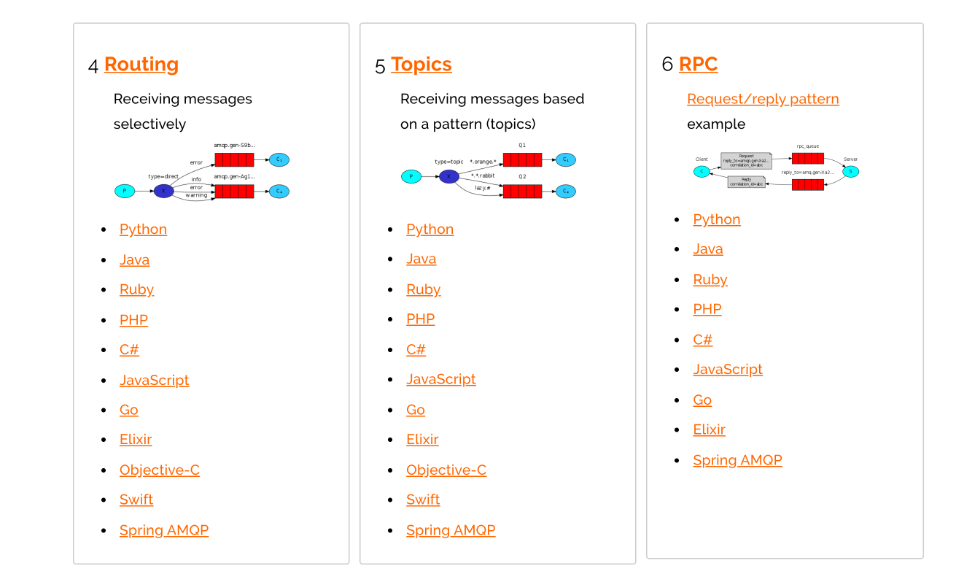
1. <dependency>
2. <groupId>com.rabbitmq</groupId>
3. <artifactId>amqp-client</artifactId>
4. <version>5.10.0</version>
5. </dependency>

**AMQP协议的回顾**



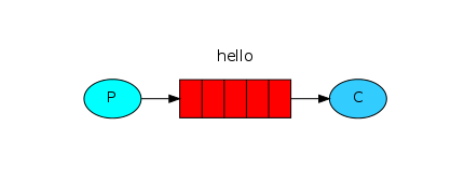
**RabbitMQ支持的消息模型**





**消息中间件模型**

**第一种模型(直连)**



在上图的模型中，有以下概念：

* P：生产者，也就是要发送消息的程序
* C：消费者：消息的接受者，会一直等待消息到来。
* queue：消息队列，图中红色部分。类似一个邮箱，可以缓存消息；生产者向其中投递消息，消费者从其中取出消息。

**开发生产者**

1. /\*\*
2. \* 生产者
3. \* <p>
4. \* 直连模式
5. \*
6. \* @author mxz
7. \*/
8. @Component
9. public class Provider {
10. public static void main(String[] args) throws IOException, TimeoutException {
11. // 获取连接对象
12. Connection connection = RabbitMQUtils.getConnection();
13. // 获取连接中通道
14. Channel channel = connection.createChannel();
15. // 通道绑定消息队列
16. // 参数1 队列的名称, 如果不存在则自动创建
17. // 参数2 用来定义队列是否需要持久化, true 持久化队列(mq关闭时, 会存到磁盘中) false 不持久化(关闭即失)
18. // 参数3 exclusive 是否独占队列 true 独占队列 false 不独占
19. // 参数4 autoDelete 是否在消费后自动删除队列 true 自动删除 false 不删除
20. // 参数5 额外的附加参数
21. channel.queueDeclare("hello", false, false, false, null);
22. // 发布消息
23. // 参数1 交换机名称
24. // 参数2 队列名称
25. // 参数3 传递消息额外设置
26. // 参数4 消息的具体内容
27. channel.basicPublish("", "hello", null, "hello rabbitMQ".getBytes());
28. RabbitMQUtils.closeConnectionAndChannel(channel, connection);
29. }
30. }

**开发消费者**

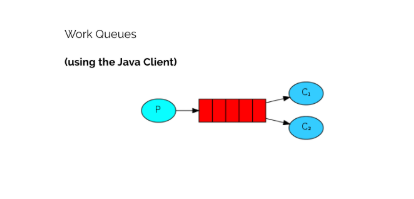
1. /\*\*
2. \* 消费者
3. \*
4. \* @author mxz
5. \*/
6. @Component
7. public class Customer {
8. public static void main(String[] args) throws IOException, TimeoutException {
9. // 获取连接对象
10. Connection connection = RabbitMQUtils.getConnection();
11. // 创建通道
12. Channel channel = connection.createChannel();
13. // 通道绑定对象
14. channel.queueDeclare("hello", false, false, false, null);
15. // 消费消息
16. // 参数1 消息队列的消息, 队列名称
17. // 参数2 开启消息的确认机制
18. // 参数3 消息时的回调接口
19. channel.basicConsume("hello", true, new DefaultConsumer(channel) {
20. // 最后一个参数 消息队列中取出的消息
21. @Override
22. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
23. System.out.println("new String(body)" + new String(body));
24. }
25. });
26. // channel.close();
27. // connection.close();
28. }
29. }

**工具类**

1. /\*\*
2. \* @author mxz
3. \*/
4. public class RabbitMQUtils {
5. private static ConnectionFactory connectionFactory;
6. // 重量级资源 类加载执行一次(即可)
7. static {
8. // 创建连接 mq 的连接工厂
9. connectionFactory = new ConnectionFactory();
10. // 设置 rabbitmq 主机
11. connectionFactory.setHost("127.0.0.1");
12. // 设置端口号
13. connectionFactory.setPort(5672);
14. // 设置连接哪个虚拟主机
15. connectionFactory.setVirtualHost("/codingce");
16. // 设置访问虚拟主机用户名密码
17. connectionFactory.setUsername("codingce");
18. connectionFactory.setPassword("123456");
19. }
20. /\*\*
21. \* 定义提供连接对象的方法
22. \*
23. \* @return
24. \*/
25. public static Connection getConnection() {
26. try {
27. return connectionFactory.newConnection();
28. } catch (Exception e) {
29. e.printStackTrace();
30. }
31. return null;
32. }
33. /\*\*
34. \* 关闭通道和关闭连接工具方法
35. \*
36. \* @param connection
37. \* @param channel
38. \*/
39. public static void closeConnectionAndChannel(Channel channel, Connection connection) {
40. try {
41. // 先关 channel
42. if (channel != null)
43. channel.close();
44. if (connection != null)
45. connection.close();
46. } catch (Exception e) {
47. e.printStackTrace();
48. }
49. }
50. }

**第二种模型(work quene)**

Work queues，也被称为（Task queues），任务模型。当消息处理比较耗时的时候，可能生产消息的速度会远远大于消息的消费速度。长此以往，消息就会堆积越来越多，无法及时处理。此时就可以使用work 模型：**让多个消费者绑定到一个队列，共同消费队列中的消息**。队列中的消息一旦消费，就会消失，因此任务是不会被重复执行的。



角色：

* P：生产者：任务的发布者
* C1：消费者-1，领取任务并且完成任务，假设完成速度较慢
* C2：消费者-2：领取任务并完成任务，假设完成速度快

**开发生产者**

1. /\*\*
2. \* 生产者
3. \* <p>
4. \* 任务模型 work quenue
5. \*
6. \* @author mxz
7. \*/
8. @Component
9. public class Provider {
10. public static void main(String[] args) throws IOException, TimeoutException {
11. Connection connection = RabbitMQUtils.getConnection();
12. Channel channel = connection.createChannel();
13. // 通过通道声明队列
14. channel.queueDeclare("work", true, false, false, null);
15. for (int i = 0; i < 10; i++) {
16. // 生产消息
17. channel.basicPublish("", "work", null, (" " + i + "work quenue").getBytes());
18. }
19. // 关闭资源
20. RabbitMQUtils.closeConnectionAndChannel(channel, connection);
21. }
22. }

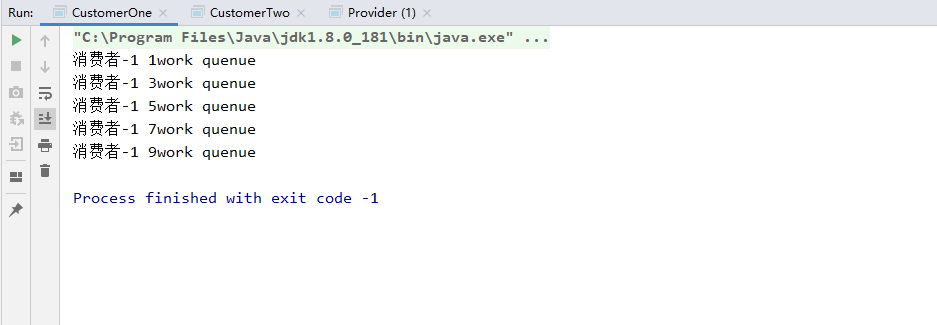
**开发消费者-1**

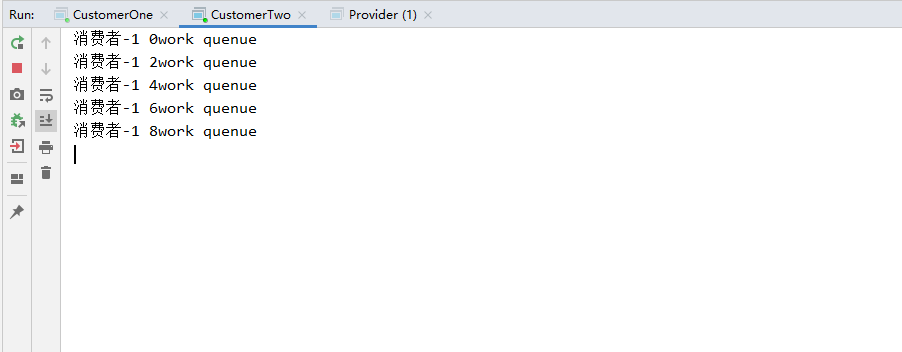
1. /\*\*
2. \* 自动确认消费 autoAck true 12搭配测试
3. \* <p>
4. \* 消费者 1
5. \*
6. \* @author mxz
7. \*/
8. @Component
9. public class CustomerOne {
10. public static void main(String[] args) throws IOException, TimeoutException {
11. // 获取连接对象
12. Connection connection = RabbitMQUtils.getConnection();
13. // 创建通道
14. Channel channel = connection.createChannel();
15. // 通道绑定对象
16. channel.queueDeclare("work", true, false, false, null);
17. // 消费消息
18. // 参数1 消息队列的消息, 队列名称
19. // 参数2 开启消息的确认机制
20. // 参数3 消息时的回调接口
21. channel.basicConsume("work", true, new DefaultConsumer(channel) {
22. // 最后一个参数 消息队列中取出的消息
23. // 默认分配是平均的
24. @Override
25. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
26. System.out.println("消费者-1" + new String(body));
27. try {
28. Thread.sleep(1000);
29. } catch (InterruptedException e) {
30. e.printStackTrace();
31. }
32. }
33. });
34. // channel.close();
35. // connection.close();
36. }
37. }

**开发消费者-2**

1. /\*\*
2. \* 自动确认消费 autoAck true 12搭配测试
3. \* <p>
4. \* 消费者 2
5. \*
6. \* @author mxz
7. \*/
8. @Component
9. public class CustomerTwo {
10. public static void main(String[] args) throws IOException {
11. // 获取连接对象
12. Connection connection = RabbitMQUtils.getConnection();
13. // 创建通道
14. Channel channel = connection.createChannel();
15. // 通道绑定对象
16. channel.queueDeclare("work", true, false, false, null);
17. channel.basicConsume("work", true, new DefaultConsumer(channel) {
18. // 最后一个参数 消息队列中取出的消息
19. @Override
20. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
21. System.out.println("消费者-1" + new String(body));
22. }
23. });
24. // channel.close();
25. // connection.close();
26. }
27. }

**测试结果**





总结:默认情况下，RabbitMQ将按顺序将每个消息发送给下一个使用者。平均而言，每个消费者都会收到相同数量的消息。这种分发消息的方式称为循环。

**消息自动确认机制**

Doing a task can take a few seconds. You may wonder what happens if one of the consumers starts a long task and dies with it only partly done. With our current code, once RabbitMQ delivers a message to the consumer it immediately marks it for deletion. In this case, if you kill a worker we will lose the message it was just processing. We’ll also lose all the messages that were dispatched to this particular worker but were not yet handled.

But we don’t want to lose any tasks. If a worker dies, we’d like the task to be delivered to another worker.

**消费者3**

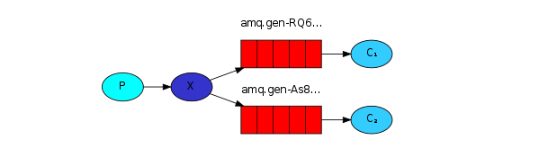
1. /\*\*
2. \* 能者多劳 34 搭配测试
3. \* <p>
4. \* 消费者 3
5. \*
6. \* @author mxz
7. \*/
8. @Component
9. public class CustomerThree {
10. public static void main(String[] args) throws IOException, TimeoutException {
11. // 获取连接对象
12. Connection connection = RabbitMQUtils.getConnection();
13. // 创建通道
14. Channel channel = connection.createChannel();
15. // 每一次只能消费一个消息
16. channel.basicQos(1);
17. // 通道绑定对象
18. channel.queueDeclare("work", true, false, false, null);
19. // 参数1 队列名称 参数2(autoAck) 消息自动确认 true 消费者自动向 rabbitMQ 确认消息消费 false 不会自动确认消息
20. // 若出现消费者宕机情况 消费者三可以进行消费
21. channel.basicConsume("work", false, new DefaultConsumer(channel) {
22. // 最后一个参数 消息队列中取出的消息
23. // 默认分配是平均的
24. @Override
25. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
26. System.out.println("消费者-1" + new String(body));
27. // 手动确认 参数1 确认队列中
28. channel.basicAck(envelope.getDeliveryTag(), false);
29. try {
30. Thread.sleep(1000);
31. } catch (InterruptedException e) {
32. e.printStackTrace();
33. }
34. }
35. });
36. // channel.close();
37. // connection.close();
38. }
39. }

**消费者4**

1. /\*\*
2. \* 能者多劳 34 搭配测试
3. \* <p>
4. \* 消费者 4
5. \*
6. \* @author mxz
7. \*/
8. @Component
9. public class CustomerFour {
10. public static void main(String[] args) throws IOException {
11. // 获取连接对象
12. Connection connection = RabbitMQUtils.getConnection();
13. // 创建通道
14. Channel channel = connection.createChannel();
15. // 每一次只能消费一个消息
16. channel.basicQos(1);
17. // 通道绑定对象
18. channel.queueDeclare("work", true, false, false, null);
19. channel.basicConsume("work", false, new DefaultConsumer(channel) {
20. // 最后一个参数 消息队列中取出的消息
21. @Override
22. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
23. System.out.println("消费者-1" + new String(body));
24. // 手动确认 参数1 手动确认
25. channel.basicAck(envelope.getDeliveryTag(), false);
26. }
27. });
28. // channel.close();
29. // connection.close();
30. }
31. }

**第三种模型(fanout)**

fanout 扇出 也称为广播



在广播模式下，消息发送流程是这样的：

* 可以有多个消费者
* 每个**消费者有自己的queue**（队列）
* 每个**队列都要绑定到Exchange**（交换机）
* **生产者发送的消息，只能发送到交换机**，交换机来决定要发给哪个队列，生产者无法决定。
* 交换机把消息发送给绑定过的所有队列
* 队列的消费者都能拿到消息。实现一条消息被多个消费者消费

**开发开发生产者**

1. /\*\*
2. \* 生产者
3. \* <p>
4. \* 任务模型 fanout
5. \*
6. \* @author mxz
7. \*/
8. @Component
9. public class Provider {
10. public static void main(String[] args) throws IOException, TimeoutException {
11. Connection connection = RabbitMQUtils.getConnection();
12. Channel channel = connection.createChannel();
13. // 将通道声明指定交换机 参数1 交换机名称 参数2 代表交换机类型 fanout 广播类型
14. channel.exchangeDeclare("logs", "fanout");
15. // 发送消息
16. channel.basicPublish("logs", "", null, "fanout type message".getBytes());
17. // 关闭资源
18. RabbitMQUtils.closeConnectionAndChannel(channel, connection);
19. }
20. }

**开发消费者**

* 消费者 1

1. /\*\*
2. \* 消费者 1
3. \* <p>
4. \* 任务模型 fanout
5. \*
6. \* @author mxz
7. \*/
8. public class CustomerOne {
9. public static void main(String[] args) throws IOException {
10. Connection connection = RabbitMQUtils.getConnection();
11. Channel channel = connection.createChannel();
12. // 通道绑定交换机
13. channel.exchangeDeclare("logs", "fanout");
14. // 临时队列
15. String queue = channel.queueDeclare().getQueue();
16. // 绑定交换机队列
17. channel.queueBind(queue, "logs", "");
18. // 消费消息
19. channel.basicConsume(queue, true, new DefaultConsumer(channel) {
20. @Override
21. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
22. System.out.println("消费者1 " + new String(body));
23. }
24. });
25. }
26. }

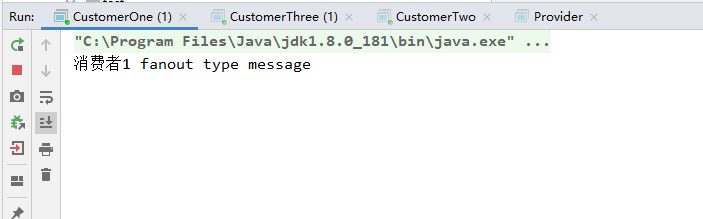
* 消费者 2

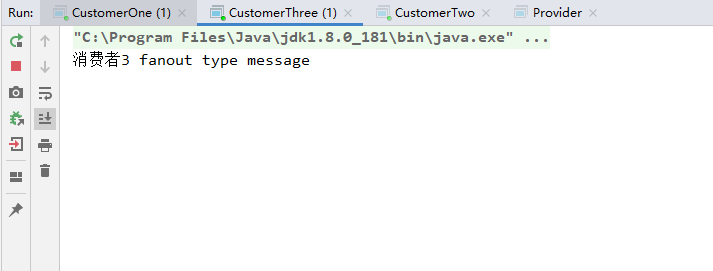
1. /\*\*
2. \* 消费者 2
3. \* <p>
4. \* 任务模型 fanout
5. \*
6. \* @author mxz
7. \*/
8. public class CustomerTwo {
9. public static void main(String[] args) throws IOException {
10. Connection connection = RabbitMQUtils.getConnection();
11. Channel channel = connection.createChannel();
12. // 通道绑定交换机
13. channel.exchangeDeclare("logs", "fanout");
14. // 临时队列
15. String queue = channel.queueDeclare().getQueue();
16. // 绑定交换机队列
17. channel.queueBind(queue, "logs", "");
18. // 消费消息
19. channel.basicConsume(queue, true, new DefaultConsumer(channel) {
20. @Override
21. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
22. System.out.println("消费者2 " + new String(body));
23. }
24. });
25. }
26. }

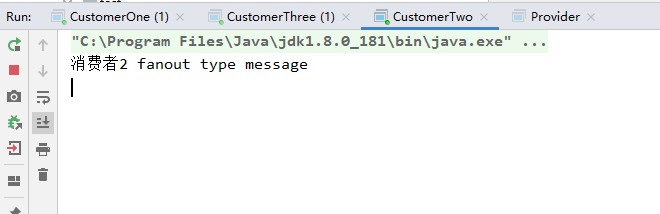
* 消费者 3

1. /\*\*
2. \* 消费者 3
3. \* <p>
4. \* 任务模型 fanout
5. \*
6. \* @author mxz
7. \*/
8. public class CustomerThree {
9. public static void main(String[] args) throws IOException {
10. Connection connection = RabbitMQUtils.getConnection();
11. Channel channel = connection.createChannel();
12. // 通道绑定交换机
13. channel.exchangeDeclare("logs", "fanout");
14. // 临时队列
15. String queue = channel.queueDeclare().getQueue();
16. // 绑定交换机队列
17. channel.queueBind(queue, "logs", "");
18. // 消费消息
19. channel.basicConsume(queue, true, new DefaultConsumer(channel) {
20. @Override
21. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
22. System.out.println("消费者3 " + new String(body));
23. }
24. });
25. }
26. }

**测试结果**







**第四种模型(Routing)**

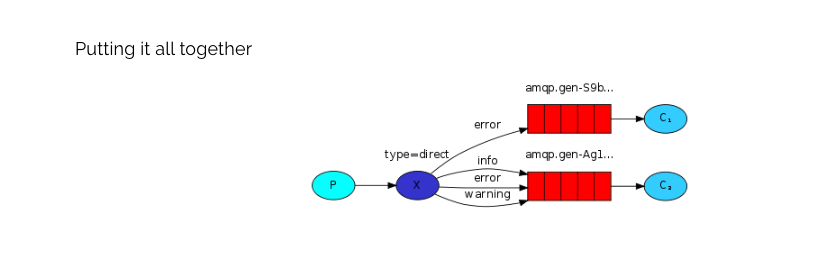
**Routing 之订阅模型-Direct(直连)**

在Fanout模式中，一条消息，会被所有订阅的队列都消费。但是，在某些场景下，我们希望不同的消息被不同的队列消费。这时就要用到Direct类型的Exchange。

在Direct模型下：

* 队列与交换机的绑定，不能是任意绑定了，而是要指定一个RoutingKey（路由key）
* 消息的发送方在 向 Exchange发送消息时，也必须指定消息的 RoutingKey。
* Exchange不再把消息交给每一个绑定的队列，而是根据消息的Routing Key进行判断，只有队列的Routingkey与消息的 Routing key完全一致，才会接收到消息

流程:



图解：

* P：生产者，向Exchange发送消息，发送消息时，会指定一个routing key。
* X：Exchange（交换机），接收生产者的消息，然后把消息递交给 与routing key完全匹配的队列
* C1：消费者，其所在队列指定了需要routing key 为 error 的消息
* C2：消费者，其所在队列指定了需要routing key 为 info、error、warning 的消息

**开发生产者**

1. /\*\*
2. \* @author mxz
3. \*/
4. public class Provider {
5. public static void main(String[] args) throws IOException {
6. Connection connection = RabbitMQUtils.getConnection();
7. Channel channel = connection.createChannel();
8. // 通过通道声明交换机 参数1 交换机名称 参数2 路由模式
9. channel.exchangeDeclare("logs\_direct", "direct");
10. // 发送消息
11. String routingKey = "error";
12. channel.basicPublish("logs\_direct", routingKey, null, ("这是 direct 模式发布基于 route\_key [" + routingKey + "]").getBytes());
13. // 关闭资源
14. RabbitMQUtils.closeConnectionAndChannel(channel, connection);
15. }
16. }

**开发消费者**

* 消费者1

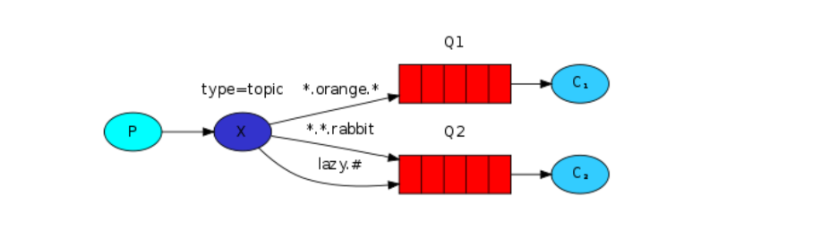
1. /\*\*
2. \* 消费者 1
3. \*
4. \* @author mxz
5. \*/
6. @Component
7. public class CustomerOne {
8. public static void main(String[] args) throws IOException, TimeoutException {
9. // 获取连接对象
10. Connection connection = RabbitMQUtils.getConnection();
11. // 创建通道
12. Channel channel = connection.createChannel();
13. // 创建一个临时队列
14. String queue = channel.queueDeclare().getQueue();
15. // 基于 route\_key 绑定队列交换机
16. channel.queueBind(queue, "logs\_direct", "error");
17. // 消费消息
18. channel.basicConsume(queue, true, new DefaultConsumer(channel) {
19. @Override
20. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
21. System.out.println("消费者1： " + new String(body));
22. }
23. });
24. // channel.close();
25. // connection.close();
26. }
27. }

* 消费者2

1. /\*\*
2. \* 消费者 2
3. \*
4. \* @author mxz
5. \*/
6. @Component
7. public class CustomerTwo {
8. public static void main(String[] args) throws IOException, TimeoutException {
9. Connection connection = RabbitMQUtils.getConnection();
10. Channel channel = connection.createChannel();
11. // 声明交换机
12. channel.exchangeDeclare("logs\_direct", "direct");
13. // 创建一个临时队列
14. String queue = channel.queueDeclare().getQueue();
15. // 临时队列和绑定交换机
16. channel.queueBind(queue, "logs\_direct", "info");
17. channel.queueBind(queue, "logs\_direct", "error");
18. channel.queueBind(queue, "logs\_direct", "warning");
19. // 消费消息
20. channel.basicConsume(queue, true, new DefaultConsumer(channel) {
21. @Override
22. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
23. System.out.println("消费者2：" + new String(body));
24. }
25. });
26. }
27. }

**Routing 之订阅模型-Topic**

Topic类型的Exchange与Direct相比，都是可以根据RoutingKey把消息路由到不同的队列。只不过Topic类型Exchange可以让队列在绑定Routing key 的时候使用通配符！这种模型Routingkey 一般都是由一个或多个单词组成，多个单词之间以”.”分割，例如： item.insert



1. # 统配符
2. \* (star) can substitute for exactly one word. 匹配不多不少恰好1个词
3. # (hash) can substitute for zero or more words. 匹配一个或多个词
4. # 如:
5. audit.# 匹配audit.irs.corporate或者 audit.irs 等
6. audit.\* 只能匹配 audit.irs

**开发生产者**

1. /\*\*
2. \* 生产者
3. \* <p>
4. \*
5. \* @author mxz
6. \*/
7. @Component
8. public class Provider {
9. public static void main(String[] args) throws IOException, TimeoutException {
10. Connection connection = RabbitMQUtils.getConnection();
11. Channel channel = connection.createChannel();
12. // 声明交换机以及交换机类型
13. channel.exchangeDeclare("topics", "topic");
14. // 路由key
15. String routeKey = "user.save";
16. channel.basicPublish("topics", routeKey, null, ("这里是 topic 动态路由模型, routeKey:[" + routeKey + "]").getBytes());
17. // 关闭资源
18. RabbitMQUtils.closeConnectionAndChannel(channel, connection);
19. }
20. }

**开发消费者**

* 消费者

1. /\*\*
2. \* @author mxz
3. \*/
4. public class CustomerOne {
5. public static void main(String[] args) throws IOException {
6. Connection connection = RabbitMQUtils.getConnection();
7. Channel channel = connection.createChannel();
8. // 声明交换机以及交换机类型
9. channel.exchangeDeclare("topics", "topic");
10. // 创建一个临时队列
11. String queue = channel.queueDeclare().getQueue();
12. // 绑定队列和交换机 动态通配符 route key
13. channel.queueBind(queue, "topics", "user.\*");
14. // 消费消息
15. channel.basicConsume(queue, true, new DefaultConsumer(channel) {
16. @Override
17. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
18. System.out.println("消费者1：" + new String(body));
19. }
20. });
21. }
22. }

* 消费者

1. /\*\*
2. \* @author mxz
3. \*/
4. public class CustomerTwo {
5. public static void main(String[] args) throws IOException {
6. Connection connection = RabbitMQUtils.getConnection();
7. Channel channel = connection.createChannel();
8. // 声明交换机以及交换机类型
9. channel.exchangeDeclare("topics", "topic");
10. // 创建一个临时队列
11. String queue = channel.queueDeclare().getQueue();
12. // 绑定队列和交换机 动态通配符 route key
13. channel.queueBind(queue, "topics", "user.#");
14. // 消费消息
15. channel.basicConsume(queue, true, new DefaultConsumer(channel) {
16. @Override
17. public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
18. System.out.println("消费者2：" + new String(body));
19. }
20. });
21. }
22. }

文章已上传gitee <https://gitee.com/codingce/hexo-blog>  
项目地址github: <https://github.com/xzMhehe/codingce-java>

标签：RabbitMQ 的第一个程序

版权声明：本文为博主原创文章，转载请附上原文出处链接和本声明，KuangStudy,以学为伴，一生相伴！

[本文链接：https://www.kuangstudy.com/bbs/1352890836446158849](https://www.kuangstudy.com/bbs/1352890836446158849)