# RabbitMQ集中典型使用场景

RabbitMQ主页：<https://www.rabbitmq.com/>

学习RabbitMQ的使用场景，来自官方教程：<https://www.rabbitmq.com/getstarted.html>

**场景1：单发送单接收**

使用场景：简单的发送与接收，没有特别的处理。



**Producer：**

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

public class Send {  
  
private final static String QUEUE\_NAME = "hello";

public static void main(String[] argv) throws Exception {  
  
ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE\_NAME, false, false, false, null);  
String message = "Hello World!";  
channel.basicPublish("", QUEUE\_NAME, null, message.getBytes());  
System.out.println(" [x] Sent '" + message + "'");  
  
channel.close();  
connection.close();  
}  
}

**Consumer：**

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

public class Recv {  
  
private final static String QUEUE\_NAME = "hello";

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

ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE\_NAME, false, false, false, null);  
System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");  
  
QueueingConsumer consumer = new QueueingConsumer(channel);  
channel.basicConsume(QUEUE\_NAME, true, consumer);  
  
while (true) {  
QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
String message = new String(delivery.getBody());  
System.out.println(" [x] Received '" + message + "'");  
}  
}  
}

**场景2：单发送多接收**

使用场景：一个发送端，多个接收端，如分布式的任务派发。为了保证消息发送的可靠性，不丢失消息，使消息持久化了。同时为了防止接收端在处理消息时down掉，只有在消息处理完成后才发送ack消息。



**Producer：**

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

public class NewTask {  
  
private static final String TASK\_QUEUE\_NAME = "task\_queue";

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

ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();  
  
channel.queueDeclare(TASK\_QUEUE\_NAME, true, false, false, null);  
  
String message = getMessage(argv);  
  
channel.basicPublish( "", TASK\_QUEUE\_NAME,   
MessageProperties.PERSISTENT\_TEXT\_PLAIN,  
message.getBytes());  
System.out.println(" [x] Sent '" + message + "'");  
  
channel.close();  
connection.close();  
}  
  
private static String getMessage(String[] strings){  
if (strings.length < 1)  
return "Hello World!";  
return joinStrings(strings, " ");  
}   
  
private static String joinStrings(String[] strings, String delimiter) {  
int length = strings.length;  
if (length == 0) return "";  
StringBuilder words = new StringBuilder(strings[0]);  
for (int i = 1; i < length; i++) {  
words.append(delimiter).append(strings[i]);  
}  
return words.toString();  
}  
}

发送端和场景1不同点：

1、使用“task\_queue”声明了另一个Queue，因为RabbitMQ不容许声明2个相同名称、配置不同的Queue

2、使"task\_queue"的Queue的durable的属性为true，即使消息队列durable

3、使用MessageProperties.PERSISTENT\_TEXT\_PLAIN使消息durable

When RabbitMQ quits or crashes it will forget the queues and messages unless you tell it not to. Two things are required to make sure that messages aren't lost: we need to mark both the queue and messages as durable.

**Consumer：**

import com.rabbitmq.client.ConnectionFactory;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.QueueingConsumer;  
  
public class Worker {

private static final String TASK\_QUEUE\_NAME = "task\_queue";

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

ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();  
  
channel.queueDeclare(TASK\_QUEUE\_NAME, true, false, false, null);  
System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");  
  
channel.basicQos(1);  
  
QueueingConsumer consumer = new QueueingConsumer(channel);  
channel.basicConsume(TASK\_QUEUE\_NAME, false, consumer);  
  
while (true) {  
QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
String message = new String(delivery.getBody());  
  
System.out.println(" [x] Received '" + message + "'");  
doWork(message);  
System.out.println(" [x] Done");

channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
}   
}  
  
private static void doWork(String task) throws InterruptedException {  
for (char ch: task.toCharArray()) {  
if (ch == '.') Thread.sleep(1000);  
}  
}  
}

接收端和场景1不同点：

1、使用“task\_queue”声明消息队列，并使消息队列durable

2、在使用channel.basicConsume接收消息时使autoAck为false，即不自动会发ack，由channel.basicAck()在消息处理完成后发送消息。

3、使用了channel.basicQos(1)保证在接收端一个消息没有处理完时不会接收另一个消息，即接收端发送了ack后才会接收下一个消息。在这种情况下发送端会尝试把消息发送给下一个not busy的接收端。

注意点：

1）It's a common mistake to miss the basicAck. It's an easy error, but the consequences are serious. Messages will be redelivered when your client quits (which may look like random redelivery), but RabbitMQ will eat more and more memory as it won't be able to release any unacked messages.

2）Note on message persistence

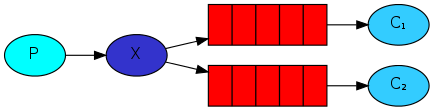
Marking messages as persistent doesn't fully guarantee that a message won't be lost. Although it tells RabbitMQ to save the message to disk, there is still a short time window when RabbitMQ has accepted a message and hasn't saved it yet. Also, RabbitMQ doesn't do fsync(2) for every message -- it may be just saved to cache and not really written to the disk. The persistence guarantees aren't strong, but it's more than enough for our simple task queue. If you need a stronger guarantee you can wrap the publishing code in atransaction.

3）Note about queue size

If all the workers are busy, your queue can fill up. You will want to keep an eye on that, and maybe add more workers, or have some other strategy.

**场景3：Publish/Subscribe**

使用场景：发布、订阅模式，发送端发送广播消息，多个接收端接收。



**Producer：**

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

public class EmitLog {

private static final String EXCHANGE\_NAME = "logs";

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

ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE\_NAME, "fanout");

String message = getMessage(argv);

channel.basicPublish(EXCHANGE\_NAME, "", null, message.getBytes());  
System.out.println(" [x] Sent '" + message + "'");

channel.close();  
connection.close();  
}  
  
private static String getMessage(String[] strings){  
if (strings.length < 1)  
return "info: Hello World!";  
return joinStrings(strings, " ");  
}  
  
private static String joinStrings(String[] strings, String delimiter) {  
int length = strings.length;  
if (length == 0) return "";  
StringBuilder words = new StringBuilder(strings[0]);  
for (int i = 1; i < length; i++) {  
words.append(delimiter).append(strings[i]);  
}  
return words.toString();  
}  
}

发送端：

发送消息到一个名为“logs”的exchange上，使用“fanout”方式发送，即广播消息，不需要使用queue，发送端不需要关心谁接收。

Consumer：

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

public class ReceiveLogs {

private static final String EXCHANGE\_NAME = "logs";

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

ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE\_NAME, "fanout");  
String queueName = channel.queueDeclare().getQueue();  
channel.queueBind(queueName, EXCHANGE\_NAME, "");  
  
System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");

QueueingConsumer consumer = new QueueingConsumer(channel);  
channel.basicConsume(queueName, true, consumer);

while (true) {  
QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
String message = new String(delivery.getBody());

System.out.println(" [x] Received '" + message + "'");   
}  
}  
}

接收端：

1、声明名为“logs”的exchange的，方式为"fanout"，和发送端一样。

2、channel.queueDeclare().getQueue();该语句得到一个随机名称的Queue，该queue的类型为non-durable、exclusive、auto-delete的，将该queue绑定到上面的exchange上接收消息。

3、注意binding queue的时候，channel.queueBind()的第三个参数Routing key为空，即所有的消息都接收。如果这个值不为空，在exchange type为“fanout”方式下该值被忽略！

**场景4：Routing (按路线发送接收)**

使用场景：发送端按routing key发送消息，不同的接收端按不同的routing key接收消息。



**Producer：**

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

public class EmitLogDirect {

private static final String EXCHANGE\_NAME = "direct\_logs";

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

ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE\_NAME, "direct");

String severity = getSeverity(argv);  
String message = getMessage(argv);

channel.basicPublish(EXCHANGE\_NAME, severity, null, message.getBytes());  
System.out.println(" [x] Sent '" + severity + "':'" + message + "'");

channel.close();  
connection.close();  
}  
  
private static String getSeverity(String[] strings){  
if (strings.length < 1)  
return "info";  
return strings[0];  
}

private static String getMessage(String[] strings){   
if (strings.length < 2)  
return "Hello World!";  
return joinStrings(strings, " ", 1);  
}  
  
private static String joinStrings(String[] strings, String delimiter, int startIndex) {  
int length = strings.length;  
if (length == 0 ) return "";  
if (length < startIndex ) return "";  
StringBuilder words = new StringBuilder(strings[startIndex]);  
for (int i = startIndex + 1; i < length; i++) {  
words.append(delimiter).append(strings[i]);  
}  
return words.toString();  
}  
}

发送端和场景3的区别：

1、exchange的type为direct

2、发送消息的时候加入了routing key

**Consumer：**

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

public class ReceiveLogsDirect {

private static final String EXCHANGE\_NAME = "direct\_logs";

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

ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE\_NAME, "direct");  
String queueName = channel.queueDeclare().getQueue();  
  
if (argv.length < 1){  
System.err.println("Usage: ReceiveLogsDirect [info] [warning] [error]");  
System.exit(1);  
}  
  
for(String severity : argv){   
channel.queueBind(queueName, EXCHANGE\_NAME, severity);  
}  
  
System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");

QueueingConsumer consumer = new QueueingConsumer(channel);  
channel.basicConsume(queueName, true, consumer);

while (true) {  
QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
String message = new String(delivery.getBody());  
String routingKey = delivery.getEnvelope().getRoutingKey();

System.out.println(" [x] Received '" + routingKey + "':'" + message + "'");   
}  
}  
}

接收端和场景3的区别：

在绑定queue和exchange的时候使用了routing key，即从该exchange上只接收routing key指定的消息。

**场景5：Topics (按topic发送接收)**

使用场景：发送端不只按固定的routing key发送消息，而是按字符串“匹配”发送，接收端同样如此。



**Producer：**

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

public class EmitLogTopic {

private static final String EXCHANGE\_NAME = "topic\_logs";

public static void main(String[] argv) {  
Connection connection = null;  
Channel channel = null;  
try {  
ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
  
connection = factory.newConnection();  
channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE\_NAME, "topic");

String routingKey = getRouting(argv);  
String message = getMessage(argv);

channel.basicPublish(EXCHANGE\_NAME, routingKey, null, message.getBytes());  
System.out.println(" [x] Sent '" + routingKey + "':'" + message + "'");

}  
catch (Exception e) {  
e.printStackTrace();  
}  
finally {  
if (connection != null) {  
try {  
connection.close();  
}  
catch (Exception ignore) {}  
}  
}  
}  
  
private static String getRouting(String[] strings){  
if (strings.length < 1)  
return "anonymous.info";  
return strings[0];  
}

private static String getMessage(String[] strings){   
if (strings.length < 2)  
return "Hello World!";  
return joinStrings(strings, " ", 1);  
}  
  
private static String joinStrings(String[] strings, String delimiter, int startIndex) {  
int length = strings.length;  
if (length == 0 ) return "";  
if (length < startIndex ) return "";  
StringBuilder words = new StringBuilder(strings[startIndex]);  
for (int i = startIndex + 1; i < length; i++) {  
words.append(delimiter).append(strings[i]);  
}  
return words.toString();  
}  
}

发送端和场景4的区别：

1、exchange的type为topic

2、发送消息的routing key不是固定的单词，而是匹配字符串，如"\*.lu.#"，\*匹配一个单词，#匹配0个或多个单词。

**Consumer：**

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

public class ReceiveLogsTopic {

private static final String EXCHANGE\_NAME = "topic\_logs";

public static void main(String[] argv) {  
Connection connection = null;  
Channel channel = null;  
try {  
ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");  
  
connection = factory.newConnection();  
channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE\_NAME, "topic");  
String queueName = channel.queueDeclare().getQueue();  
  
if (argv.length < 1){  
System.err.println("Usage: ReceiveLogsTopic [binding\_key]...");  
System.exit(1);  
}  
  
for(String bindingKey : argv){   
channel.queueBind(queueName, EXCHANGE\_NAME, bindingKey);  
}  
  
System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");

QueueingConsumer consumer = new QueueingConsumer(channel);  
channel.basicConsume(queueName, true, consumer);

while (true) {  
QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
String message = new String(delivery.getBody());  
String routingKey = delivery.getEnvelope().getRoutingKey();

System.out.println(" [x] Received '" + routingKey + "':'" + message + "'");   
}  
}  
catch (Exception e) {  
e.printStackTrace();  
}  
finally {  
if (connection != null) {  
try {  
connection.close();  
}  
catch (Exception ignore) {}  
}  
}  
}  
}

接收端和场景4的区别：

1、exchange的type为topic

2、接收消息的routing key不是固定的单词，而是匹配字符串。

注意点：

Topic exchange

Topic exchange is powerful and can behave like other exchanges. When a queue is bound with "#" (hash) binding key - it will receive all the messages, regardless of the routing key - like in fanout exchange. When special characters "\*" (star) and "#" (hash) aren't used in bindings, the topic exchange will behave just like a direct one.