Rabbit Tutorials

# Hello World 示例

**"Hello World"** (using the Java Client)

In this part of the tutorial we'll write two programs in Java; a producer that sends a single message, and a consumer that receives messages and prints them out. We'll gloss over some of the detail in the Java API, concentrating on this very simple thing just to get started. It's a "Hello World" of messaging.

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.



## The Java client library

**RabbitMQ** speaks multiple protocols. This tutorial uses **AMQP 0-9-1**, which is an open, general-purpose protocol for messaging. There are a number of clients for **RabbitMQ** in many different languages. We'll use the Java client provided by RabbitMQ.

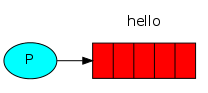
Download the client library and its dependencies (SLF4J API and SLF4J Simple). Copy those files in your working directory, along the tutorials Java files.

Please note **SLF4J Simple** is enough for tutorials but you should use a **full-blown logging library** like Logback in production.

(The RabbitMQ Java client is also **in the central Maven repository**, with the groupId com.rabbitmq and the artifactId amqp-client.)

Now we have the Java client and its dependencies, we can write some code.

## Sending



We'll call our message publisher (sender) Send and our message consumer (receiver) Recv. The publisher will connect to RabbitMQ, **send a single message**, then exit.

### In Send.java, we need some classes imported:

import com.rabbitmq.client.**ConnectionFactory**;

import com.rabbitmq.client.**Connection**;

import com.rabbitmq.client.**Channel**;

### Set up the class and name the queue:

public class Send {

private final static String QUEUE\_NAME = "hello";

public static void main(String[] argv)

throws java.io.IOException {

...

}

}

### then we **can create a connection** to the server and create a channel：

ConnectionFactory factory = new ConnectionFactory();

factory.setHost("localhost");

Connection connection = factory.newConnection();

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

The connection abstracts the socket connection, and takes care of protocol version negotiation and authentication and so on for us. Here we connect to a broker on the local machine - hence the localhost. If we wanted to connect to a broker on a different machine we'd simply specify its name or IP address here.

Next we create a channel, which is where most of the API for getting things done resides.

### **declare a queue：声明一个队列**

To send, we must **declare a queue** for us to send to; then we can publish a message to the queue:

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 + "'");

Declaring a queue is **idempotent** - it will only be created if it doesn't exist already. The message content is **a byte array**, so you can encode whatever you like there.

Lastly, we close the channel and the connection;

**channel.close();**

**connection.close();**

### **Send.java**

[Here's the whole Send.java class](http://github.com/rabbitmq/rabbitmq-tutorials/blob/master/java/Send.java).

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.Connection;

import com.rabbitmq.client.ConnectionFactory;

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("UTF-8"));

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

channel.close();

connection.close();

}

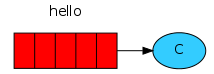
}

Sending doesn't work!

If this is your first time using RabbitMQ and you don't see the "Sent" message then you may be left scratching your head wondering what could be wrong. Maybe the broker was started **without enough free disk space** (by default it needs at least 200 MB free) and is therefore refusing to accept messages. Check the broker logfile to confirm and reduce the limit if necessary. The configuration file documentation will show you how to set **disk\_free\_limit**.

## Receiving

That's it for our publisher. **Our consumer is pushed messages from RabbitMQ**, so unlike the publisher which publishes a single message, we'll keep it running to listen for messages and print them out.



The code (in **Recv.java**) has almost the same imports as Send:

import com.rabbitmq.client.ConnectionFactory;

import com.rabbitmq.client.Connection;

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.Consumer;

import com.rabbitmq.client.DefaultConsumer;

The extra **DefaultConsumer** is a class implementing the Consumer interface we'll use to buffer the messages pushed to us by the server.

Setting up is the same as the publisher; we open a connection and a channel, and declare the queue from which we're going to consume. Note this matches up with the queue that send publishes to.

public class Recv {

private final static String QUEUE\_NAME = "hello";

public static void main(String[] argv)

throws java.io.IOException,

java.lang.InterruptedException {

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");

...

}

}

Note that we declare the queue here, as well. Because we might start the consumer before the publisher, we want to make sure the queue exists before we try to consume messages from it.

We're about to tell the server to deliver us the messages from the queue. Since it will push us messages **asynchronously**, we provide a callback in the form of an object that will buffer the messages until we're ready to use them. That is what a **DefaultConsumer** subclass does.

Consumer consumer = new DefaultConsumer(channel) {

@Override

public void handleDelivery(String consumerTag, Envelope envelope,

AMQP.BasicProperties properties, **byte[] body**)

throws IOException {

String message = new String(body, "UTF-8");

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

}

};

channel.basicConsume(QUEUE\_NAME, true, consumer);

Recv.java

[Here's the whole Recv.java class](http://github.com/rabbitmq/rabbitmq-tutorials/blob/master/java/Recv.java).

import com.rabbitmq.client.\*;

import java.io.IOException;

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");

Consumer consumer = new DefaultConsumer(channel) {

@Override

public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body)

throws IOException {

String message = new String(body, "UTF-8");

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

}

};

channel.basicConsume(QUEUE\_NAME, true, consumer);

}

}

## Putting it all together

You can compile both of these with just the RabbitMQ java client on the classpath:

javac -cp amqp-client-4.0.2.jar Send.java Recv.java

To run them, you'll need rabbitmq-client.jar and its dependencies on the classpath. In a terminal, run the consumer (receiver):

java -cp .:amqp-client-4.0.2.jar:slf4j-api-1.7.21.jar:slf4j-simple-1.7.22.jar Recv

then, run the publisher (sender):

java -cp .:amqp-client-4.0.2.jar:slf4j-api-1.7.21.jar:slf4j-simple-1.7.22.jar Send

On Windows, use a semicolon instead of a colon to separate items in the classpath.

The consumer will print the message it gets from the publisher via RabbitMQ. The consumer will keep running, waiting for messages (Use Ctrl-C to stop it), so try running the publisher from another terminal.

## Listing queues

You may wish to see what queues **RabbitMQ** has and how many messages are in them. You can do it (as a privileged user) using the rabbitmqctl tool:

**sudo rabbitmqctl list\_queues**

On Windows, omit the sudo:

**rabbitmqctl.bat list\_queues**

**(在sbin目录下有该工具。)**