01-RabbitMQ 开篇 Hello World

先决条件

本教程假定 RabbitMQ 已在标准端口（5672）上的 localhost 上安装并运行。如果使用不同的主机，端口或凭据，连接设置将需要调整。

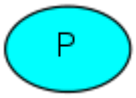
介绍

RabbitMQ 是一个消息代理：它接受并转发消息。您可以将其视为邮局：当您将要发布的邮件放在邮箱中时，您可以确信邮差最终会将邮件发送给收件人。在这个比喻中，RabbitMQ 是一个邮箱，邮局和邮递员。

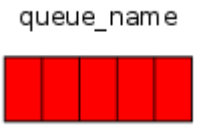
RabbitMQ 和邮局之间的主要区别在于它不处理纸张，而是接受，存储和转发二进制数据块的消息。

RabbitMQ 和消息传递一般使用一些术语。

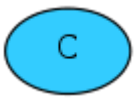
生产意味着不仅仅是发送消息。发送消息的程序是一个生产者：



队列是存在于 RabbitMQ 中的邮箱的名称。虽然消息流过 RabbitMQ 和您的应用程序，但它们只能存储在队列中。一个队列仅由主机的存储器和磁盘限制约束，它本质上是一个大的消息缓冲器。多个生产者可以发送消息到到一个队列，多个消费者可以尝试从一个队列接收数据。下图是代表一个队列的示意图



消费具有与接收相似的含义。一个消费者是一个程序，主要是等待接收信息：



请注意，生产者，消费者和消息代理（Broker）不必驻留在同一个主机上；确实在大多数应用程序中，它们不是。

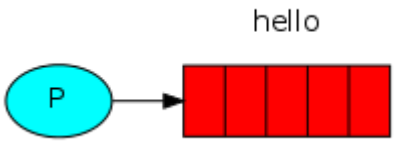
Hello World

在本教程的这一部分，我们将用 Java 编写两个程序；发送单个消息的生产者，以及接收消息并将其打印出来的消费者。我们将介绍 Java API 中的一些细节，专注于这个非常简单的事情，只需要开始。这是一个“Hello World”的消息传递。

在下图中，“P”是我们的生产者，“C”是我们的消费者。中间的框是队列 - RabbitMQ 代表消费者的消息缓冲区。

注意：RabbitMQ 使用了多种协议。本教程使用 AMQP 0-9-1，它是一种开放的通用协议，用于消息传递。有许多不同语言的 RabbitMQ 客户端。我们将使用 RabbitMQ 提供的 Java 客户端。

消息发送



在 Send.java 中，我们需要一些导入的类：

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

设置类并命名队列：

```
public  class  Send  {
   private  final  static String QUEUE_NAME = “hello” ;

   public  static  void  main （String [] argv）
      throws java.io.IOException {
      ...
  }
}
```

那么我们可以创建一个到服务器的连接：

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

这个连接抽象了套接字连接，并且为我们处理了协议版本协商和认证。接下来我们连接到本地。如果我们想连接到不同机器上的代理，我们可以在此处指定其名称或 IP 地址。

接下来，我们创建一个通道，这是大部分用于完成任务的 API 所在的地方。

要发送消息，我们必须申请一个队列给我们发送;我们可以将消息发布到队列中：

```
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 + "'");
```

声明队列是幂等的 - 只有当它不存在时才会被创建。消息内容是一个字节数组，所以你可以编码你喜欢的任何东西。

最后，我们关闭通道和连接;

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

接收消息

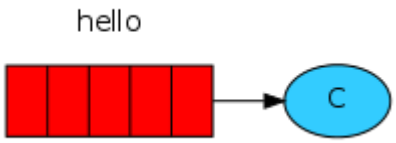
上面就是我们的发布者。我们的消费者是从 RabbitMQ 推送消息，所以不同于发布单个消息的发布者，我们的消息接收者将继续运行，以收听消息并打印出来。

代码（在 Recv.java 中）具有与发送几乎相同的导入：

```
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;
```

额外的 DefaultConsumer 是一个实现 Consumer 接口的类，用于缓冲由服务器推送给我们的消息。

设置与发布者相同; 我们打开一个连接和一个通道，并声明我们要消费的队列。注意这匹配发送者发送的队列。




```
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");
    ...
    }
}
```

请注意，我们也在这里声明队列。因为我们可能会在发布者之前启动消费者，所以我们要确保队列存在，然后再尝试从中消费消息。

我们即将告诉服务器将队列中的消息传递给我们。由于它会异步地推送我们的邮件，所以我们提供一个对象形式的回调，缓冲消息，直到我们准备好使用它们。这是一个 DefaultConsumer 子类。

```
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);
```

运行

先运行接收者

再运行发送者，