



RabbitMQ 研究与应用

一.背景介绍

1.1 相关概念

谈到 RabbitMQ,首先要谈到 MQ 和 AMQP. MQ 全称为 Message Queue, 消息队列 (MQ) 是一种应用程序对应用程序的通信方法。应用程序通过写和检索出入列队的针对应用程序的数据 (消息) 来通信, 而无需用专用连接来链接它们。AMQP:Advanced Message Queuing Protocol, 高级消息队列协议, 是应用层协议的一个开放标准, 为面向消息的中间件设计。AMQP 的主要特征是面向消息、队列、路由 (包括点对点和发布/订阅)、可靠性、安全性要求很严格。

AMQP 允许来自不同供应商的消息生产者和消费者实现真正的互操作扩展, 就如同 SMTP、HTTP、FTP 等协议采用的方式一样。而此前对于消息中间件的标准化努力则集中在 API 层面上(比如 JMS), 且没有提供互操作性的途径。不同于 JMS 的仅仅定义 API, AMQP 是一个线路级的协议——它描述了通过网络传输的字节流的数据格式。因此, 遵从这个协议的任何语言编写的工具均可以操作 AMQP 消息。

AMQP 的实现有:

- 1.OpenAMQAMQP 的开源实现, 用 C 语言编写, 运行于 Linux、AIX、Solaris、Windows、OpenVMS。
- 2.Apache QpidApache 的开源项目, 支持 C++、Ruby、Java、JMS、Python 和 .NET。
- 3.Redhat Enterprise MRG 实现了 AMQP 的最新版本 0-10, 提供了丰富的特征集, 比如完全管理、联合、Active-Active 集群, 有 Web 控制台, 还有许多企业级特征, 客户端支持 C++、Ruby、Java、JMS、Python 和 .NET。
- 4.RabbitMQ 一个独立的开源实现, 服务器端用 Erlang 语言编写, 支持多种客户端, 如: Python、Ruby、.NET、Java、JMS、C、PHP、ActionScript、XMPP、STOMP 等, 支持 AJAX。RabbitMQ 发布在 Ubuntu、FreeBSD 平台。
- 5.AMQP InfrastructureLinux 下, 包括 Broker、管理工具、Agent 和客户端。
- 6.?MQ 一个高性能的消息平台, 在分布式消息网络可作为兼容 AMQP 的 Broker 节点, 绑定了多种语言, 包括 Python、C、C++、Lisp、Ruby 等。
- 7.Zyre 是一个 Broker, 实现了 RestMS 协议和 AMQP 协议, 提供了 RESTful HTTP 访问网络 AMQP 的能力。

RabbitMQ 是由 LShift 提供的一个 AMQP 的开源实现, 由以高性能、健壮以及 Scalability 出名的 Erlang 写成, 因此也是继承了这些优点。ZeroMQ 和 RabbitMQ 都支持开源消息协议 AMQP, 不过 ZeroMQ 暂时不支持消息持久化和崩溃恢复, 且稳定度稍差。而 RabbitMQ 克服了这几大缺点, 因而成功的在这几大协议中脱颖而出。

1.2 诞生背景

AMQP 主要是由金融领域的软件专家们贡献的创意，而联合了通讯和软件方面的力量，一起打造出来的规范。

粗略的讲 AMQP 首先满足的是金融系统的消息通讯业务需求。这是一个可以和 JMS 进行类比的消息中间件开放规范，所不同的是 AMQP 同时定义了消息中间件的语义层面和协议层面；另外一个不同是 AMQP 是语言中立的，而 JMS 仅和 Java 相关。AMQP 在“语义层面的定义”，这就意味着，它并不仅仅是象 JMS 或者其他的 MQ 一样，仅能按照预定义的方式工作，而是“可编程”的消息中间件。而“语言中立”则意味着只要遵循 AMQP 的协议，任何一种语言都可以开发消息组件乃至中间件本身。

二. 基本原理

2.1 核心组件：Exchange & Queue

RabbitMQ 的两大核心组件是 Exchange 和 Queue，以下是它的运行原理图：

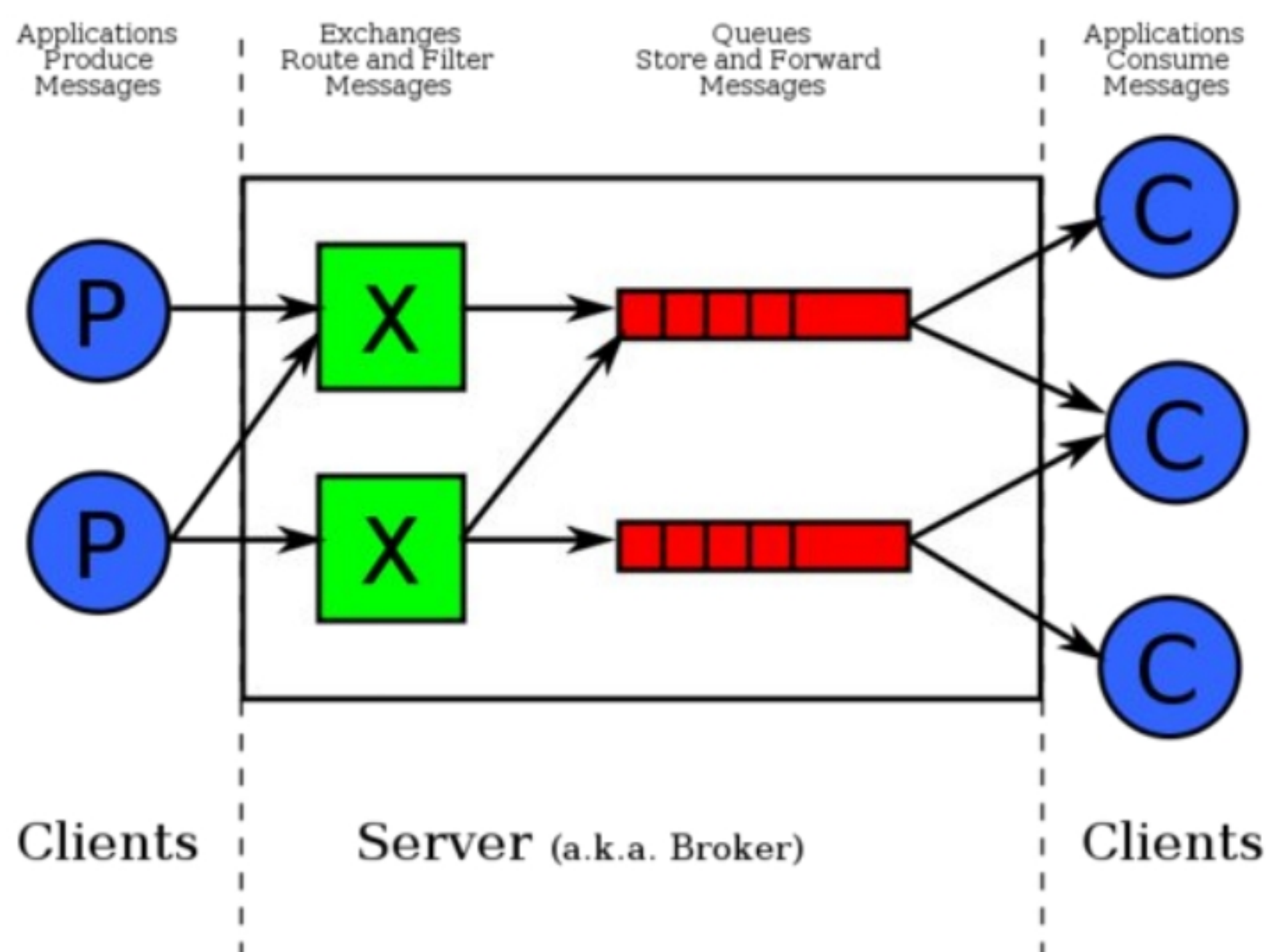


图 1 RabbitMQ 通信原理

图1中,绿色的 X 就是 Exchange ,红色的是 Queue ,这两者都在 Server 端, 又称作 Broker , 这部分是 RabbitMQ 实现的, 而蓝色的则是客户端, 通常有 Producer 和 Consumer 两种类型。

Exchange，又称交换器，它接受消息和路由信息，然后将消息发送给消息队列。对于每个虚拟主机内部，交换器有独一无二的名字。应用程序在其权限范围之内可以自由的创建、共享、使用和销毁交换器实例。

交换器可以是持久的、临时的或者自动删除的。持久交换器会一直存在于服务器，直到它被显式删除；临时交换器会工作到服务器被关闭时为止；而一旦没有应用程序使用自动删除交换器，它就会被服务器自动删除掉。

消息队列是一个具名缓冲区，它们代表一组消费者应用程序保存消息。应用程序在其权限范围之内可以自由的创建、共享、使用和消费消息队列。

消息队列提供了有限制的先进先出保证。服务器会将从某一个生产者发出的同等优先级的消息按照它们进入队列的顺序传递给某个消费者，万一某些消息不能被消费者处理，它们可能会被打乱顺序重新传递。

消息队列可以是持久的、临时的或者自动删除的。临时消息队列会工作到服务器被关闭时为止；而一旦没有应用程序使用自动删除消息队列，它就会被服务器自动删除掉。只要用户（客户端）拥有相应的权限，任何队列都可以被显式删除。

消息队列将消息保存在内存、硬盘，或者这两种介质的组合之中。消息队列限定在虚拟主机范围之内。消息队列保存消息，并将消息分发给一个或多个订阅客户端。

消息队列会跟踪消息的获取情况，消息要出队就必须被获取（**acquire** 和 **consume** 是两个动作，先执行 **acquire**，相当于对消息加锁）。这阻止了多个客户端同时获取和消费同一条消息，也可以被用来做单个队列多个消费者之间的负载均衡。

2.2 重要特性：持久化

持久化是 **RabbitMQ** 的一大重要特性。设想你花了大量的时间来创建队列、交换机，然后，服务器程序在不明原因下崩溃了，你的队列、交换机，甚至是放在队列里面但是尚未处理的消息都有可能消失。这种后果在某些场景下将会带来不会估计的损失。

然而 **RabbitMQ** 正有方法解决这一问题。

队列和交换机有一个创建时候指定的标志 **durable**（持久化）。**durable** 的唯一含义就是具有这个标志的队列和交换机会在重启之后重新建立，它不表示说在队列当中的消息会在重启后恢复。那么如何才能做到不只是队列和交换机，还有消息都是持久的呢？

但是首先一个问题是，我们需要考虑是否真的需要消息是持久化得？对于一个需要在重启之后回复的消息来说，它需要被写入到磁盘上，而即使是最简单的磁盘操作也是要消耗时间的。如果和消息的内容相比，你更看重的是消息处理的速度，那么不要使用持久化的消息。不过大多数场合下将消息设为持久化是很重要的。

因而一个完全的持久化应当如下：

1. 将交换机设成 durable。
2. 将队列设成 durable。
3. 将消息设成 durable

2.3 核心概念：绑定

所谓绑定就是将一个特定的 Exchange 和一个特定的 Queue 绑定起来，绑定关键字成为 BindingKey，

程序中语句声明方法为：

```
channel.queueBind("Exchange", "Queue", "BindingKey");
```

Exchange 和 Queue 的绑定可以是多对多的关系，每个发送给 Exchange 的消息都有一个叫做 RoutingKey 的关键字，Exchange 要想将该消息转发给特定队列，该队列与交换器的 BindingKey 必须与消息的 RoutingKey 相匹配才行。

为了进一步了解绑定过程，必须先了解 Exchange 的常见三种类型：

2.2.1. 直接式交换器类型（direct）

直接式交换器类型提供了这样的消息路由机制：通过精确匹配消息的路由关键字，将消息路由到零个或者多个队列中，绑定关键字用来将队列和交换器绑定到一起。这让我们可以构建经典的点对点队列消息传输模型，不过和任何已定义的交换器类型一样，当消息的路由关键字与多个绑定关键字匹配时，消息可能会被发送到多个队列中。

直接式交换器的工作方式如下：

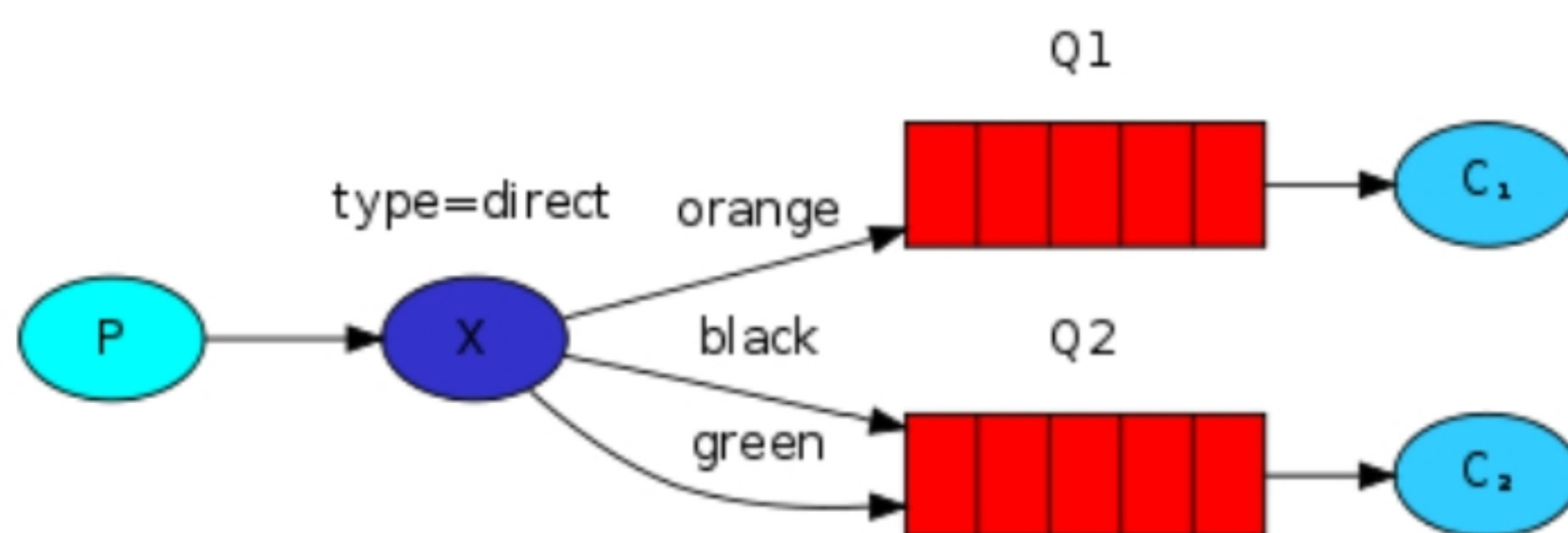


图 2 direct 类型 Exchange 通信原理

从图中可以看出，当某个消息到达交换器 X 时，如果他的 RoutingKey 是 orange,则他将被交付给队列 Q1，如果他的 RoutingKey 是 black 或者是 green,则他将被交付给队列 Q2.

此外，**direct** 模式下还可以实现多路绑定，即一个 **Exchange** 和多个 **Queue** 绑定时具有同样的 **BindingKey**，如图示：

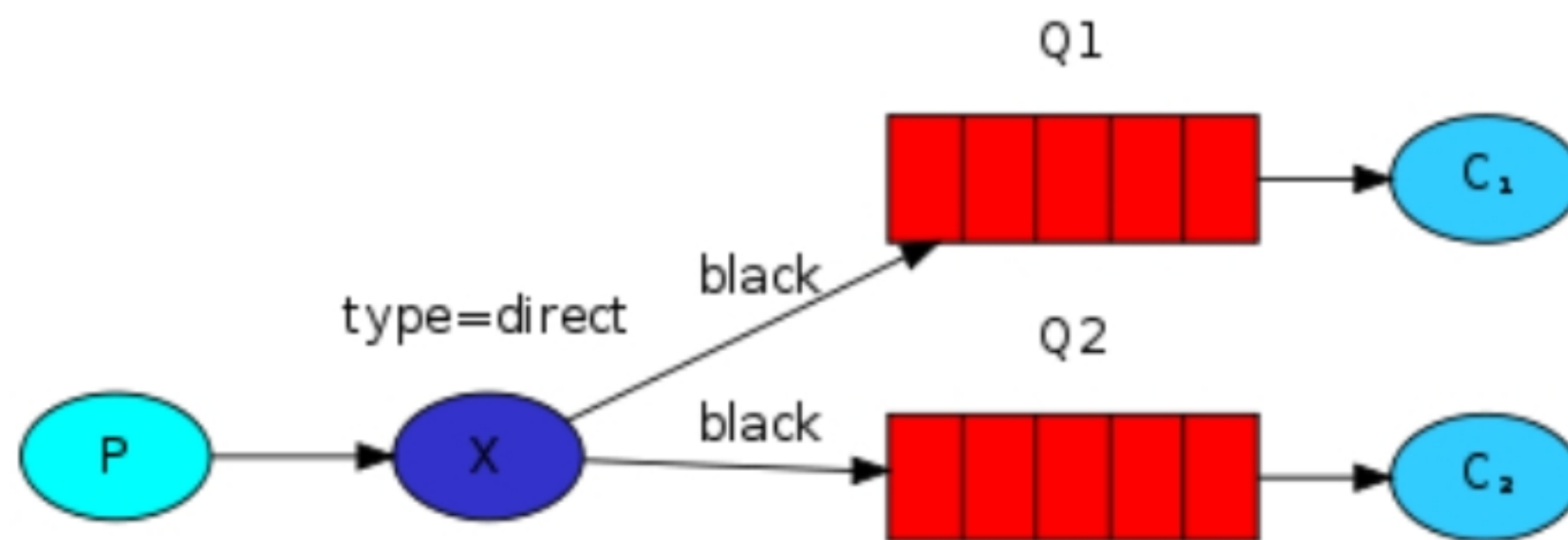


图 3 实现多路绑定 Exchange（direct 类型）

图中，当一个送到交换器 X 的消息的 **RoutingKey** 是 **black** 时，该消息将被同时送往队列 Q1 和队列 Q2。

2.2.2. 广播式交换器类型（fanout）

广播式交换器类型提供了这样的路由机制：不论消息的路由关键字是什么，这条消息都会被路由到所有与该交换器绑定的队列中。

广播式交换器类型的工作方式如下：

不使用任何参数将消息队列与交换器绑定在一起。发布者（直接式交换器类型描述中的 **producer** 变成了 **publisher**，已经隐含了二种交换器类型的区别）向交换器发送一条消息。消息被无条件的传递到所有和这个交换器绑定的消息队列中。

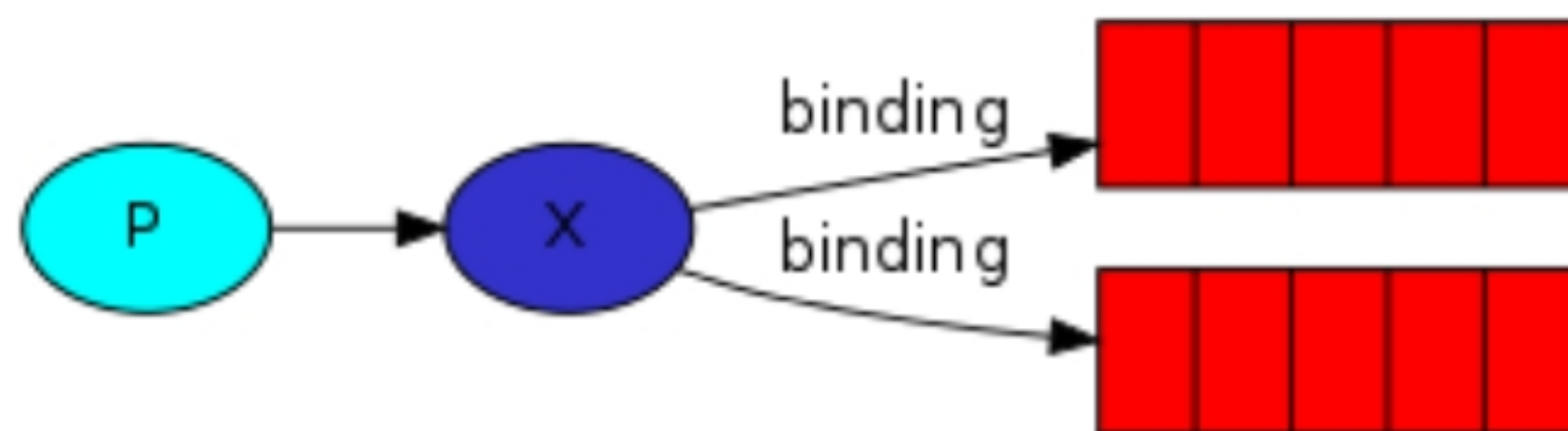


图 4 fanout 类型 Exchange 通信原理

在图中，送达交换器 X 的所有消息，将被送到所有和交换器绑定的 **Queue** 上。

2.2.3. 主题式交换器类型（topic）

主题式交换器类型提供了这样的路由机制：通过消息的路由关键字和绑定关键字的模式匹配，将消息路由到被绑定的队列中。这种路由器类型可以被用来支

持经典的发布/订阅消息传输模型——使用主题名字空间作为消息寻址模式，将消息传递给那些部分或者全部匹配主题模式的多个消费者。

主题交换器类型的工作方式如下：

绑定关键字用零个或多个标记构成，每一个标记之间用“.”字符分隔。绑定关键字必须用这种形式明确说明，并支持通配符：“*”匹配一个词组，“#”零个或多个词组。

因此绑定关键字“*.stock.#”匹配路由关键字“usd.stock”和“eur.stock.db”，但是不匹配“stock.nasdaq”。

这种交换器类型是可选的。

服务器应该（不是必须）实现主题式交换器类型，在这种情况下，服务器必须事先在每一个虚拟主机中定义至少一个主题式交换器：名为“topic”。

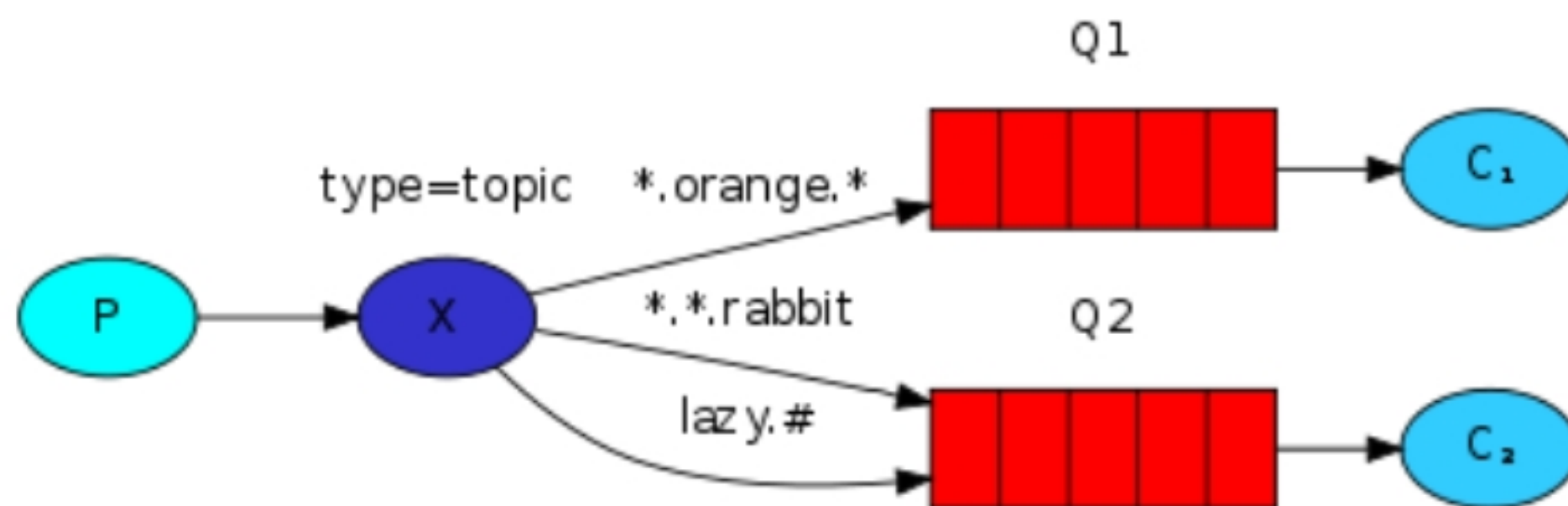


图 4 topic 类型 Exchange 通信原理

图中，*可匹配任意一个单词，#可匹配任意多个（包括 0 个）单词，送到交换器 X 的消息，匹配了特定的 RoutingKey 时，就能送达特定的队列。

2.4 通信过程

图 1 中，左边的客户向右边的客户发送消息：

1. 在 RabbitMQ 中申明 Connection
2. 在 RabbitMQ 中申明 Channel
3. 左边的客户获取 Connection
4. 左边的客户获取 Channel
5. 定义 Exchange, Queue
6. 使用一个 RoutingKey 将 Queue Binding 到一个 Exchange 上
7. 通过指定一个 Exchange 和一个 RoutingKey 来将消息发送到对应的 Queue 上
8. 接收方在接收时也是获取 connection，接着获取 channel，然后指定一个

Queue 直接到它关心的 Queue 上取消息，它对 Exchange，RoutingKey 及如何 binding 都不关心，到对应的 Queue 上去取消息就可以了。

三. RabbitMQ 应用测试

3.1 平台

发送端：linux 环境

服务器：linux 环境

接收端：Windows 环境

3.2 开发工具及语言

Eclipse、JAVA

3.3 实验原理

发送端定义 5 个线程，每个线程连续不断地发送数据，数据大小为 6KB，RabbitMQ-Server 中定义 5 个 Exchange 和 5 个 Queue，Exchange 全部采用 direct 类型，接收端 5 个接收方分别从 5 个队列中取数据，并行写入数据库，同时写入 JFrame，以动态显示所收到的内容，接收完毕后，查询数据库，可获知每条消息的内容、发送时间、接收时间、耗时。

原理图如下所示：

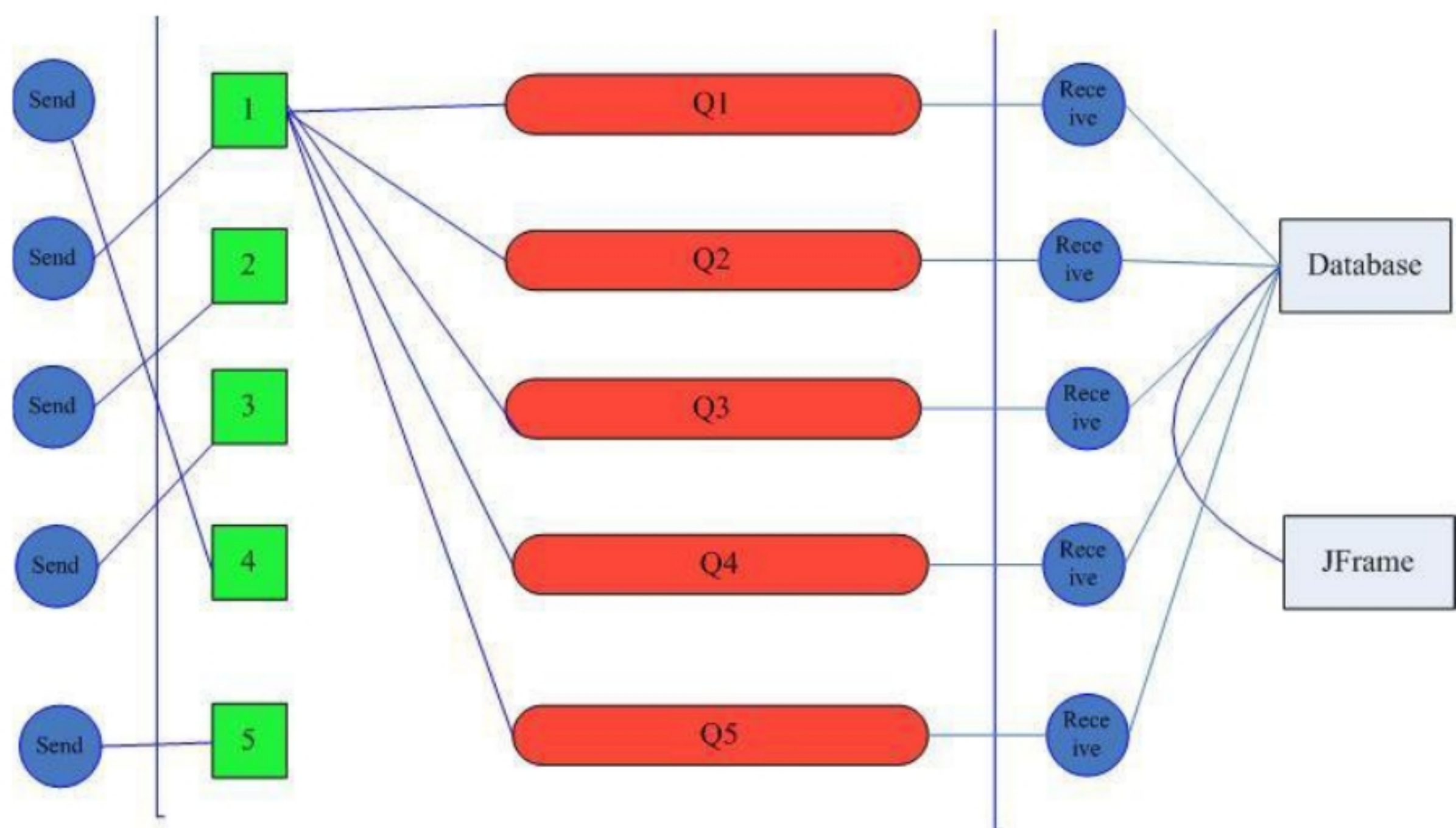
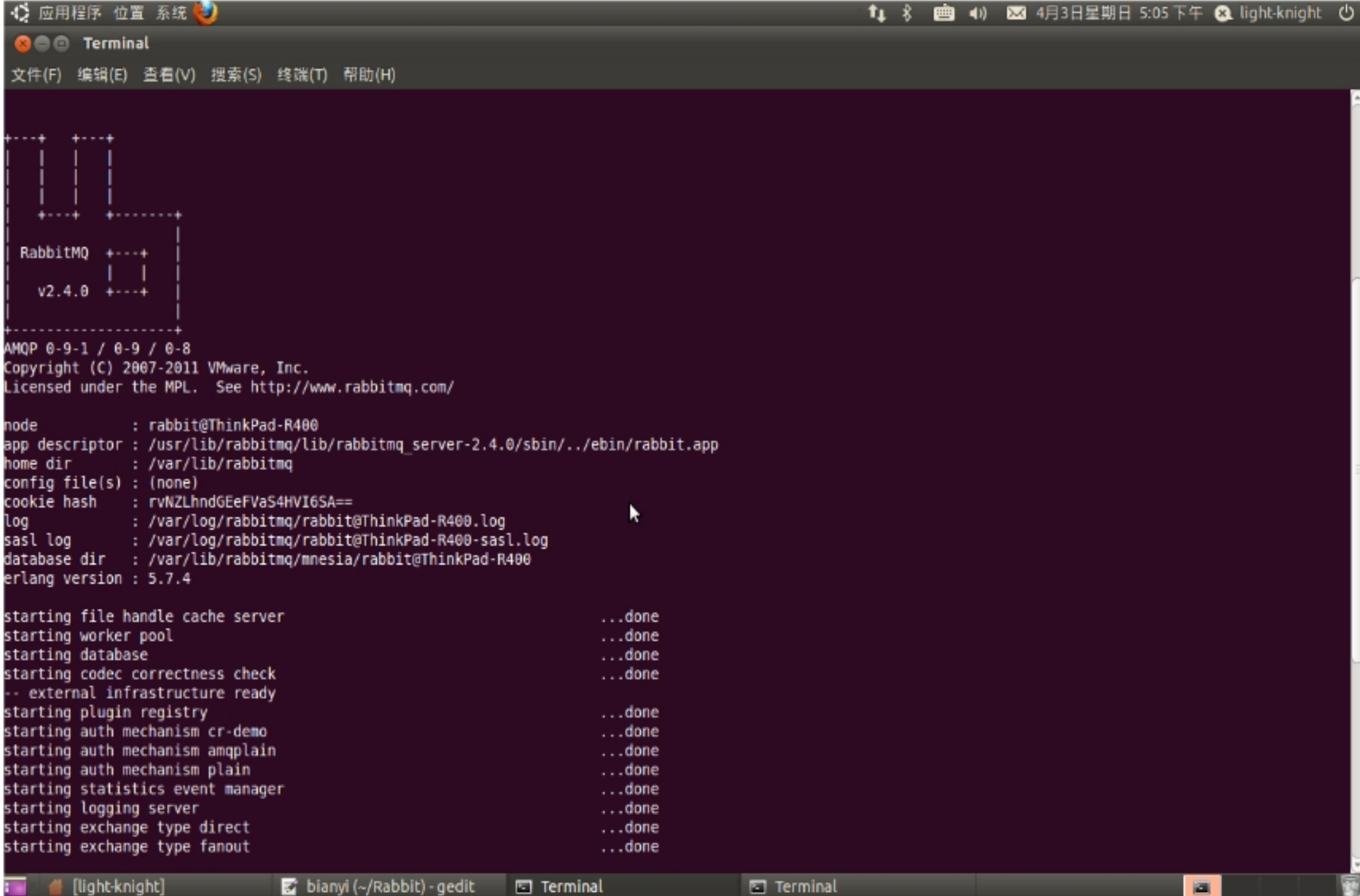


图 5 实验原理图

3.4 测试过程

3.4.1 启动 RabbitMQ-Server

A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help) and a system bar at the top showing '应用程序 位置 系统' and '4月3日星期日 5:05 下午 light-knight'. The terminal output shows the RabbitMQ v2.4.0 startup process. It begins with a dashed box containing the RabbitMQ logo and version 'v2.4.0'. Below this, it shows the AMQP version '0-9-1 / 0-9 / 0-8', copyright information for VMware, Inc., and the license URL. Then, it lists configuration details for the 'rabbit@ThinkPad-R400' node, including the app descriptor, home directory, config file(s), cookie hash, log files, sasl log, database directory, and Erlang version (5.7.4). Finally, it shows a series of startup messages for various components, each followed by '...done': starting file handle cache server, starting worker pool, starting database, starting codec correctness check, external infrastructure ready, starting plugin registry, starting auth mechanism cr-demo, starting auth mechanism amqpplain, starting auth mechanism plain, starting statistics event manager, starting logging server, starting exchange type direct, and starting exchange type fanout.

```
应用程序 位置 系统 4月3日星期日 5:05 下午 light-knight
Terminal
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

+---+ +---+
|   | |   |
+---+ +---+
RabbitMQ +---+
v2.4.0 +---+
+---+ +---+

AMQP 0-9-1 / 0-9 / 0-8
Copyright (C) 2007-2011 VMware, Inc.
Licensed under the MPL. See http://www.rabbitmq.com/

node       : rabbit@ThinkPad-R400
app descriptor : /usr/lib/rabbitmq/lib/rabbitmq_server-2.4.0/sbin/../ebin/rabbit.app
home dir    : /var/lib/rabbitmq
config file(s) : (none)
cookie hash : rvNZLhndGEeFVaS4HVI6SA==
log         : /var/log/rabbitmq/rabbit@ThinkPad-R400.log
sasl log    : /var/log/rabbitmq/rabbit@ThinkPad-R400-sasl.log
database dir : /var/lib/rabbitmq/mnesia/rabbit@ThinkPad-R400
erlang version : 5.7.4

starting file handle cache server      ...done
starting worker pool                  ...done
starting database                     ...done
starting codec correctness check       ...done
-- external infrastructure ready
starting plugin registry               ...done
starting auth mechanism cr-demo        ...done
starting auth mechanism amqpplain      ...done
starting auth mechanism plain          ...done
starting statistics event manager      ...done
starting logging server                ...done
starting exchange type direct          ...done
starting exchange type fanout          ...done
```

图 6 启动 rabbitmq-server

看到一个类似兔子头的标识，标识启动 RabbitMQ 服务成功

3.4.2 传送数据

五个线程同时发送数据，数据大小为 6KB。传送过程：

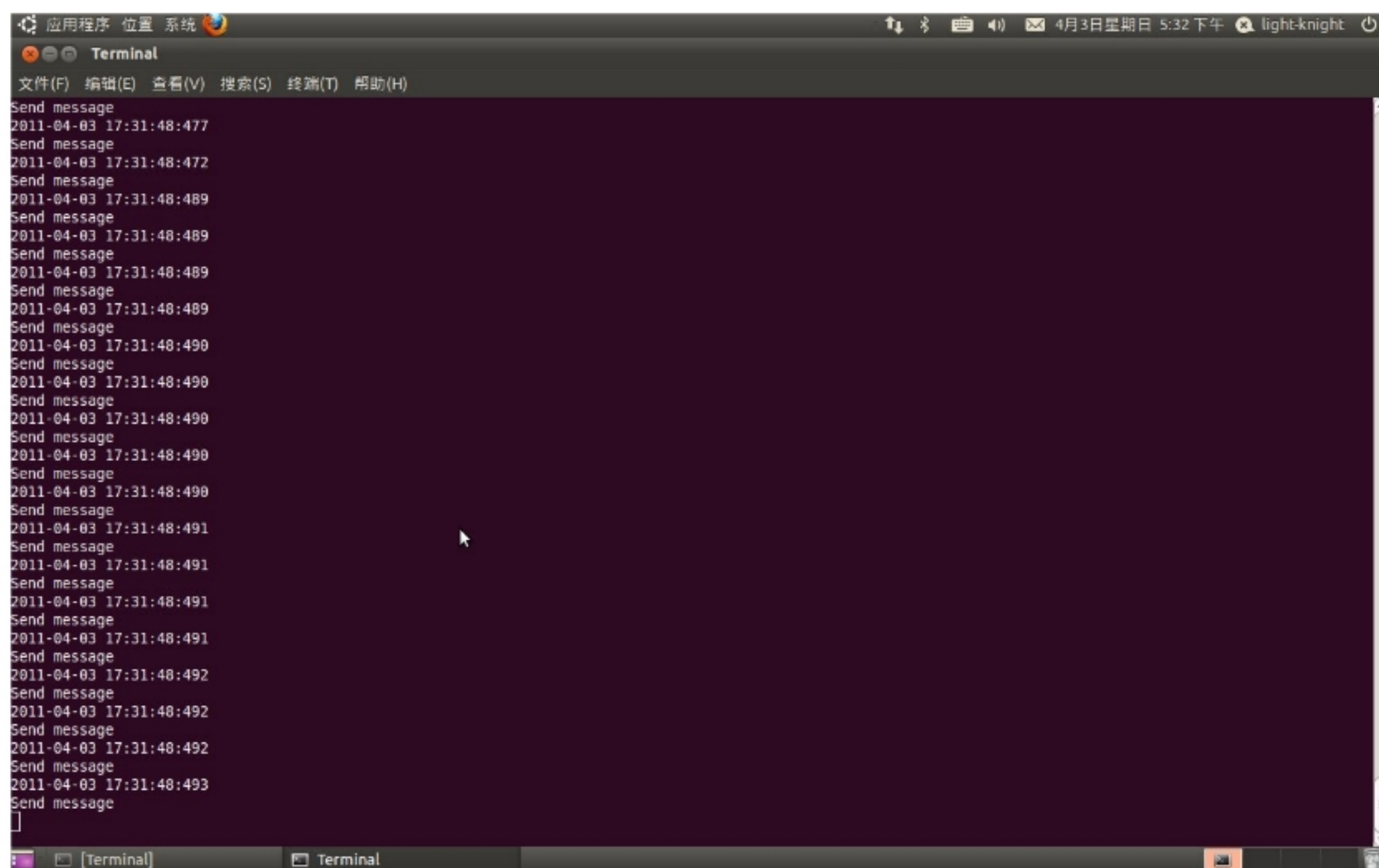


图 7 传送数据过程

3.4.3 接收数据

接收数据在 windows 环境下，用 JFrame 察看接受的数据：

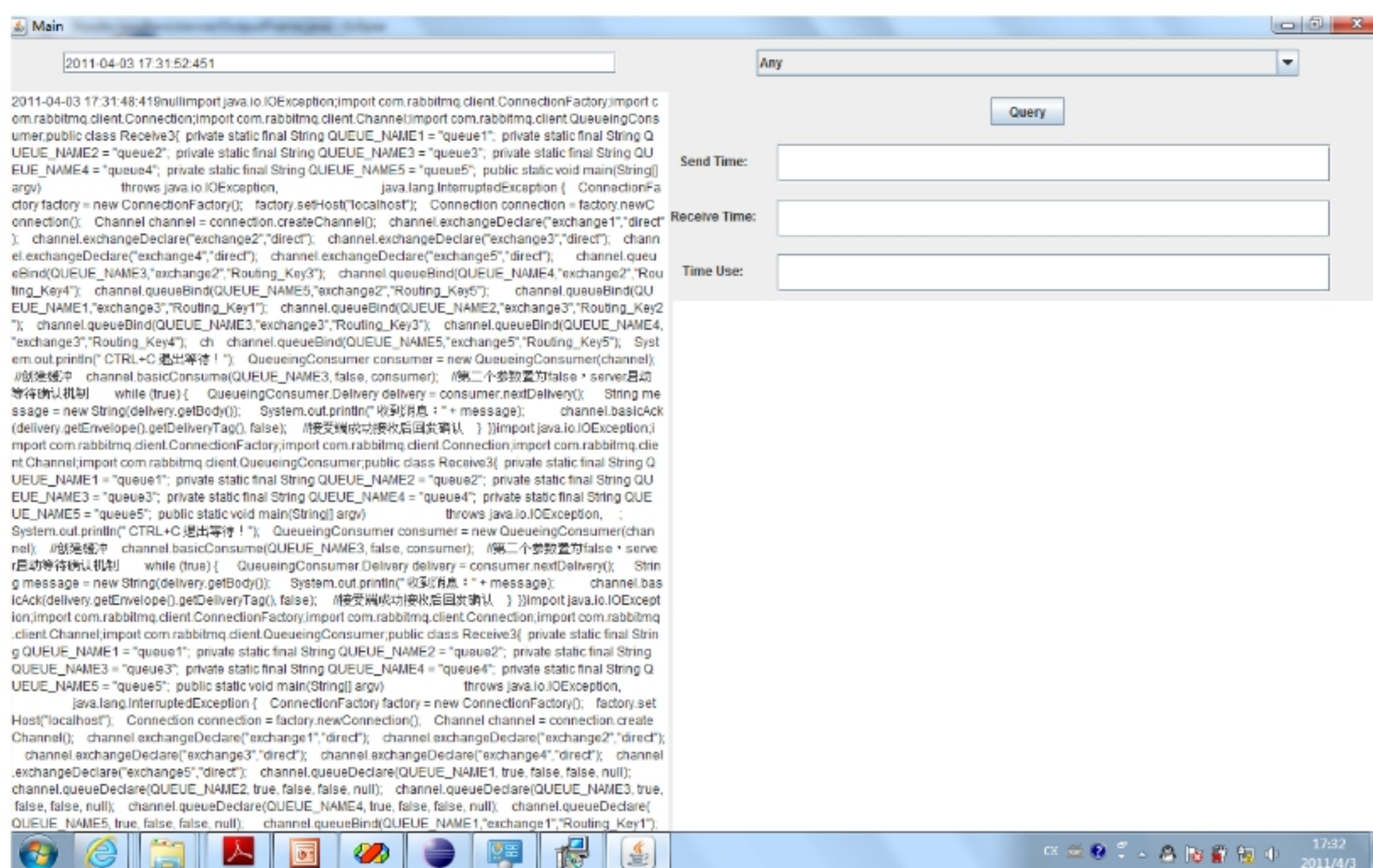


图 8 接收数据过程

```
2011-04-03 17:31:52:451
2011-04-03 17:31:48:419nullimport java.io.IOException;import com
om.rabbitmq.client.Connection;import com.rabbitmq.client.Channe
umer;public class Receive3{ private static final String QUEUE_NAM
QUEUE_NAME2 = "queue2"; private static final String QUEUE_NAME
QUEUE_NAME4 = "queue4"; private static final String QUEUE_NAME5
argv) throws java.io.IOException, java
ctory factory = new ConnectionFactory(); factory.setHost("localhos
onnection(); Channel channel = connection.createChannel(); ch
); channel.exchangeDeclare("exchange2","direct"); channel.exch
el.exchangeDeclare("exchange4","direct"); channel.exchangeDec
eBind(QUEUE_NAME3,"exchange2","Routing_Key3"); channel.qu
419:419); channel.exchangeBind(QUEUE_NAME5,"exchange4","R
```

图 9 显示接收时间和发送时间

从图示可以看出，报文段的发送时间为 2011-04-03 17:31:48:419，接收时间为 2011-04-03 17:31:52:451 时间差为 4 秒，发送时间后面即为所传送的数据内容！