目录

[RabbitMQ课程计划 2](#_Toc23739)

[1. 目前市场上各MQ的比较 3](#_Toc25735)

[1.1 基本信息比较 3](#_Toc28813)

[1.2 支持的协议 3](#_Toc3950)

[1.3 不同场景下各MQ的压力测试 4](#_Toc8800)

[1.3.1 Scenario A 4](#_Toc12664)

[1.3.2 Scenario C 4](#_Toc10327)

[1.3.3 Scenario D 5](#_Toc9081)

[1.3.4 Scenario B 5](#_Toc28227)

[1.3.5 结论 6](#_Toc1799)

[2. RabbitMQ原理 6](#_Toc2118)

[2.1 什么是RabbitMQ 6](#_Toc15633)

[2.2 RabbitMQ原理 6](#_Toc25479)

[2.2.1 AMQP 6](#_Toc5717)

[2.2.2 几个重要的概念 8](#_Toc27980)

[2.2.3 Send-Receive流程图 9](#_Toc13428)

[2.2.4 结论 10](#_Toc27714)

[3. RabbitMQ服务器的安装 10](#_Toc2227)

[3.1 下载Erlang包和RabbitMQ服务端 10](#_Toc25793)

[3.1.1 安装Erlang 10](#_Toc1446)

[3.1.2 安装RabbitMQ服务器 11](#_Toc26684)

[3.1.3 启动服务 12](#_Toc858)

[3.1.4 配置RabbitMQ监控端 13](#_Toc29005)

[3.1.5 RabbitMQ Management 14](#_Toc4544)

[4. Hello World 14](#_Toc14580)

[4.1 下载所需jar包，rabbit-client.jar 14](#_Toc4957)

[4.2 hello world 14](#_Toc5895)

[4.3 编写发送消息客户端/消费客户端 15](#_Toc2699)

[4.4 运行结果 16](#_Toc20196)

[5. RabbitMQ深入 17](#_Toc5595)

[5.1 工作队列（一） 17](#_Toc29406)

[5.1.1 概念及使用 17](#_Toc15437)

[5.1.2 使用 18](#_Toc7174)

[5.1.3 完整实例 18](#_Toc28793)

[5.1.4 结论 18](#_Toc24889)

[5.2 工作队列（二） 19](#_Toc10723)

[5.2.1 消息应答 19](#_Toc3991)

[5.2.2 Work修改 20](#_Toc11791)

[5.3 工作队列（三） 21](#_Toc30682)

[5.3.1 消息持久化 21](#_Toc12048)

[5.3.2 结论 22](#_Toc32539)

[5.4 工作队列（四） 22](#_Toc21410)

[5.4.1 公平转发 22](#_Toc22280)

[5.5 发布/订阅 23](#_Toc7445)

[5.5.1 交换机（Exchanges） 24](#_Toc8353)

[5.5.2 匿名转发器（nameless exchange） 24](#_Toc403)

[5.5.3 临时队列（Temporary queue） 25](#_Toc5613)

[5.5.4 绑定（Binding） 26](#_Toc1410)

[5.5.5 完整实例 26](#_Toc31304)

[5.6 路由选择（Routing） 26](#_Toc13589)

[5.6.1 绑定（Bindings） 26](#_Toc20029)

[5.6.2 直接转发（Direct exchange） 27](#_Toc10059)

[5.6.3 多重绑定（multiple bindings） 28](#_Toc27394)

[5.6.4 发送日志（EmittingLogs） 28](#_Toc15844)

[5.6.5 订阅 29](#_Toc2840)

[5.6.6 完整实例 29](#_Toc12830)

[5.7 主题（Topic） 30](#_Toc12440)

[5.7.1 主题转发（Topic Exchange） 30](#_Toc15413)

[5.7.2 图解 31](#_Toc7986)

[5.7.3 完整实例 32](#_Toc21492)

[6. RabbitMQ企业应用案例 32](#_Toc3823)

[6.1 案例 32](#_Toc15080)

[6.2 总结 36](#_Toc9777)

[7. 参考资料 36](#_Toc17639)

RabbitMQ课程计划

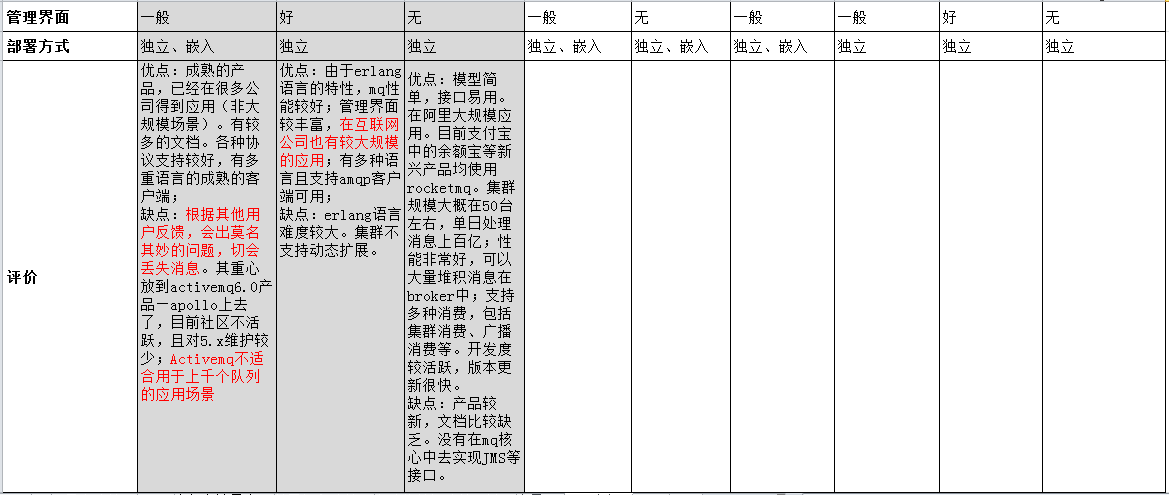
**作者：阮文**

**时间：2015-07-14**

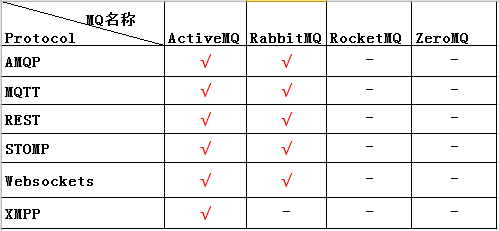
# 目前市场上各MQ的比较

## 基本信息比较





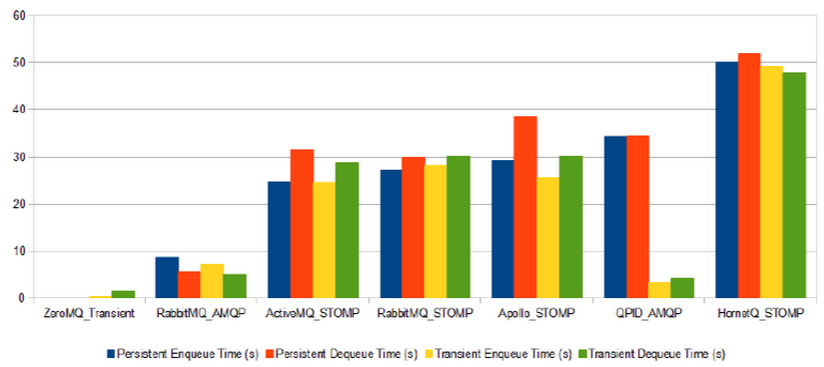
## 支持的协议



## 不同场景下各MQ的压力测试

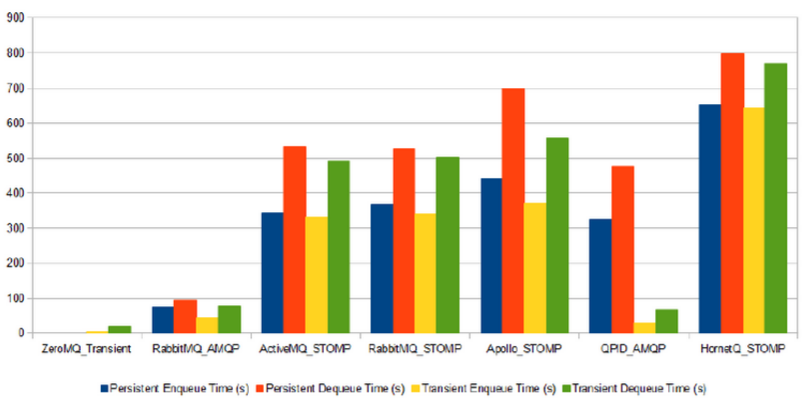
### Scenario A

Enqueuing 20,000 messages of 1024 bytes each, then dequeuing them afterwards.



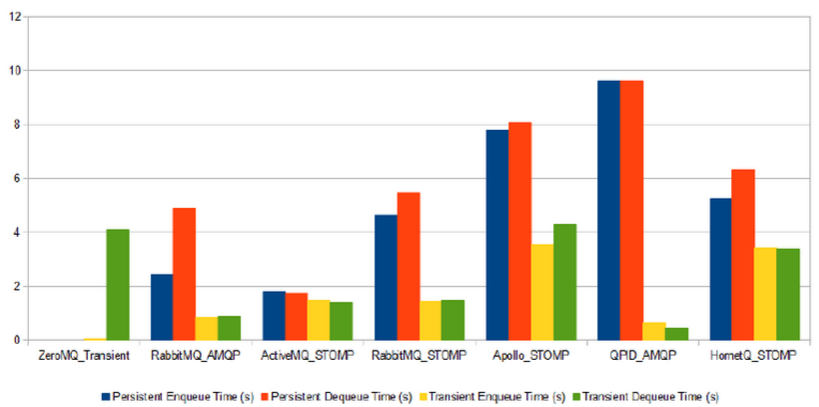
### Scenario C

Enqueuing and dequeuing simultaneously 20,000 messages of1024 bytes each.



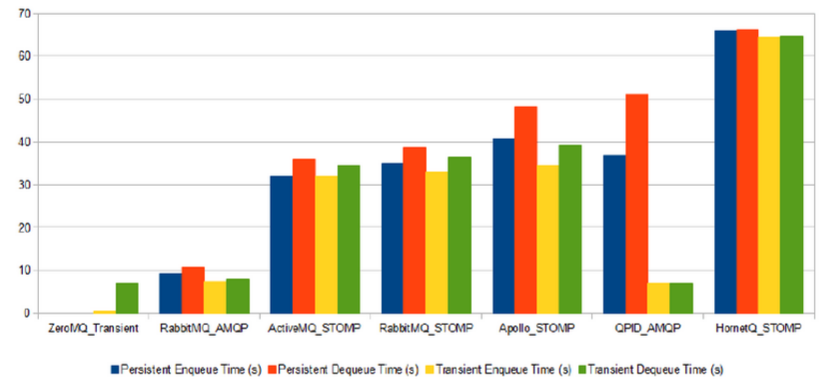
### Scenario D

Enqueuing and dequeuing simultaneously 200 messages of32768 bytes each



### Scenario B

Enqueuing and dequeuing simultaneously 20,000 messages of1024 bytes each.



### 结论

在传统MQ的压力测试的比较过程中，RabbitMQ的性能表现最好也是最稳定的；如果使用到分布式存储或者其他关于Hadoop方便的计算框架，我们推荐使用kafka。

# RabbitMQ原理

## 什么是RabbitMQ

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。应用程序通过读写出入队列的消息（针对应用程序的数据）来通信，而无需专用连接来链接它们。消息传递指的是程序之间通过在消息中发送数据进行通信，而不是通过直接调用彼此来通信，直接调用通常是用于诸如远程过程调用的技术。排队指的是应用程序通过队列来通信。队列的使用除去了接收和发送应用程序同时执行的要求。

RabbitMQ是一个在AMQP基础上完整的，可复用的企业消息系统。

## RabbitMQ原理

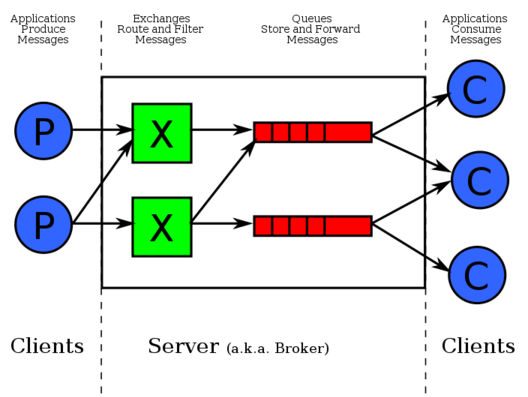
### AMQP

AMQP（Advanced Message Queue Protocol），高级消息队列协议

AMQP当中有四个非常重要的概念：虚拟主机（virtual host），交换机（exchange），队列（queue）和绑定（binding）。一个虚拟主机持有一组交换机、队列和绑定。为什么需要那么多个虚拟主机呢？很简单，RabbitMQ中，用户只能砸虚拟主机的粒度进行权限控制。例如，如果需要禁止A组访问B组的交换机/队列/绑定，必须为A和B分别穿件一个虚拟主机。每一个RabbitMQ都有一个默认的虚拟主机。

Producer要生产消息必须要创建一个Exchange，Exchange用于转发消息，但是它不会做存储，如果没有Queue 绑定到Exchange的话，它会直接丢弃掉Producer发送过来的消息，当然如果消息总是发送过去就直接别丢弃那就没有什么意义了，一个Consumer想要接收消息的话，当然就要创建一个Queue，并把这个Queue绑定到指定的Exchange上，然后Exchange会把消息转发到Queue那里，Queue会负责存储消息，Consumer可以通过Pop或者Subscribe之后被动回调的方式从Queue中获取消息。

原理图如下：



### 几个重要的概念

Broker：简单来说就是消息队列服务器实体

Vhost：虚拟主机，一个broker里可以开设多个vhost，用作不同用户的权限分离。

Exchange：消息交换机，它指定消息按什么规则，路由到哪个队列。

Queue：消息队列载体，每个消息都会被投入到一个或多个队列中。

RoutingKey：路由关键字，Exchange根据这个关键字进行消息的投递，即该消息投递到哪个Queue中。

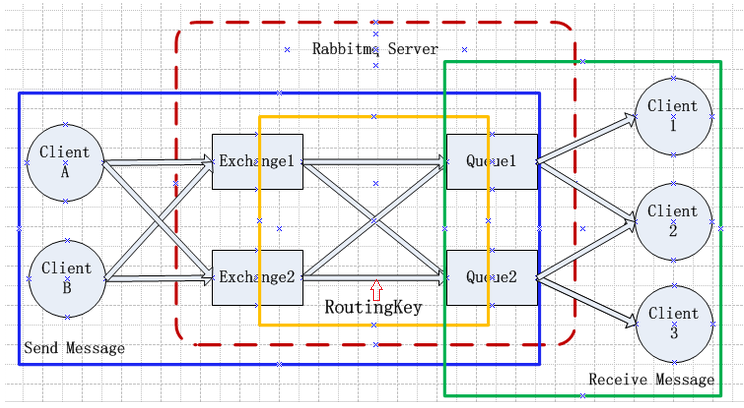
Binding：绑定，它的作用就是把Exchange和Queue按照路由规则绑定起来。

Channel：消息通道，在客户端的每个连接里，可以建立多以channel，每个channel一个会话任务。

Producer：消息生产者，就是投递消息的程序。

Consumer：消息消费者，就是接受消息的程序。

### Send-Receive流程图

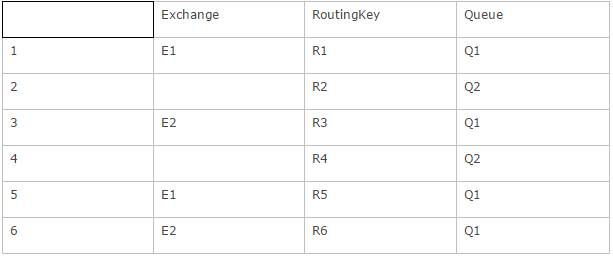


左边的Client向右边的Client发送消息，流程如下：

* 获取Connection，即连接到RabbitMQ主机
* 获取Channel
* 定义Exchange、Queue并设置相关属性
* 使用一个RoutingKey将Queue Binding到一个Exchange上
* 通过指定一个Exchange和一个RoutingKey来将消息发送到对应的Queue上。
* 接收方在接收时也是获取connection，接着获取channel，然后指定一个Queue直接关联到它关心的Queue上取消息，它对Exchange、RoutingKey及如何binding都不关心，到对应的Queue上取信息就OK了。

### 结论

一个Client发送消息，哪些Client可以接收到消息，其核心就在于Exchange、RoutingKey、Queue的关系上。



可以这样理解，RoutingKey就像是个中间表，将两个表的数据进行多对多关联，只不过对于相同的Exchange和Queue，可以使用不同的RoutingKey重复关联多次。

# RabbitMQ服务器的安装

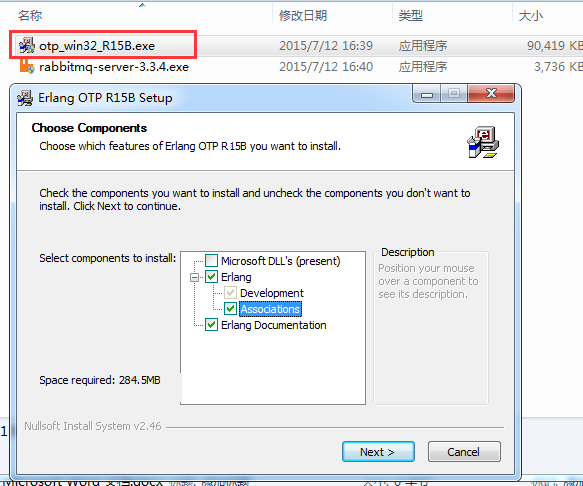
## 下载Erlang包和RabbitMQ服务端

Erlang下载地址：<http://www.erlang.org/download/>，选择对应的版本

RabbitMQ下载地址：<http://www.rabbitmq.com/releases/rabbitmq-server/>，选择对应版本

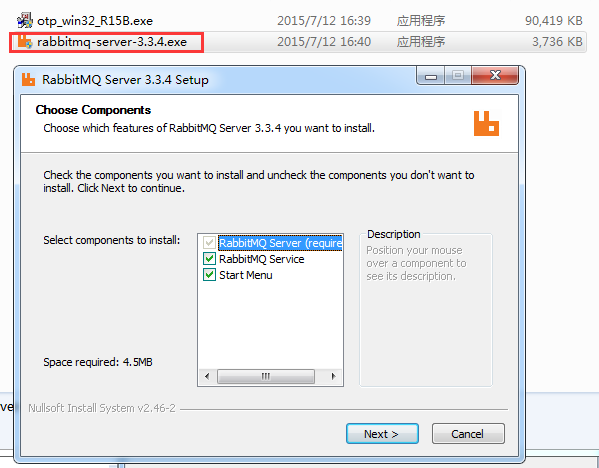
### 安装Erlang

由于RabbitMQ由Erlang语言开发，因此先需要安装其运行环境。



### 安装RabbitMQ服务器

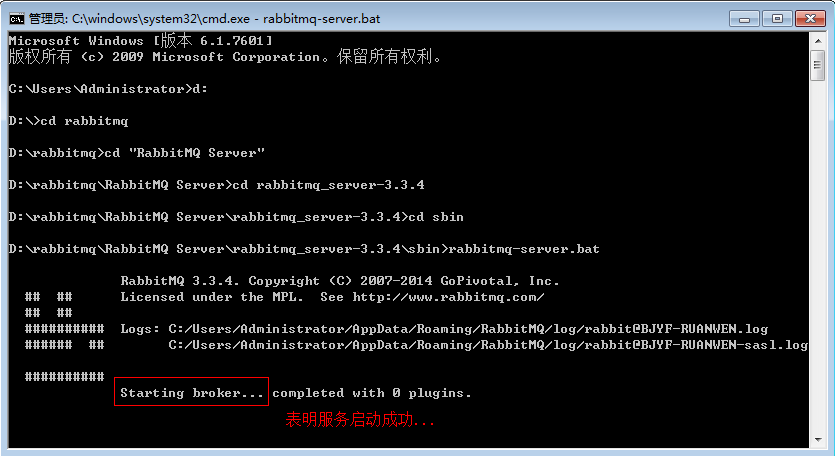
双击rabbitmq-server-3.3.4.exe，进入安装界面，然后next…



### 启动服务

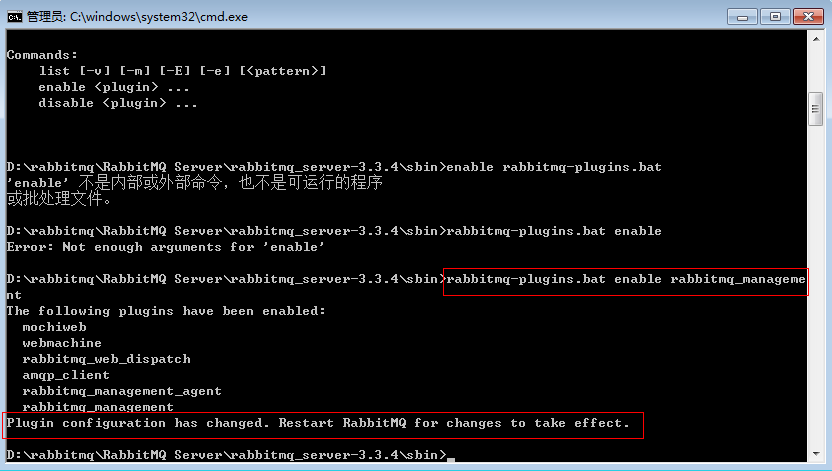
安装完成后，该服务会自动启动，如果没有自动启动，可以进入服务器安装目录的sbin目录下，执行该rabbitmq-server.bat批处理文件。若出现如下图所示的红色标记部分，则表示服务已启动。

需要注意的是，如果是通过命令启动的服务，该dos命令行窗口不能关闭，如果关闭窗口相当与将服务停止，除非将其注册为服务（没研究）。



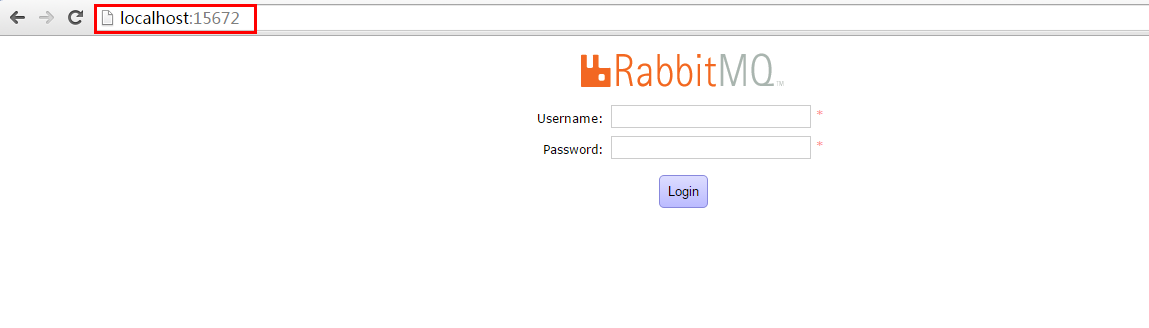
### 配置RabbitMQ监控端

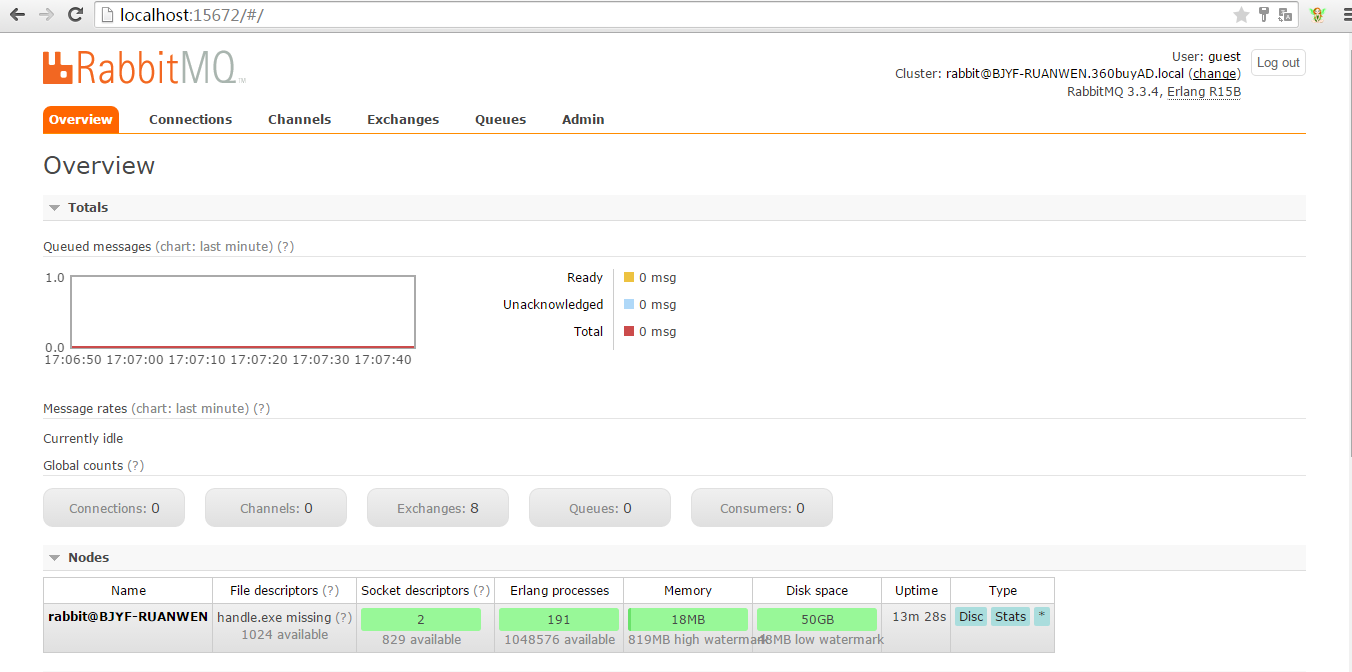
进入sbin目录，执行以下命令：rabbitmq-plugins.bat enable rabbitmq\_management，执行完成后**需重启才能访问该客户端**。默认情况下用户名/密码为：guest/guest。我们可以在sbin目录下通过命令来创建用户等操作。



### RabbitMQ Management

访问地址：<http://localhost:15672/> 用户名/密码：默认guest/guest。





# Hello World

## 下载所需jar包，rabbit-client.jar

下载地址：<http://www.rabbitmq.com/releases/rabbitmq-java-client>，选择对应的版本，将其解压后复制该jar包导入项目中。

## hello world

一个producer发送消息，一个接收者接收消息，并在控制台打印出来。如下图：



## 编写发送消息客户端/消费客户端

发送程序：

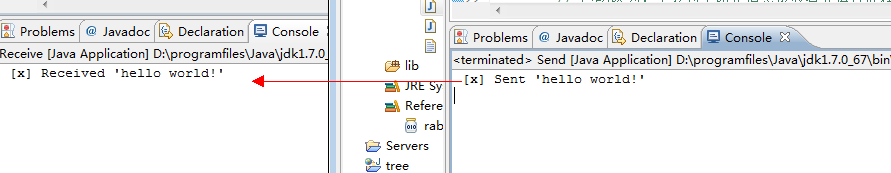


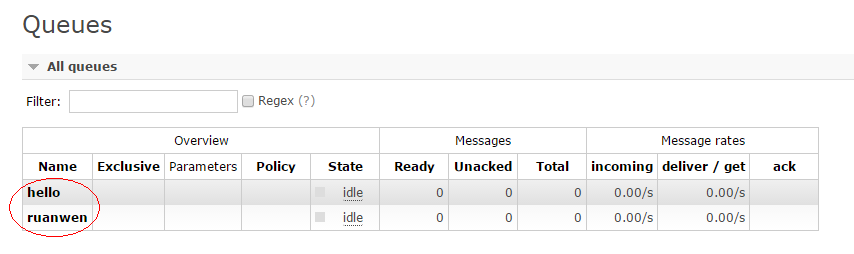
接收程序：



## 运行结果

声明的队列名称必须一致，这样获取的时候才能在指定队列名称中获取消息。





信息发送完成后，我们可通过监控系统去查看信息



# RabbitMQ深入

## 工作队列（一）

### 概念及使用

在hello world的demo中，我们通过命名队列来发送和接收消息，这里我们将创建一个工作队列用来在工作者间分发耗时任务

工作队列的主要任务是：避免立刻执行资源密集型任务，然后必须等其完成。相反的，我们进行任务调度：我们把任务封装为消息发送给队列。工作进行在后台运行并不断的从队列中取任务然后执行。当你运行多个工作进程时，任务队列中的任务将会被工作进程共享执行。

这样的概念在web应用中及其有用，当在很短的HTTP请求间需要执行复杂的任务。为了避免等待一些占用大量资源、时间的操作。当我们把任务（Task）当作消息发送到队列中，一个运行在后台的工作者（worker）进程就会取出任务然后处理。当你运行多个工作者（workers），任务就会在它们之间共享。

默认的，RabbitMQ会一个一个的发送信息给下一个消费者(consumer)，而不考虑每个任务的时长等等，且**是一次性分配，并非一个一个分配**。平均的每个消费者将会获得相等数量的消息。这种发送消息得方式叫做——轮询（round-robin）。

### 使用

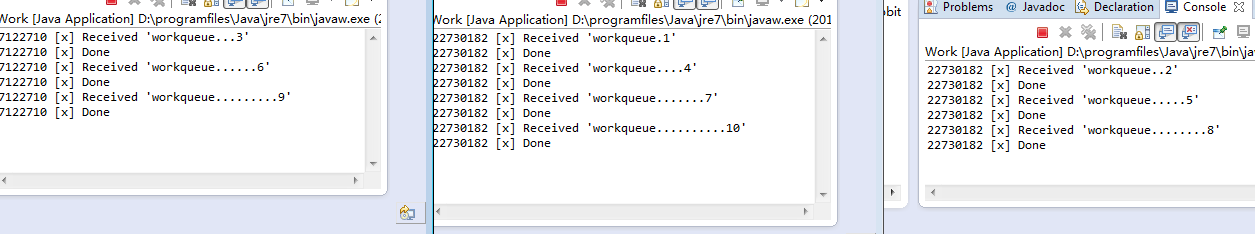
首先开启3个消费端的work，然后在开启发送方并发送信息，这时候我们可以看到每个消费者都会接收到同等数量的消息。这样就分担了其他消费者的任务压力，从而也提高了效率。

### 完整实例

略，查看IDE：NewTask.java（生产）、Work.java（消费）

### 结论

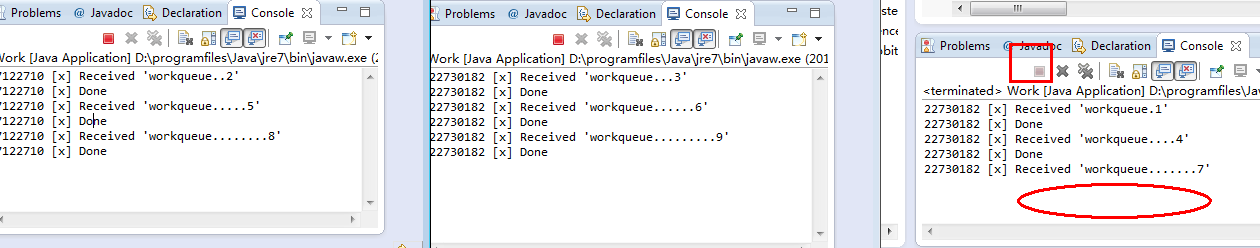
使用任务队列的好处就是能够让消费者并行工作，如果积压的工作很多，我们就只要增加更多的工作者就可以解决问题，使系统的伸缩性更加容易。



## 工作队列（二）

### 消息应答

执行一个任务需要花费几秒。你可能担心在一个工作者正在执行的任务的过程中发生中断。【6.1】的代码，一旦RabbitMQ将信息交付给消费者，立马会从内存中移除这个消息。在这种情况下，如果杀死正在执行的某个工作者，我们的处理的消息会丢失，而且分配给该工作者的还未执行的信息也会丢失。在【6.1】中，如果一个正在执行的工作者（work3）突然发生中断，这信息如下。



可以看到，【信息7】未执行完，【信息10】直接丢失。

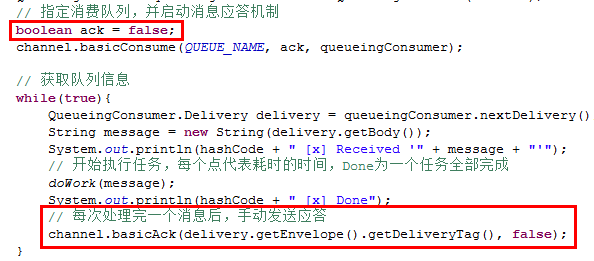
为了保证信息的永远不丢失，RabbitMQ支持应答消息（message acknowledgments）。消费者发送应答给RabbitMQ，告知它信息已经被接收和处理，然后RabbitMQ可以自由的进行信息删除。

如果消费者被杀死而没有发送应答，RabbitMQ会认为信息没有完全的被处理，然后将会重新转发给别的消费者，通过这种方式，你可以确认信息不会丢失，即使消费者偶尔被杀死。

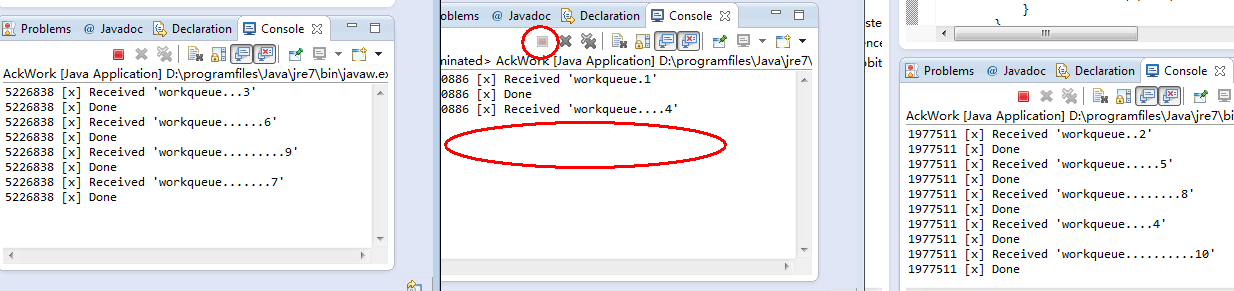
这种机制是没有超时这中概念，RabbitMQ只有消费者连接断开后才重新转发该消息的，如果消费者处理一个比较长的消息是允许的。消息应答默认是打开的。

### Work修改

将消费者代码修改为：



结果如下：



从结果中我们可以看出，我们将work2执行了【消息1】和为执行完了【消息4】与服务断开后，即【4】、【7】、【10】消息没有执行完，我们开启消息应答后，这三个消息被剩余两个work执行了。

## 工作队列（三）

### 消息持久化

前面已经讲到消费者被kill掉，也能保证消息不丢失。但是如果此时的RabbitMQ服务被停止或宕机，那么消息仍然会丢失。

当RabbitMQ服务退出或异常退出时，将会丢失所有的队列和信息，除非你告诉它不要丢失。我们需要确保两件事来确认信息不丢失：将所有的队列和消息设置持久化标识。消息持久化（Message durability）的就是将消息保存到了内存或硬盘中。

第一，我们需要确认RabbitMQ永远不会丢失我们的队列，为了这样，我们需要将其声明为持久化。

boolean durable = true;

channel.queueDeclare(QUEUE\_NAME,durable,false,false,null);

**注：RabbitMQ不允许使用不同的参数重新定义同一个队列，所以已存在的队列我们无法修改其属性。**

第二，我们需要标识我们的消息为持久化的。通过设置MessageProperties(implements BasicProperties)值为PERSISTENT\_TEXT\_PLAIN.

channel.basicPublish("","task\_queue",MessageProperties.PERSISTENT\_TEXT\_PLAIN,message.getBytes());

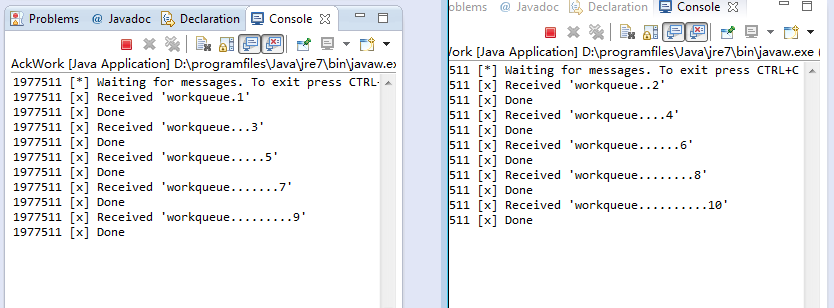
### 结论

如果设置了队列和消息持久化的标识，在重启RabbitMQ服务后，仍然能从队列中获取该信息。

## 工作队列（四）

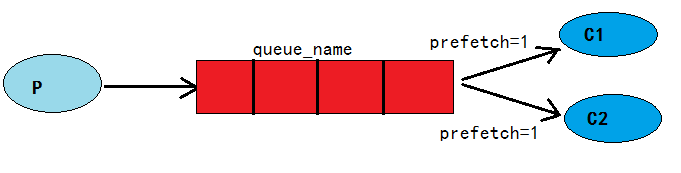
### 公平转发

之前我们讲过消息的转发机制（Round-robin），也许你会发现，这个机制并不是我们想要的。例如，这样一个情况，对两个消费者，有一系列任务，奇数的任务比较耗时，而偶数的任务却很轻松，这样就会造成一个消费者一直繁忙，另一个消费者却很快执行完任务后等待。

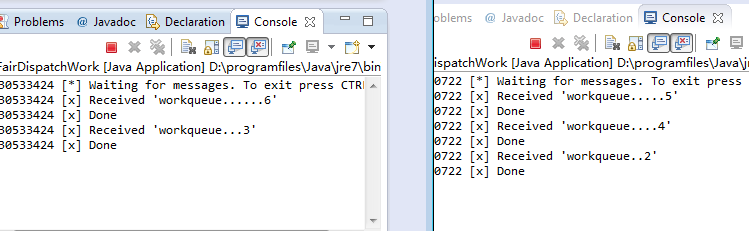


造成这样原因是因为RabbitMQ仅仅是当消息到达队列进行转发消息。仅仅盲目转发所有奇数给一个消费者，偶数给另一个消费者。

为了解决这个问题，我们使用basicQos方法，传递参数prefetchCount=1.这样告诉RabbitMQ不要在同一时间给消费者超过一条信息。换句话说，只有在消费者空闲时发送下一条信息。



使用公平转发策略后，结果如下：



注：如果所有的工作者都处于繁忙状态，队列有可能都被充满。我们可以观察队列的使用情况，然后增加工作者或者使用其他策略。

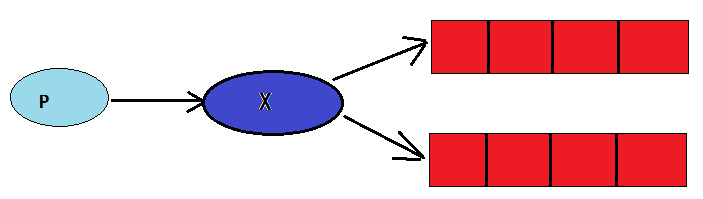
## 发布/订阅

之前我们讲的都是一对一的关系，即一个生产者-一个消费者。这里将做一些改变，就是把一个消息发送给多个消费者，这种模式称之为发布/订阅（类似于观察者模式）。

案例：构建一个日志系统。这个系统包括两个程序，一类程序发动日志，另一类程序接收和处理日志。在日志系统中，每一个运行的接收者程序都会接收到日志，一个接收者接收到的数据我们写到硬盘上，与此同时，另一个接收者把接收到的消息展现在屏幕上。本质上来说，发布的日志消息会转发给所有的接收者。

### 交换机（Exchanges）

RabbitMQ消息模型的核心理念是生产者永远不会直接发送任何消息给队列，一般情况下生产者甚至不知道消息应该发送到哪个队列。生产者只把消息发送给交换机（Exchange）。交换机一边接收从生产者发送来的消息，另一边把信息推送到队列中。交换机必须清楚的知道如何处理它接收到的每条消息。是否追加到一个指定队列？是否应该追加到多个队列？或者是否丢弃？这些规则都是通过交换机的类定义的。



交换机类型：Direct、Topic、Headers、Fanout，这里我们关注fanout，申明交换机类型为fanout，**该类型特别简单，把它所接收的消息广播到它所知道的队列中**。channel.exchangeDeclare(“logs”,”fanout”).不过这正是我们日子系统所需要的。

### 匿名转发器（nameless exchange）

前面的例子中，我们并没有使用到交换机，但是仍然可以发送和接收消息。因为这是我们使用了一个默认的交换机，它的标识是“”，前面的代码，channel.basicPublish(“”,…,….);第一个参数为交换机名称，我们设置了“”；如果存在routingkey（第二个参数），消息由RoutingKey决定发送到哪个队列。即：Exchange-RoutingKey-Queue.指定交换机，channel. basicPublish( "logs","", null, message.getBytes());

### 临时队列（Temporary queue）

前面的例子中，我们都为队列指定了一个特定的名称。为队列命名是非常关键的，我们需要指定消费者为某个队列。当希望在生产者与消费者间共享队列时，为队列命名是非常重要的。不过，对于我们的日志系统而言，我们并不关心队列的名称。我们想要接收到所有消息，而且我们也只对当前正在传递的数据感兴趣。为了满足需求，我们需要做两件事。

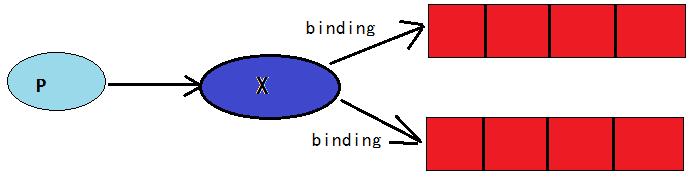
第一，无论什么时间连接到RabbitMQ我们都需要一个新的空队列，我们可以生成随机数来创建队列，或者让服务器给我们提供一个随机的名称。

第二，一旦消费者与RabbitMQ断开，消费者所接收的那个队列应该被自动删除。

我们可以通过channel.queueDeclare()方法，不传递任何参数，来创建一个非持久的、唯一的、自动删除（当消费者与服务断开连接时，该生成的队列会自动删除）的队列且队列名称可通过服务器随机产生。String name = channel.queueDeclare().getQueue().一般情况下这个名称与amq.gen-JzTY20BRgKO-HjmUJj0wLg 类似。

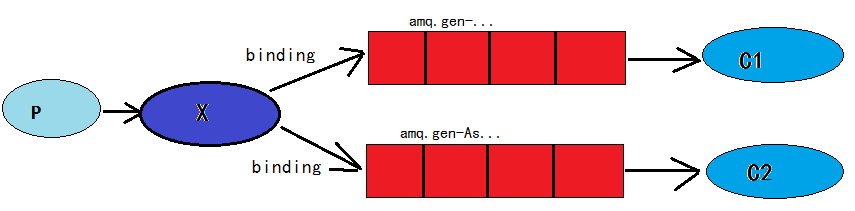
### 绑定（Binding）

我们已经创建了一个类型为fanout的交换机和队列，我们需要通过binding告诉交换机把信息发送给我们的队列，参数1，队列名称；参数2，交换机名。channel.queueBind(queueName,”logs”,””).



### 完整实例

该信息交互的总图如下：



## 路由选择（Routing）

发布/订阅时，消费者订阅了所有消息，这里我们将对系统做出调整，让日志接收者只订阅部分消息。例如，我们可以仅仅将致命的错误写入到日志文件中，然后在控制面板打印其他的类型的日志消息。

### 绑定（Bindings）

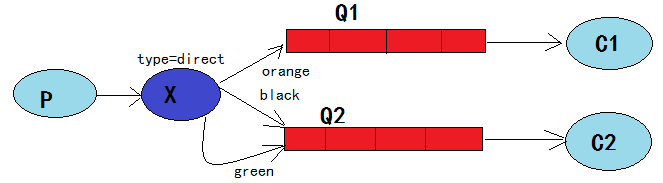
上一节已经使用到了绑定，channel.queueBind(queueName,E\_N,””);绑定表示交换机与队列之间的关系，也可理解为队列对该交换机上的信息感兴趣。绑定可以带一个额外的参数RoutingKey。为了避免与basicPublish方法（发布消息的方法）的参数混淆，我们将其称为绑定键（routing key）。下面展示如何使用绑定键（routing key）来创建一个绑定：channel.queueBind(queueName,EXCHANGE\_NAME,”black”);

绑定键的意义依赖于交换机类型。**对于fanout类型的交换机可以忽略此参数**。

### 直接转发（Direct exchange）

上一节日志系统广播所有的消息给所有的消费者。我们希望可以对其扩展，来允许根据日志的严重性进行过滤日志。例如，我们可能将致命错误的日志写入硬盘，而不把硬盘的空间浪费在警告或者消息类型的日志上。

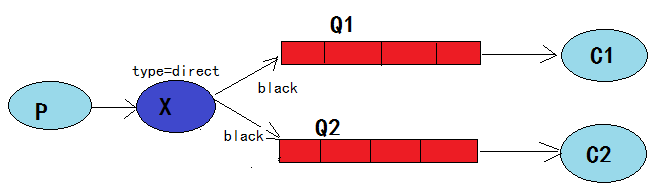
之前我们使用fanout类型的交换机，但是没有给我们带来更多的灵活性，仅仅是笨笨的转发。这里我们将使用direct类型的转发器进行代替。Direct类型的交换机背后的路由算法很简单：消息被推送至绑定键（routing key）和消息发布自带的选择键（routing key）完全匹配的队列中。图：



上图，可以看到direct类型的转发器与两个队列绑定。第一个队列与绑定键orange绑定，第二个队列与转发器间有两个绑定，一个与绑定键black绑定，另一个与绑定键green绑定。

这样，当一个消息附带一个选择键（routing key）orange发布至转发器将会被导向队列Q1中；消息附带一个选择器black或者green将会被导向队列Q2中，所有其他的消息将会被丢弃。

### 多重绑定（multiple bindings）



使用一个绑定键（binding key）绑定多个队列是完全合法的，如图，一个附带选择键（routing key）的消息将会被转发到Q1和Q2队列中。

### 发送日志（EmittingLogs）

将这种模式应用于我们的日志系统。我们将消息发送到direct类型的转发器上而不是fanout类型。我们将日志的严重性作为选择键（routing key）。这样的话，接收程序可以根据严重性来选择接收。先创建一个转发器，channel.exchangeDeclare(EXCHANGE\_NAME,”direct”),然后我们发送一条消息，channel.basicPublish(E\_N,severity,null,m.get..)

我们将severity定义三种级别：info、warning、error

### 订阅

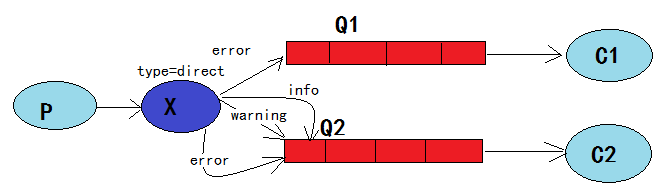
接收代码与之前的比较相似，只有一点不同：我们给我们所感兴趣的严重性类型的日志创建一个绑定。

String queueName = channel.queueDeclare().getQueue();

for(String severity : argv){

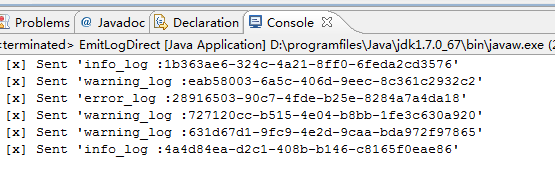
channel.queueBind(queueName,EXCHANGE\_NAME,severity);  
}

### 完整实例

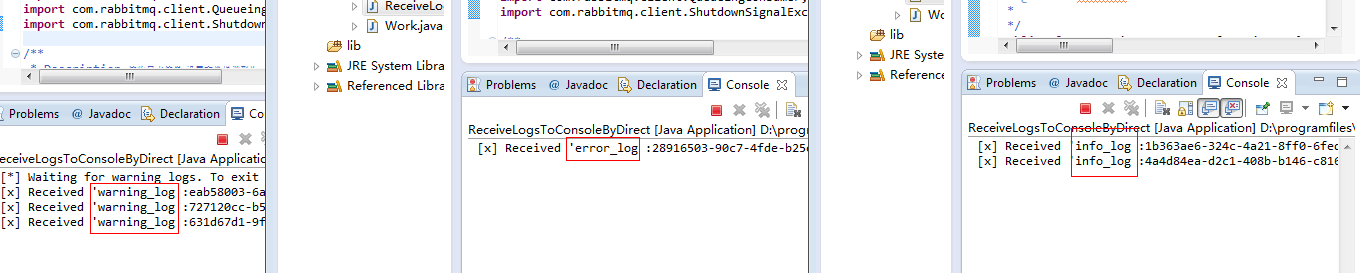


接收端启动3个实例，然后发送端发送消息，结果如下：

发送：



接收：



结论：接收者可以自定义自己感兴趣的日志类型。发送消息时可以设置RoutingKey，接收队列与交换机之间设置BindingKey，当这两个key一致时，接收者就可以接收发送者发送指定的文件了。

## 主题（Topic）

上一节我们改良的日志系统，我们使用direct类型交换机，使得接收者有能力进行选择性的接收日志，而非fanout那样全盘照收。

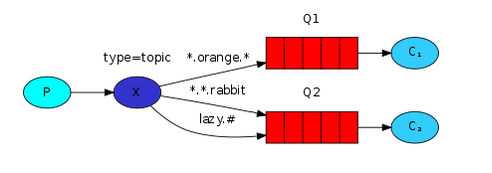
虽然使用direct改良了我们的系统，但是仍然存在一些局限性：它不能进行多重条件进行路由选择。实际，在日志系统中，我们可能不仅根据日志的级别而且还想更具日志的来源进行订阅。这个概念类似Unix工具：syslog，它转发日志基于严重性（info/warning...）和设备（auth/cron/kern…）。这样可能给我们带来更多的灵活性：我们可能只想订阅来自“cron”设备的致命日志，而不是来自kern的。为了满足需求，我们需要学习更为复杂的主题类型转换器（topic exchange）。

### 主题转发（Topic Exchange）

发往主题类型的转发器的消息不能任意的设置选择键（RoutingKey），必须是由点隔开的一系列的标识符组成。标识符可以是任何东西，但是一般都与消息的某些特性相关。合法的选择键：”stock.usd.nyse”，"nyse.vmw","quick.orange.rabbit".可以定义任何数量的标识符，但是上限为255个字节。

绑定键和选择键的形式一样，主题类型的转发器背后的逻辑和直接类型的转发器类似：一个附带特殊的选择键将会被转发到绑定键与之对应的队列中。需要注意的是：关于绑定键有两种特殊的情况：”\*”：可以匹配一个标识符；”#”：可以匹配0个或多个标识符。

### 图解



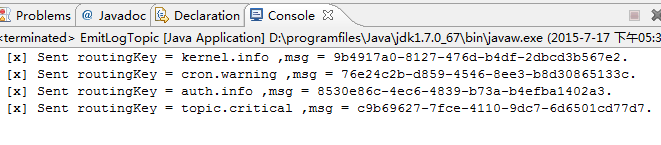
我们准备发送关于动物的消息。消息会附件一个选择键包含3个标识符（两个点隔开）。第一个标识符描述动物的速度，第二个标识符描述动物的颜色，第三个标识符描述动物的物种。如：<speed>.<color>.<species>。我们创建3个绑定键：Q1与\*.orange.\*绑定，Q2与\*.\*.rabbit和lazy.#绑定。可以简单的认为：Q1对所有橙色的动物感兴趣；Q2想要关于兔子的一切以及关于懒洋洋动物的一切。

一个附带quick.orange.rabbit的选择键的消息将会被转发到两个队列。附带lazy.orange.elephant的消息也会被转发到两个队列。另一方面quick.orange.fox只会被转发到Q1，lazy.brown.fox将会被转发到Q2。lazy.pink.rabbit虽然与两个绑定键匹配，但是也只会被转发到Q2一次。quick.brown.fox不能与任何绑定键匹配，所以会被丢弃。如果我们违反我们的约定，发送一个或者四个标识符的选择键，类似：orange，quick.orange.male.rabbit，这些选择键不能与任何绑定键匹配，所以消息将会被丢弃。另一方面，lazy.orange.male.rabbit，虽然是四个标识符，也可以与lazy.#匹配，从而转发至Q2。

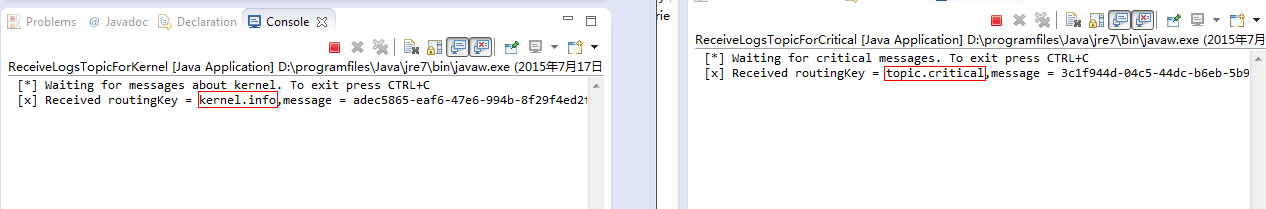
主题类型的转发器非常强大，可以实现其他类型的转发器。当一个队列与绑定键”#”绑定，将会接收所有消息，类似于fanout类型转发器，当绑定键中不包含任何”#”和”.”时，类似于direct类型转发器。

### 完整实例

发送端



接收端



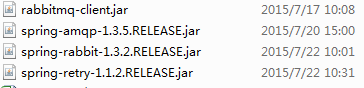
# RabbitMQ企业应用案例

## 案例

访问标准数据库信息,并将其信息同步到目的源中.

* 添加所需jar包

Spring jar包这里不再做叙述,添加Spring集成RabbitMQ所需包



* 生产者的配置 mq-producer.xml



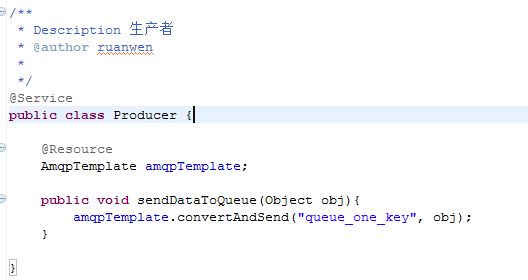
* 消费者的配置 mq-consumer.xml



* Spring集成RabbitMQ时，消息会被器解析成json格式的数据发送，由于底层使用的是jackson解析数据，这里我们使用fastjson来解析数据，因为更高效。



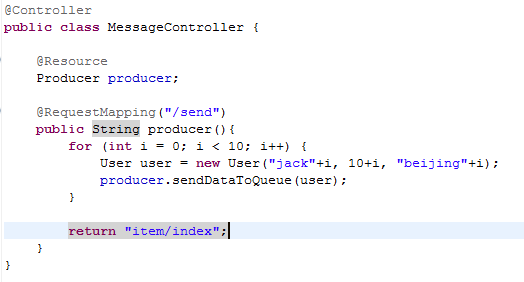
* 生产者代码，发送消息



* 配置了消费者为监听端，所以只要有消息就会被消费

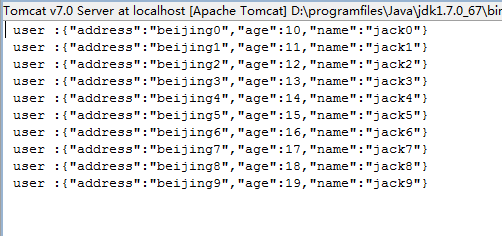


* 测试



通过地址访问：<http://localhost:8080/RabbitMQ-Spring/send>

结果数据，json格式的数据，获取到这样的数据我们就可以解析后存入到自己的库中。



从而解决了开头讲出的需求。

## 总结

在项目中，将一些无需即时返回且耗时的操作提取出来，进行了异步处理，而这种异步处理的方式大大的节省了服务器的请求响应时间，从而提高了系统的吞吐量。比如，系统间的数据交互、一个请求执行多个复杂任务等等。

# 参考资料

略。

