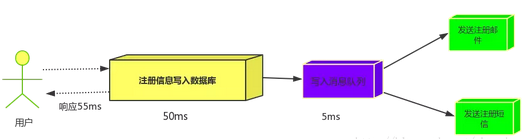
# RavvitMQ

1.消息队列

是一种应用间的异步协作机制

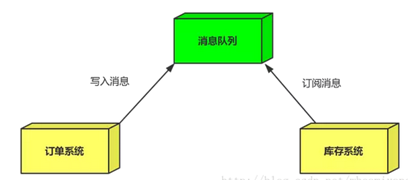
2.应用

（1）异步处理



eg：用户注册后需要发邮件和短信

（2）应用解耦



（3）流量削峰

eg：秒杀活动

3.mq的种类及其特点

（1）kafka

是LinkedIn开源的分发-订阅消息系统。目前是Apache顶级项目。kafka是基于Pull的模式来处理消息消费，追求高吞吐量，一开始的目的是用于日志的收集与传输。0.8版本支持复制，不支持事务，对消息的重复、丢失、错误没有严格要求，适合产生大量数据的互联网服务的数据收集业务。

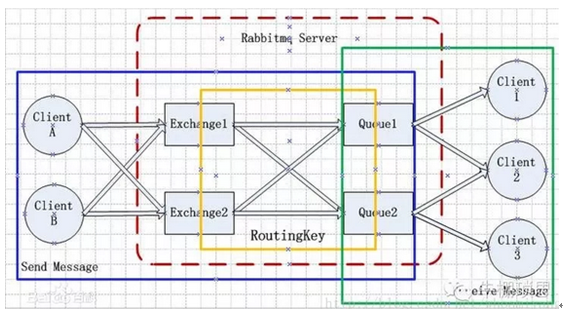
（2）RabbitMQ

是使用Erlang语言开发的开源消息队列系统，基于AMQP协议来实现。AMQP的主要特征是面向消息、队列、路由、可靠性、安全。AMQP协议更多应用在企业系统内，对数据一致性，稳定性和可靠性要求很高的场景，对性能和吞吐量的要求还在其次。

（3）RabbitMQ

是阿里开源的消息中间件，纯Java开发，具有高吞吐量、高可用性、适合大规模分布式系统应用的特点。思路源于kafka，但对消息的可靠传输及事务的性能做了优化。应用于交易、充值、流计算、消息推送、日志流试处理、binglog分发等场景。

4.RabbitMQ系统结构



（1）Message 由消息头和消息体组成。消息体不透明。消息头由rooting-key、priority（优先权）、delivery-mode（持久存储）

（2）Broker 提供一种传输服务，维护一条从生产者到消费者的路线，保证数据能按指定的方式进行传输

（3）Exchange 消息交换机，指定消息按binding规则，路由到绑定的队列

（4）binding 消息的路由规则，相当于路由表，用于路由器与队列之间的沟通

（5）queue 用来保存消息直到发送给消费者。一个消息可以进入一个或多个队列。如果消费值不取走消息，消息将一直在消息队列里。

（6）Connection 网络连接

（7）Producer(publisher) 消息生产者

（8）Consumer 消息消费者

（9）Channel 信道，连接里可以创建多个双向数据传输通道（多路复用），发布消息、订阅队列、接受消息都是通过信道来完成的。复用信道是为了降低系统资源的消耗。

（10）Virtual Host 交换器、消息队列相关的对象。一个vhost可以看成一个rabbitmp服务器，它拥有自己的队列、交换器、绑定与权限机制等。rabbitmq默认vhost是/。

一个broker里可以有多个vhost，用作不同用户的权限分离。

5.RabbitMQ的特点

1）可靠性（Reliability  
RabbitMQ 使用一些机制来保证可靠性，如持久化、传输确认、发布确认。  
2）灵活的路由（Flexible Routing）  
在消息进入队列之前，通过Exchange 来路由消息的。对于典型的路由功能，RabbitMQ 已经提供了一些内置的Exchange 来实现。针对更复杂的路由功能，可以将多个Exchange绑定在一起，也通过插件机制实现自己的 Exchange 。  
3）消息集群（Clustering）  
多个 RabbitMQ 服务器可以组成一个集群，形成一个逻辑 Broker 。  
4）高可用（Highly Available Queues）  
队列可以在集群中的机器上进行镜像，使得在部分节点出问题的情况下队列仍然可用。  
5）多种协议（Multi-protocol）  
RabbitMQ 支持多种消息队列协议，比如 STOMP、MQTT 等等。  
6）多语言客户端（Many Clients）  
RabbitMQ 几乎支持所有常用语言，比如 Java、.NET、Ruby 等等。  
7）管理界面（Management UI）  
RabbitMQ 提供了一个易用的用户界面，使得用户可以监控和管理消息 Broker 的许多方面。  
8）跟踪机制（Tracing）  
如果消息异常，RabbitMQ 提供了消息跟踪机制，使用者可以找出发生了什么。  
9）插件机制（Plugin System）  
RabbitMQ 提供了许多插件，来从多方面进行扩展，也可以编写自己的插件。

6.RabbitMQ服务安装部署

（1）安装Erlang环境

https://bintray.com/rabbitmq/rpm/erlang  
[root@master ~]# rpm -ivh erlang-20.3.8-1.el7.centos.x86\_64.rpm  
准备中...                         ################################# [100%]  
正在升级/安装...  
   1:erlang-20.3.8-1.el7.centos   ################################# [100%]  
[root@master ~]# erl  
Erlang/OTP 20 [erts-9.3.3] [source] [64-bit] [smp:1:1] [ds:1:1:10] [async-threads:10] [hipe] [kernel-poll:false]  
  
Eshell V9.3.3  (abort with ^G)  
1>

（2）下载安装RabbitMQ

https://github.com/rabbitmq/rabbitmq-server/releases/download/v3.7.6/rabbitmq-server-3.7.6-1.el7.noarch.rpm  
[root@master ~]# ll rabbitmq-server-3.7.6-1.el7.noarch.rpm  
-rw-r--r-- 1 root root 9511623 6月  27 14:00 rabbitmq-server-3.7.6-1.el7.noarch.rpm  
[root@master ~]# rpm --import https://dl.bintray.com/rabbitmq/Keys/rabbitmq-release-signing-key.asc  
[root@master ~]# yum install rabbitmq-server-3.7.6-1.el7.noarch.rpm  
[root@master ~]# chkconfig rabbitmq-server on  
注意：正在将请求转发到“systemctl enable rabbitmq-server.service”。  
Created symlink from /etc/systemd/system/multi-user.target.wants/rabbitmq-server.service to /usr/lib/systemd/system/rabbitmq-server.service.  
[root@master ~]# /sbin/service rabbitmq-server start  
Redirecting to /bin/systemctl start rabbitmq-server.service  
[root@master ~]# ps -ef|grep rabbitmq  
rabbitmq  5609     1 23 14:22 ?        00:00:02 /usr/lib64/erlang/erts-9.3.3/bin/beam.smp -W w -A 64 -MBas ageffcbf -MHas ageffcbf -MBlmbcs 512 -MHlmbcs 512 -MMmcs 30 -P 1048576 -t 5000000 -stbt db -zdbbl 1280000 -K true -- -root /usr/lib64/erlang -progname erl -- -home /var/lib/rabbitmq -- -pa /usr/lib/rabbitmq/lib/rabbitmq\_server-3.7.6/ebin -noshell -noinput -s rabbit boot -sname rabbit@master -boot start\_sasl -kernel inet\_default\_connect\_options [{nodelay,true}] -sasl errlog\_type error -sasl sasl\_error\_logger false -rabbit lager\_log\_root "/var/log/rabbitmq" -rabbit lager\_default\_file "/var/log/rabbitmq/rabbit@master.log" -rabbit lager\_upgrade\_file "/var/log/rabbitmq/rabbit@master\_upgrade.log" -rabbit enabled\_plugins\_file "/etc/rabbitmq/enabled\_plugins" -rabbit plugins\_dir "/usr/lib/rabbitmq/plugins:/usr/lib/rabbitmq/lib/rabbitmq\_server-3.7.6/plugins" -rabbit plugins\_expand\_dir "/var/lib/rabbitmq/mnesia/rabbit@master-plugins-expand" -os\_mon start\_cpu\_sup false -os\_mon start\_disksup false -os\_mon start\_memsup false -mnesia dir "/var/lib/rabbitmq/mnesia/rabbit@master" -kernel inet\_dist\_listen\_min 25672 -kernel inet\_dist\_listen\_max 25672  
rabbitmq  5752     1  0 14:22 ?        00:00:00 /usr/lib64/erlang/erts-9.3.3/bin/epmd -daemon  
rabbitmq  5921  5609  0 14:22 ?        00:00:00 erl\_child\_setup 1024  
rabbitmq  5937  5921  0 14:22 ?        00:00:00 inet\_gethost 4  
rabbitmq  5938  5937  0 14:22 ?        00:00:00 inet\_gethost 4  
root      5951  5300  0 14:23 pts/0    00:00:00 grep --color=auto rabbitmq  
[root@master ~]# netstat -lntup |grep 5672  
-bash: netstat: 未找到命令  
[root@master ~]# netstat -lntup |grep 5672  
tcp        0      0 0.0.0.0:25672           0.0.0.0:\*               LISTEN      5609/beam.smp         
tcp6       0      0 :::5672                 :::\*                    LISTEN      5609/beam.smp

（3）查看状态信息

[root@master ~]# rabbitmqctl status  
Status of node rabbit@master ...  
[{pid,5609},  
 {running\_applications,  
     [{rabbit,"RabbitMQ","3.7.6"},  
      {mnesia,"MNESIA  CXC 138 12","4.15.3"},  
      {rabbit\_common,  
          "Modules shared by rabbitmq-server and rabbitmq-erlang-client",  
          "3.7.6"},  
      {ranch\_proxy\_protocol,"Ranch Proxy Protocol Transport","1.5.0"},  
      {ranch,"Socket acceptor pool for TCP protocols.","1.5.0"},  
      {ssl,"Erlang/OTP SSL application","8.2.6"},  
      {public\_key,"Public key infrastructure","1.5.2"},  
      {asn1,"The Erlang ASN1 compiler version 5.0.5","5.0.5"},  
      {inets,"INETS  CXC 138 49","6.5.2"},  
      {jsx,"a streaming, evented json parsing toolkit","2.8.2"},  
      {os\_mon,"CPO  CXC 138 46","2.4.4"},  
      {xmerl,"XML parser","1.3.16"},  
      {crypto,"CRYPTO","4.2.2"},  
      {recon,"Diagnostic tools for production use","2.3.2"},  
      {lager,"Erlang logging framework","3.5.1"},  
      {goldrush,"Erlang event stream processor","0.1.9"},  
      {compiler,"ERTS  CXC 138 10","7.1.5"},  
      {syntax\_tools,"Syntax tools","2.1.4"},  
      {syslog,"An RFC 3164 and RFC 5424 compliant logging framework.","3.4.2"},  
      {sasl,"SASL  CXC 138 11","3.1.2"},  
      {stdlib,"ERTS  CXC 138 10","3.4.5"},  
      {kernel,"ERTS  CXC 138 10","5.4.3"}]},  
 {os,{unix,linux}},  
 {erlang\_version,  
     "Erlang/OTP 20 [erts-9.3.3] [source] [64-bit] [smp:1:1] [ds:1:1:10] [async-threads:64] [hipe] [kernel-poll:true]\n"},  
 {memory,  
     [{connection\_readers,0},  
      {connection\_writers,0},  
      {connection\_channels,0},  
      {connection\_other,0},  
      {queue\_procs,0},  
      {queue\_slave\_procs,0},  
      {plugins,5936},  
      {other\_proc,20754544},  
      {metrics,184432},  
      {mgmt\_db,0},  
      {mnesia,72912},  
      {other\_ets,1873688},  
      {binary,55376},  
      {msg\_index,28720},  
      {code,25082003},  
      {atom,1041593},  
      {other\_system,9173572},  
      {allocated\_unused,8315896},  
      {reserved\_unallocated,3670016},  
      {strategy,rss},  
      {total,[{erlang,58272776},{rss,70258688},{allocated,66588672}]}]},  
 {alarms,[]},  
 {listeners,[{clustering,25672,"::"},{amqp,5672,"::"}]},  
 {vm\_memory\_calculation\_strategy,rss},  
 {vm\_memory\_high\_watermark,0.4},  
 {vm\_memory\_limit,3281294131},  
 {disk\_free\_limit,50000000},  
 {disk\_free,122394349568},  
 {file\_descriptors,  
     [{total\_limit,924},{total\_used,2},{sockets\_limit,829},{sockets\_used,0}]},  
 {processes,[{limit,1048576},{used,204}]},  
 {run\_queue,0},  
 {uptime,111},  
 {kernel,{net\_ticktime,60}}]

7.RabbitMQ CLI介绍

自带命令行工具

（1）rabbitmqct 管理与操作命令

停止节点

访问节点状态，有效配置，运行状况检查

虚拟主机管理

用户和权限管理

政策管理

列出队列、连接、渠道、交流、消费者

集群成员管理

[root@master ~]# rabbitmqctl list\_bindings    #查看绑定信息  
Listing bindings for vhost /...  
[root@master ~]# rabbitmqctl list\_exchanges   #查看交换器  
Listing exchanges for vhost / ...  
amq.rabbitmq.trace    topic  
amq.match    headers  
    direct  
amq.headers    headers  
amq.direct    direct  
amq.rabbitmq.event    topic  
amq.topic    topic  
amq.fanout    fanout  
[root@master ~]# rabbitmqctl list\_queues   #查看队列  
Timeout: 60.0 seconds ...  
Listing queues for vhost / ...  
更多操作内容请参考：http://www.rabbitmq.com/rabbitmqctl.8.html

（2）rabbitmq-plugins 是一个管理插件的工具

[root@master ~]# rabbitmq-plugins --help  
Usage:  
rabbitmq-plugins [-n <node>] [-l] [-q] <command> [<command options>]  
  
General options:  
    -n node  
    -q quiet  
    -l longnames  
  
Default node is "rabbit@server", where `server` is the local hostname. On a host  
named "server.example.com", the node name of the RabbitMQ Erlang node will  
usually be rabbit@server (unless RABBITMQ\_NODENAME has been set to some  
non-default value at broker startup time). The output of hostname -s is usually  
the correct suffix to use after the "@" sign. See rabbitmq-server(1) for  
details of configuring the RabbitMQ broker.  
  
Quiet output mode is selected with the "-q" flag. Informational messages are  
suppressed when quiet mode is in effect.  
  
If RabbitMQ broker uses long node names for erlang distribution, "longnames"  
option should be specified.  
  
Some commands accept an optional virtual host parameter for which  
to display results. The default value is "/".  
  
Commands:  
    disable <plugin>|--all [--offline] [--online]  
    enable <plugin>|--all [--offline] [--online]  
    help <command>  
    list [pattern] [--verbose] [--minimal] [--enabled] [--implicitly-enabled]  
    set [<plugin>] [--offline] [--online]  
  
<timeout> - operation timeout in seconds. Default is "infinity".  
参考文章：http://www.rabbitmq.com/rabbitmq-plugins.8.html

（3）rabbitmqadmin 执行一些与web界面相同的操作，只是一个专门的HTTP客户端

安装管理插件后，http://localhost:15672/cli/rabbitmqadmin 进行下载

[root@master ~]# wget https://raw.githubusercontent.com/rabbitmq/rabbitmq-management/v3.7.6/bin/rabbitmqadmin  
[root@master ~]# cp rabbitmqadmin /usr/local/bin/  
[root@master ~]# chmod +x /usr/local/bin/rabbitmqadmin

8.RabbitMQ集群

RabbitMQ 是用 erlang 开发的，集群非常方便，因为 erlang 天生就是一门分布式语言，但其本身并不支持负载均衡。

**RabbitMQ 的集群节点包括：**

1）内存节点  
内存节点就是将所有数据放在内存，只保存状态到内存（例外的情况：持久的queue内容将被保存到 disk）  
2）磁盘节点。  
 磁盘节点将数据放在磁盘，保存状态到内存和磁盘，内存节点虽然不写入磁盘，但是它执行比磁盘节点要好，集群中，只需要一个磁盘节点来保存状态 就足够了，如果集群中只有内存节点，那么不能停止它们，否则所有的状态，消息等都会丢失。

不过，如前文所述，如果在投递消息时，打开了消息的持久化，那即使是内存节点，数据还是安全的放在磁盘。一个 RabbitMQ 集群中可以共享 user、vhost、queue、exchange 等，所有的数据和状态都是必须在所有节点上复制的，一个例外是那些当前只属于创建它的节点的消息队列，尽管它们可见且可被所有节点读取。 RabbitMQ 节点可以动态地加入到集群中，一个节点它可以加入到集群中，也可以从集群环集群进行一个基本的负载均衡。

Rabbit 模式大概分为以下三种：

1）单一模式  
2）普通模式  
3）镜像模式

**单一模式：**最简单的情况，非集群模式。

**普通模式：**默认的集群模式。

对于 Queue 来说，消息实体只存在于其中一个节点，A、B 两个节点仅有相同的元数据，即队列结构。当消息进入 A 节点的 Queue 中后，consumer 从 B 节点拉取时，RabbitMQ 会临时在 A、B 间进行消息传输，把 A 中的消息实体取出并经过 B 发送给 consumer. 所以 consumer 应尽量连接每一个节点，从中取消息。即对于同一个逻辑队列，要在多个节点建立物理 Queue。否则无论 consumer 连 A 或 B，出口总在 A，会产生瓶颈。该模式存在一个问题就是当 A 节点故障后，B 节点无法取到 A 节点中还未消费的消息实体。如果做了消息持久化，那么得等 A 节点恢复，然后才可被消费。

**镜像模式：**

把需要的队列做成镜像队列，存在于多个节点，属于 RabbitMQ 的 HA 方案。  
该模式解决了上述问题，其实质和普通模式不同之处在于，消息实体会主动在镜像节点间同步，而不是在 consumer 取数据时临时拉取。该模式带来的副作用也很明显，除了降低系统性能外，如果镜像队列数量过多，加之大量的消息进入，集群内部的网络带宽将会被这种同步通讯大大消耗掉,所，在对可靠性要求较高的场合中适用。

9.任务分发机制

（1）Round-robin dispathching循环分发

RabbbitMQ的分发机制非常适合扩展,而且它是专门为并发程序设计的,如果现在load加重,那么只需要创建更多的Consumer来进行任务处理。

（2）Message acknowledgment消息确认

消息确认为了保证数据不被丢失,RabbitMQ支持消息确认机制,为了保证数据能被正确处理而不仅仅是被Consumer收到,那么我们不能采用no-ack，而应该是在处理完数据之后发送ack.   
在处理完数据之后发送ack,就是告诉RabbitMQ数据已经被接收,处理完成,RabbitMQ可以安全的删除它了.   
如果Consumer退出了但是没有发送ack,那么RabbitMQ就会把这个Message发送到下一个Consumer，这样就保证在Consumer异常退出情况下数据也不会丢失.   
RabbitMQ它没有用到超时机制.RabbitMQ仅仅通过Consumer的连接中断来确认该Message并没有正确处理，也就是说RabbitMQ给了Consumer足够长的时间做数据处理。   
如果忘记ack,那么当Consumer退出时,Mesage会重新分发,然后RabbitMQ会占用越来越多的内存.

10.Message durability 消息持久化

要持久化队列queue的持久化需要在声明时指定durable=True;   
**这里要注意,队列的名字一定要是Broker中不存在的,不然不能改变此队列的任何属性**.   
 队列和交换机有一个创建时候指定的标志durable,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,它不表示说在队列中的消息会在重启后恢复   
消息持久化包括3部分   
1. exchange持久化,在声明时指定durable => true

channel.ExchangeDeclare(ExchangeName, "direct", durable: true, autoDelete: false, arguments: null);//声明消息队列，且为可持久化的

2.queue持久化,在声明时指定durable => true

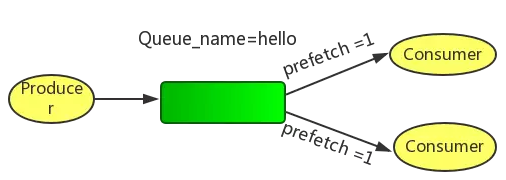
channel.QueueDeclare(QueueName, durable: true, exclusive: false, autoDelete: false, arguments: null);//声明消息队列，且为可持久化的

3.消息持久化,在投递时指定delivery\_mode => 2(1是非持久化).

channel.basicPublish("", queueName, MessageProperties.PERSISTENT\_TEXT\_PLAIN, msg.getBytes());

如果exchange和queue都是持久化的,那么它们之间的binding也是持久化的,如果exchange和queue两者之间有一个持久化，一个非持久化,则不允许建立绑定.   
 注意：一旦创建了队列和交换机,就不能修改其标志了,例如,创建了一个non-durable的队列,然后想把它改变成durable的,唯一的办法就是删除这个队列然后重现创建。l

11.公平分发 Fair dispath

你可能也注意到了,分发机制不是那么优雅,默认状态下,RabbitMQ将第n个Message分发给第n个Consumer。n是取余后的,它不管Consumer是否还有unacked Message，只是按照这个默认的机制进行分发.   
 那么如果有个Consumer工作比较重,那么就会导致有的Consumer基本没事可做,有的Consumer却毫无休息的机会,那么,Rabbit是如何处理这种问题呢?   
   
 通过basic.qos方法设置prefetch\_count=1，这样RabbitMQ就会使得每个Consumer在同一个时间点最多处理一个Message，换句话说,在接收到该Consumer的ack前,它不会将新的Message分发给它

channel.basic\_qos(prefetch\_count=1)

**注意**，这种方法可能会导致queue满。当然，这种情况下你可能需要添加更多的Consumer，或者创建更多的virtualHost来细化你的设计。

12.分发到多个Consumer

（1）Exchange

先来温习以下交换机路由的几种类型:   
 Direct Exchange:直接匹配,通过Exchange名称+RountingKey来发送与接收消息.   
 Fanout Exchange:广播订阅,向所有的消费者发布消息,但是只有消费者将队列绑定到该路由器才能收到消息,忽略Routing Key.   
 Topic Exchange：主题匹配订阅,这里的主题指的是RoutingKey,RoutingKey可以采用通配符,如:\*或#，RoutingKey命名采用.来分隔多个词,只有消息这将队列绑定到该路由器且指定RoutingKey符合匹配规则时才能收到消息;   
 Headers Exchange:消息头订阅,消息发布前,为消息定义一个或多个键值对的消息头,然后消费者接收消息同时需要定义类似的键值对请求头:(如:x-mactch=all或者x\_match=any)，只有请求头与消息头匹配,才能接收消息,忽略RoutingKey.   
 默认的exchange:如果用空字符串去声明一个exchange，那么系统就会使用”amq.direct”这个exchange，我们创建一个queue时,默认的都会有一个和新建queue同名的routingKey绑定到这个默认的exchange上去

channel.BasicPublish("", "TaskQueue", properties, bytes);

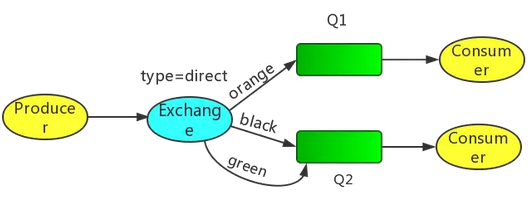
因为在第一个参数选择了默认的exchange，而我们申明的队列叫TaskQueue，所以默认的，它在新建一个也叫TaskQueue的routingKey，并绑定在默认的exchange上，导致了我们可以在第二个参数routingKey中写TaskQueue，这样它就会找到定义的同名的queue，并把消息放进去。   
 如果有两个接收程序都是用了同一个的queue和相同的routingKey去绑定direct exchange的话，分发的行为是负载均衡的，也就是说第一个是程序1收到，第二个是程序2收到，以此类推。   
 如果有两个接收程序用了各自的queue，但使用相同的routingKey去绑定direct exchange的话，分发的行为是复制的，也就是说每个程序都会收到这个消息的副本。行为相当于fanout类型的exchange。   
下面详细来说:

（2）Bindings 绑定

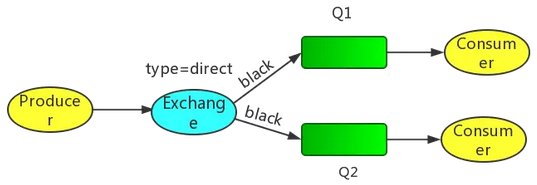
绑定其实就是关联了exchange和queue，或者这么说:queue对exchange的内容感兴趣,exchange要把它的Message deliver到queue。

（3）Direct exchange

Driect exchange的路由算法非常简单:通过bindingkey的完全匹配，可以用下图来说明.

  
   
 Exchange和两个队列绑定在一起,Q1的bindingkey是orange，Q2的binding key是black和green.   
 当Producer publish key是orange时,exchange会把它放到Q1上,如果是black或green就会到Q2上,其余的Message被丢弃.

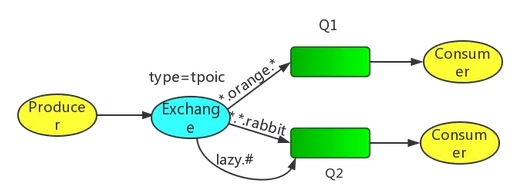
（4）Multiple bindings

多个queue绑定同一个key也是可以的,对于下图的例子,Q1和Q2都绑定了black,对于routing key是black的Message，会被deliver到Q1和Q2，其余的Message都会被丢弃.   
 

（5）Topic exchange

对于Message的routing\_key是有限制的，不能使任意的。格式是以点号“.”分割的字符表。比如：”stock.usd.nyse”, “nyse.vmw”, “quick.orange.rabbit”。你可以放任意的key在routing\_key中，当然最长不能超过255 bytes。   
对于routing\_key，有两个特殊字符

\*(星号)代表任意一个单词

#(hash)0个或多个单词   
   
 Producer发送消息时需要设置routing\_key，routing\_key包含三个单词和连个点号o,第一个key描述了celerity(灵巧),第二个是color(色彩),第三个是物种:   
在这里我们创建了两个绑定： Q1 的binding key 是”.orange.“； Q2 是 “..rabbit” 和 “lazy.#”：

Q1感兴趣所有orange颜色的动物

Q2感兴趣所有rabbits和所有的lazy的.   
 例子:rounting\_key 为 “quick.orange.rabbit”将会发送到Q1和Q2中   
rounting\_key 为”lazy.orange.rabbit.hujj.ddd”会被投递到Q2中,#匹配0个或多个单词。

13.消息序列化

RabbitMQ使用ProtoBuf序列化消息,它可作为RabbitMQ的Message的数据格式进行传输,由于是结构化的数据,这样就极大的方便了Consumer的数据高效处理,当然也可以使用XML，与XML相比,ProtoBuf有以下优势:   
 1.简单   
 2.size小了3-10倍   
 3.速度快了20-100倍   
 4.易于编程   
 6.减少了语义的歧义.   
 ProtoBuf具有速度和空间的优势，使得它现在应用非常广泛