# 项目课程系列之RabbitMQ3.8.1

## 尚硅谷JavaEE教研组

## 版本：V1.0

# 课程计划

* AMQP和JMS
* RabbitMQ介绍
* 下载与安装
* RabbitMQ管理界面
* 五种消息模型
  + 基本消息模型
  + Work消息模型
  + 订阅模型-Fanout
  + 订阅模型-Direct
  + 订阅模型-Topic
* 持久化
* Spring AMQP

# 1. RabbitMQ

## 1.1. 搜索与商品服务的问题

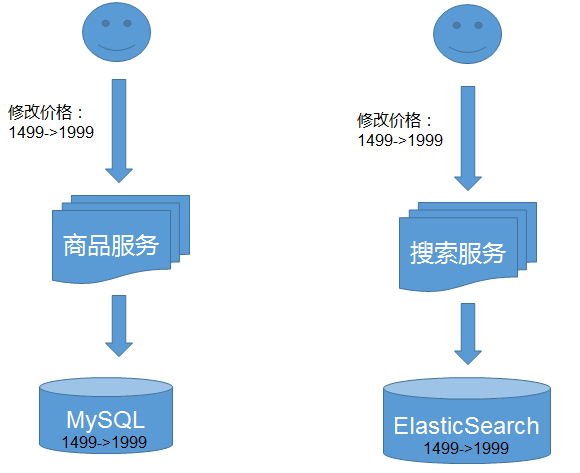
假设我们已经完成了商品详情和搜索系统的开发。我们思考一下，是否存在问题？

* 商品的原始数据保存在数据库中，增删改查都在数据库中完成。
* 搜索服务数据来源是索引库，如果数据库商品发生变化，索引库数据能否及时更新。

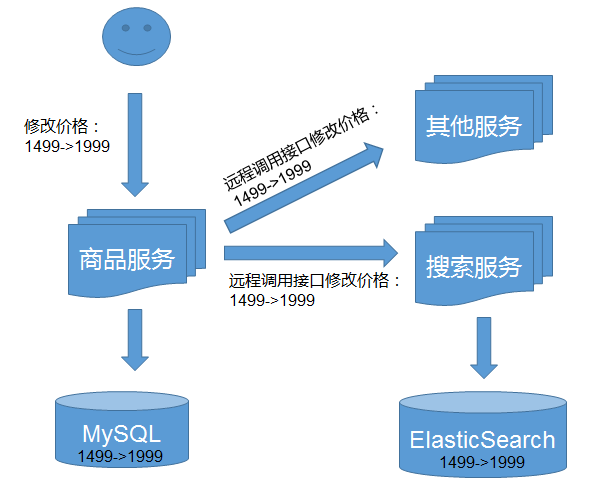
如果我们在后台修改了商品的价格，搜索页面依然是旧的价格，这样显然不对。该如何解决？

这里有两种解决方案：

* **方案1**：每当后台对商品做增删改操作，同时要修改索引库数据

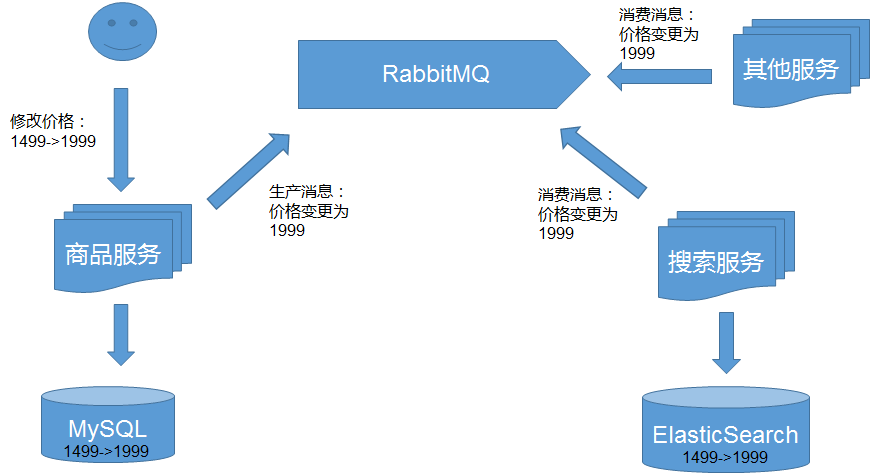


* **方案2**：搜索服务对外提供操作接口，后台在商品增删改后，调用接口



以上两种方式都有同一个严重问题：就是代码耦合，后台服务中需要嵌入搜索和商品页面服务，违背了微服务的独立原则。

所以，我们会通过另外一种方式来解决这个问题：消息队列



## 1.2. 消息队列（MQ）

### 1.2.1. 什么是消息队列

消息队列，即MQ，Message Queue。

计算机生成了可选文字:
MQ全称为Messageoueue，消息队歹d(MO）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。；11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。其中较为成熟的MQ产品有旧MWEBSp日EREMO等等。
中文名
消息队列
外文名
简称
MessageQueue
MO

消息队列是典型的：生产者、消费者模型。生产者不断向消息队列中生产消息，消费者不断的从队列中获取消息。因为消息的生产和消费都是异步的，而且只关心消息的发送和接收，没有业务逻辑的侵入，这样就实现了生产者和消费者的解耦。

结合前面所说的问题：

* 商品服务对商品增删改以后，无需去操作索引库，只是发送一条消息，也不关心消息被谁接收。
* 搜索服务接收消息，去处理索引库。

如果以后有其它系统也依赖商品服务的数据，同样监听消息即可，商品服务无需任何代码修改。

### 1.2.2. AMQP和JMS

MQ是消息通信的模型，并不是具体实现。现在实现MQ的有两种主流方式：**AMQP、JMS**。

计算机生成了可选文字:
AMQP，编。
胡OP，即AdvanCedMessageoueuingProtocol一个提供统一消息服务的应用层标准高级消息队列协议．是应用层协议的
一个开放标准．为面向消息的中间件设计。基于此协议的客户端与消息中间件可传递消息，并不受客户端冲间件不同产品，不同
的开发语言等条件的限制。Erlang中的实现有Rab匕比MO等。
中文名
外文名
高级消息队列协议
AdvancedMessageOueuingProtocol
应用粉或
应用层标准协议
计算机

计算机生成了可选文字:
JMS
通常而言提到JMS(JavaMessageservice）实际上是指JMSAPI.JMS是由Sun公司早期提出的消息标准
旨在为Java应用提供统一的消息操作，包括create、send、receive
等。JMS已经成划avaEnterPriseEdition的一部分。从便用角度看，JMS和JDBC担任差不多的角色，用户都是恨
据相应的接口可以和实现了JMS的服务进行通信，进行相关的操作。

**两者间的区别和联系：**

* JMS是定义了统一接口，对消息操作进行统一；AMQP通过规定协议统一数据交互的格式；
* JMS限定了必须使用Java语言；AMQP只是协议，不规定实现方式，因此是跨语言的
* JMS规定了两种消息模型（queue, topic）；而AMQP的消息模型更加丰富

### 1.2.3. 常见MQ产品

计算机生成了可选文字:
生
AMQP；肖息服务器RabbitMQ
httPS办盯WwOS山inane如lrabbi加q
RabbitMQ是臼LShift提供的一个AdvancedMessageQueuingProtocol(AMQp）的开源实现．由以高性能、健壮以及可伸缩性出名的Edang写成
RabbitMQ同类软件
DJMS消息服务器ActiveMQa分布式发布订阅消息系统Kafka
a分布式消息中I'＠件Metamorphosisa轻量级消息队列beanstalkd
a.NET消息中间件DotNetMQDErlangMQTT消息服务器emqttd
口分布式消息系统Jafka口开源消息系统NSQ

* ActiveMQ：基于JMS
* RabbitMQ：基于AMQP协议，erlang语言开发，稳定性好
* RocketMQ：基于JMS，阿里巴巴产品，目前交由Apache基金会
* Kafka：分布式消息系统，高吞吐量，处理日志，Scala和Java编写，Apache

### 1.2.4. RabbitMQ

* RabbitMQ是基于AMQP的一款消息管理系统
* 官网： <http://www.rabbitmq.com/>
* 官方教程：<http://www.rabbitmq.com/getstarted.html>
* RabbitMQ是一个开源的，在AMQP基础上完成的，可服用的企业消息系统。
* 支持主流的操作系统，Linux,Windows,MacOX等
* 多种开发语言支持，Java,Python,Ruby,.Net,PHP,C/C++,node.js等

### 1.2.5. MQ三大主要功能

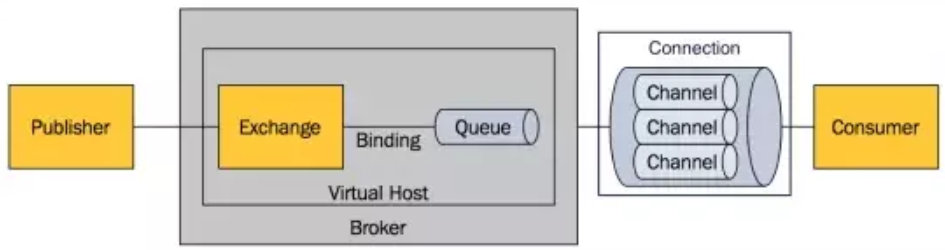
* 异步
* 解耦
* 削峰

### 1.2.6 RabbitMQ 特点

* RabbitMQ 是一个由 Erlang 语言开发的 AMQP 的开源实现。
* 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。
* 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 。
* 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 。
* 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。
* 多种协议（Multi-protocol）  
  RabbitMQ 支持多种消息队列协议，比如 STOMP、MQTT 等等。
* 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。
* 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。
* 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。
* 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。

### 1.2.7 RabbitMQ 基本概念

上面只是最简单抽象的描述，具体到 RabbitMQ 则有更详细的概念需要解释。上面介绍过 RabbitMQ 是 AMQP 协议的一个开源实现，所以其内部实际上也是 AMQP 中的基本概念：



* Message  
  消息，消息是不具名的，它由消息头和消息体组成。消息体是不透明的，而消息头则由一系列的可选属性组成，这些属性包括routing-key（路由键）、priority（相对于其他消息的优先权）、delivery-mode（指出该消息可能需要持久性存储）等。
* Publisher  
  消息的生产者，也是一个向交换器发布消息的客户端应用程序。
* Exchange  
  交换器，用来接收生产者发送的消息并将这些消息路由给服务器中的队列。
* Binding  
  绑定，用于消息队列和交换器之间的关联。一个绑定就是基于路由键将交换器和消息队列连接起来的路由规则，所以可以将交换器理解成一个由绑定构成的路由表。
* Queue  
  消息队列，用来保存消息直到发送给消费者。它是消息的容器，也是消息的终点。一个消息可投入一个或多个队列。消息一直在队列里面，等待消费者连接到这个队列将其取走。
* Connection  
  网络连接，比如一个TCP连接。
* Channel  
  信道，多路复用连接中的一条独立的双向数据流通道。信道是建立在真实的TCP连接内地虚拟连接，AMQP 命令都是通过信道发出去的，不管是发布消息、订阅队列还是接收消息，这些动作都是通过信道完成。因为对于操作系统来说建立和销毁 TCP 都是非常昂贵的开销，所以引入了信道的概念，以复用一条 TCP 连接。
* Consumer  
  消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。
* Virtual Host  
  虚拟主机，表示一批交换器、消息队列和相关对象。虚拟主机是共享相同的身份认证和加密环境的独立服务器域。每个 vhost 本质上就是一个 mini 版的 RabbitMQ 服务器，拥有自己的队列、交换器、绑定和权限机制。vhost 是 AMQP 概念的基础，必须在连接时指定，RabbitMQ 默认的 vhost 是 / 。
* Broker  
  表示消息队列服务器实体。

## 1.3. 下载和安装

### 1.3.1. 下载

**1.下载Erlang的rpm包**

RabbitMQ是Erlang语言编写，所以Erang环境必须要有，注：Erlang环境一定要与RabbitMQ版本匹配：<https://www.rabbitmq.com/which-erlang.html>

计算机生成了可选文字:
pabbitMQ
Verslon
MinimUm
required
Ertang/OTP
MaXimUm
Supported
ErLang/OTP
NOt65
381
380
22X
Ertang22115recommended.
Ertang22xdroppedsupportforHIPE
37.21
37.20
37.19
3718
37.17
3716
3715
3714
3713
37.12
3711
37.10
37.9
378
377
37.6
37.5
374
373
372
37.1
370
22X
一Lm·g/｛弓丁「20.3.x二＝巨port怪．discont旧ued
·Erlang22xdroppedsupportforHIPE
22X
·丛迎g丛嗯困t迎四业业仙丛丝
·TLSvloandTLSvlisup匹退15dlsabtedbyde怕uttonErlang22沉
一」
.
213X
业还亘困兰
·ForthebestTLSsupport.thetatestversionofErtang/OTPZi.3.x15recommended
203X
·ForthebestTLSsupport.thetatestversionofErtang/OTPZo.3.x15recommended
.Ertangversionspriorto19.3.6.4haveknownbugs(e.g旦丝乡3旦．旦丝乡组）thatcan
preventRabbltMQnodesfromacceptingconnectlons(inctudingfromCLItoots)
andstopping
．央rsionsprlorto19364arevutnerabtetothe巫延迫1遨迈迷（CVE一2017一1000385)
·OnWindows.Ertang/OTP20.2changed〔｛efaultcoo从leflteIc::,tl·：·，1
20.3。X

Erlang下载地址：<https://www.rabbitmq.com/releases/erlang/>（根据自身需求及匹配关系，下载对应rpm包）

<https://dl.bintray.com/rabbitmq-erlang/rpm/erlang/21/el/7/x86_64/erlang-21.3.8.9-1.el7.x86_64.rpm>

**2.下载socat的rpm包**

rabbitmq安装依赖于socat，所以需要下载socat。

socat下载地址：<http://repo.iotti.biz/CentOS/7/x86_64/socat-1.7.3.2-5.el7.lux.x86_64.rpm>

**3.下载RabbitMQ的rpm包**

RabbitMQ下载地址：<https://www.rabbitmq.com/download.html>（根据自身需求及匹配关系，下载对应rpm包）rabbitmq-server-3.8.1-1.el7.noarch.rpm

计算机生成了可选文字:
目rabbitmq.com/download.html
幽RabbitMQ、pivota.
FeatUreS
GetStarted
Suppo比
Community
DOCS
BLog
Down[oadingandInstaL[ingRabbitMO
TheLatestreteaseofRabbltMQ153.8.1See迎旦Dg旦3旦9forreteasenotesSee旦二卫
reLeaseserlesaresupported.
,12巡卫工卫旦廷丝生旦tofindoutwhat
RabbitMO56四el
InstatlationGUide
L1nUX.BSD.UNIXDebianUbUntU
C凶颤C巨皿r帅血引5区如S
windows业士巡旦土垫匹叮进坦丛玺些｝旦皿华凶鲤
MacoS:Homebrew}GenericbinarybuiLd
〕r·。OTPforRabbltMQ
DOWnloadsOnGitHUb
V叮IndOWSinstaLLer
Deblan,UbUntU
RHEL/CentOS8.X
尸“接下“{
巨票夔惠习…～‘旦～'~‘吻g皿
皿创旦鱼卫到丛鱼卫鱼丫
助园。竖巨皿以
!nThiSSeCtion
}nstatLWindOWS
}nsta比DebisnandUbUntU
!nsta比Rp州一basedLlnuX
}nsta【LHomebreW
}nsta比Wind0WS(manL通己．
}nst己．1Gen曰旧I3inar'.[:.l＿』｝比
}）一‘土〕15．二｛云
r一二土三、三二二
旦pgrade
BLue一greende例卫yment-
匕旦父过卫pgr明e
题QportedPLatforms
Chang旦旦g
ErLang央rslons
旦gnedPaCkag丝
JaVaCtlentDOWnLOadS
NETC[lentDOWn{oads
〔rLangCLlentD洲n}oads
Communit迎坦gJ归互
鱼旦pshots

### 1.3.2. 安装

#### 1.安装Erlang、Socat、RabbitMQ

计算机生成了可选文字:
圆erlang一21.3.a.9一1.eI7.xa6一rpm
圆rabbitmq一，erver一3.5.1一1.eI7.noarc卜．rpm
园socat一1.7.3.2一1.eI6.lux.x86se64.rpm

**①rpm -ivh erlang-21.3.8.9-1.el7.x86\_64.rpm**

**②rpm -ivh socat-1.7.3.2-1.el6.lux.x86\_64.rpm**

在安装rabbitmq之前需要先安装socat，否则，报错。

计算机生成了可选文字:
[rootozhanqyuopt]#rpm
警告：
错误：
robbltmq一server一3.
依赖检测失败：
一Ivhrabbltmq一server一3.8.1一1.e17.noarch.rpm
8.1一1.e17.noorch.rpm：头V4RSA/SHA256Slgnature，密钥ID6026dfca:NOKEY
socat被robbltmq一server一3.8.1一1.e17.noarch需要

可以采用yum安装方式：yum install socat，我们这里采用rpm安装方式

**③rpm -ivh rabbitmq-server-3.8.1-1.el7.noarch.rpm**

/usr/lib/rabbitmq/lib/rabbitmq\_server-3.8.1/sbin

#### 2.启用管理插件

rabbitmq-plugins enable rabbitmq\_management

计算机生成了可选文字:
[root坦zhongyuoptl#robbitmq一pluglnsenablerobbltmq_manogement
Enablingpluglnsonnoderabblt@zhangyu:
rabbltmq_monagement
Thefollowlngpluglnshavebeenconfigured:
rabbitmq_management
rabbltmq_management_agent
rabbltmq一eb_d1SpatCh
Applylngpluglnconfiguratlontorabblt但zhangyu…
Thefollowlngpluglnshovebeenenobled:
rabbltmq_management
rabbltmq_management_agent
rabbltmq少eb_dispatch
started3pluglns.

#### 3.启动RabbitMQ

systemctl start rabbitmq-server.service

systemctl status rabbitmq-server.service

systemctl restart rabbitmq-server.service

systemctl stop rabbitmq-server.service

计算机生成了可选文字:
【root@zhongyuopt]#systemctlstatusrobbltmq一server.servlce
.rabbltmq一server.servlce一RabbltHQbroker
Looded:looded(/usr/lib/systemd/system/rabbltmq一server.servlce;dlsabled二vendorpreset:
dlsabled)
Active:active(running)slnce六2019一11一1623:54:48CST;75ago
HalnPID:9015(beam.smp)
StotUS:"InltlallZed"
CGroup:/system.slice/rabbitmq一server.service
9015/usr/11b64/erlang/erts一10.3.5.6/bin/beam.smp一树w一A128一HBasageffcbf
9231/usr/11b64/erlang/erts一10.3.5.6/bln/epmd一doemon
9475erl_chlld_setup32768
9510inet_gethost4
9511Inet_gethost4
11月
11月
11月
11月
11月
11月
11月
11月
11月
11月
Hint:
1623:54:48zhangyu.atgulgurabbltmq一server【9015』：Docguldes:https://rabbltmq.co…1
1623:54:48zhangyu.atgulgurabbltmq一server[9015]:Support:https://rabbltmq.co…1
1623:54:48zhangyu.otgulgurabbltmq一server【9015』：Tutorlals:https://rabbltmq.co…1
1623:54:48zhangyu.atgulgurabbltmq一server[9015]:Monltoring:https://rabbltmq.co…1
1623:54:48zhangyu.atgulgurabbltmq一server【9015]:Logs:/var/109/rabbltmq/rabblt＠…g
1623:54:48zhangyu.atgulgurabbltmq一server【9015]:/var/109/rabbltmq/rabbltozhangy…g
1623:54:48zhongyu.otgulgurobbitmq一server【9015』：Conflgflle(s):(none)
1623:54:48zhangyu.atguigurabbltmq一server[9015]:Startingbroker…systemdunit
1623:54:48zhangyu.otgulgusystemd[11:StartedRobbltHQbroker.
1623:54:50zhangyu.atgulgurabbltmq一server[9015]:completedwith0pluglns.
Some11neswereellipslzed,use一1toshowinfull.

#### 4.查看进程

ps -ef | grep rabbitmq

计算机生成了可选文字:
【root坦zhongyuopt】＃ps一ef}greprobbltmq
rabbitmq90151723:54?00:00:11/usr/lib64/erlang/erts一10.3.5.6/bin/beam.smp一Ww一A128
一HBasageffcbf一HHasageffcbf一HBlmbcs512一HHlmbcs512一HHmcs30一P1048576一t5000000一stbtdb一zdbbll
28000一Ktrue－一root/usr/lib64/erlang一prognameerl－一home/var/11b/rabbitmq－一pa/usr/lib/rabbitm
/11b/rabbitmq_server一3.8.1/ebln一noshell一nolnput一5robbitboot一snamerabbitozhangyu一bootstart_sasl-
kernelInet_default_connect_opt10ns【｛nodelay,true｝】一SaSIerrlog_typeerror一SaSISaSI_error_10ggerfa1Se
一rabbltlager_109_root''/var/109/rabbitmq”一rabbltlager_default_flle"/var/109/rabbitmq/rabbltozhangyu.1
09'‘一rabbitlager_upgrade_flle''/var/109/rabbitmq/rabblt尽zhangyu_upgrade.log'’一rabbitfeature_flags_flle''
/vor/11b/rabbitmq/mnesl。／robblt坦zhongyu一feoture_flogs'，一robbltenobled_pluglns_flle"/etc/rabbitmq/enobled
_pluglns"‘一rabbltpluglns_dlr'"/usr/11b/rabbitmq/pluglns:/usr/11b/rabbitmq/11b/rabbitmq_server一3.8.1/plugl
ns'‘一rabbltpluglns_expand_dlr''/vor/11b/rabbitmq/mnesla/rabblt@zhangyu一pluglns一expand'’一05_monstart_cpu_
Supfa1Se一05_monStart_d1Sksupfa1Se一05_monStart_memsupfa1Se一mneSladlr'"/var/11b/rabbitmq/mnesla/rab
blt坦zhangyu'”一radata_dlr''/var/11b/rabbitmq/mnesla/rabblt@zhangyu/quorum'”一kernelinet_dlst_11sten_min25
672一keFnelInetd1St11StenmdX25672
rabbitoq92311023:54?00:00:00/usr/lib64/erlang/erts一10.3.5.6/bln/epmd一daemon
rabbitmq94759015023:54?00:00:00erl_child_setup32768
rabbitoq95109475023:54?00:00:00Inet_gethost4
rabbitmq95119510023:54?00:00:00Inet_gethost4
root98102762023:57pts/000:00:00grep一color=autorabbitmq

### 1.3.3. 测试

* 关闭防火墙：systemctl stop firewalld.service
* 在web浏览器中输入地址：http://虚拟机ip:15672/
* 输入默认账号密码: guest : guest，guest用户默认不允许远程连接。

计算机生成了可选文字:
▲不安全｝192.168.137.137二15672
O,
☆
出Ra匕b1tMO
USeFname:
paSSWOFd:
Usercanonly109inv{alocalhost
巨亘

* 增加自定义账号
* 添加管理员账号密码：rabbitmqctl add\_user admin admin

计算机生成了可选文字:
[root坦zhangyuopt]#rabbltmqctladd_useradmlnadmln
Addlnguser”。dmln"

* 分配账号角色：rabbitmqctl set\_user\_tags admin administrator

计算机生成了可选文字:
[root坦zhangyu
Settlngtags
opt]#rabbitmqctlset_user_tagsadmlnOdmln1Strator
foruser"admln''to[admlnistrator]

* 修改密码：rabbitmqctl change\_password admin 123456

计算机生成了可选文字:
[rootozhangyuopt]#robbltmqctlChonge_paSSwordadmln123456
Changlngpasswordforuser"admln"

* 查看用户列表：rabbitmqctl list\_users

计算机生成了可选文字:
【root@zhangyuopt】＃rabbltmqctl11St_userS
Llstlngusers
usertags
。dmln【administrotor】
guest[admlnlstratorl

* 使用新账号登录，成功界面

计算机生成了可选文字:
令令C合▲不安全｝192.168.137.137:15672/#Z
比Ra匕匕itMO
Ref陀Shed2019一11一1700:38:06
3.8.1
Edang21.3.8.9
ConneCtions
Ch己nne!S
EXch口nQes
Queues
Admin
O．☆昌e
Re,esh。verys,econds，一
v．。u。．h。st画万
Clusterrabbit@zh己ngyu
use。。dmonLo。ou:l
OVeFVieW
甲Tota.S
Queuedmessageslastminute?
CurrentIYidle
Messagerateslastminute?
Currentlyidle
G10balCOUntS7
}C。…，。一0{Ch一‘5:0}Ex·h一。一7}{Q一o)}c·，一o}
甲Node'
NameFIIedescriptors?
匣画巫巫回36
32768avai!able
Socketdescriptors?
0
29401available
Erlangprocesses
Memory?
Diskspace
Re弓et弓tat弓
+/-
420
1048576aVai!ab!e
76MIB
7.4GIB
UptimeInfo
42m59弓basiCdi弓C
1「55回国皿IA"nodes
794MIBhighwate翩a们M旧lowwatermark
卜Churn业atistics
卜port'dndcontexts
卜Exportde6nitions
卜Importdefinitions
HTTpApl
SerVerOOC亏
TUtori己15
CommunitySupport
ConlmunitySlack
CommercialSupport
Plugins
GitHUb
Chanoeloo
ov一“l
L。。。u:l
C卜anne!S:0
面足抓且口．
A!1nodes

* 管理界面标签页介绍
* overview：概览
* connections：无论生产者还是消费者，都需要与RabbitMQ建立连接后才可以完成消息的生产和消费，在这里可以查看连接情况
* channels：通道，建立连接后，会形成通道，消息的投递获取依赖通道。
* Exchanges：交换机，用来实现消息的路由
* Queues：队列，即消息队列，消息存放在队列中，等待消费，消费后被移除队列。
* 端口：
  + 5672：rabbitMq的编程语言客户端连接端口
  + 15672：rabbitMq管理界面端口
  + 25672：rabbitMq集群的端口

### 1.3.4. 卸载

* + rpm -qa | grep rabbitmq
  + rpm -e rabbitmq-server

## 1.4. 管理界面

### 1.4.1. 添加用户

如果不使用guest，我们也可以自己创建一个用户：

计算机生成了可选文字:
比R日匕b1tMO
Refreshed2019一11一1715:51:35一Refreshevery5seconds
3.8.1
Erlang21.3.8.9
V1rtualhost一A}l,
OVerVieW
ConneCtionS
ChannelS
Exchanges
Queues
心
C1usterrabbit@zhangyu
。se。。dm.nl遥口口
USeFS
AllUserS
Vil'tUalHOStS
F1lteF:
Regex?
2items,pagesizeupto
100
FeatureFlags
Ndltle
adlnin
Tags
administ旧t0F
CanaccessvirtualhostsHaspassword
Poli〔ies
NOaCCeSS
guest
admlnistrator
/
LimitS
?
C1uster
AddaUSel'
Username:zhangsan
.．…
PaSSW0rd
o;
……
(conflrm)
几。。．adminis七ra七or
．。甘J,
,etAdmin}Mollitoring}Policvmaker
Management
Impersonator一None
画亘，
目四

1、 超级管理员(administrator)

可登录管理控制台，可查看所有的信息，并且可以对用户，策略(policy)进行操作。

2、 监控者(monitoring)

可登录管理控制台，同时可以查看rabbitmq节点的相关信息(进程数，内存使用情况，磁盘使用情况等)

3、 策略制定者(policymaker)

可登录管理控制台, 同时可以对policy进行管理。但无法查看节点的相关信息(上图红框标识的部分)。

4、 普通管理者(management)

仅可登录管理控制台，无法看到节点信息，也无法对策略进行管理。

5、 其他

无法登录管理控制台，通常就是普通的生产者和消费者。

### 1.4.2. 创建Virtual Hosts

虚拟主机：类似于mysql中的database。他们都是以“/”开头

计算机生成了可选文字:
比Ra匕匕itMO
Refreshed2019一11一1716:07:32一Refreshevery5seconds
3.8.1
Erlang21.3.8.9
Virtualhost一All,
OVerVieW
ConneCtionS
ChannelS
Exchanges
Queues
Admin
龟
Clusterrabbit@zhangyu
。Se。。dm.nl班口口
Vi比U日1HOStS
USerS
甲Allvirtualhosts
Vi性UalHOStS
‘、＿「
．气
‘勺
Fi!ter:
一0Regex?
1item,pagesizeupto
冬
100
FeatureFlags
OVe四iew
Name
/
USe「S?
guest
St日te
「Unn川g
MessageS
ReadyUnackedTotal
NaNN日NNSN
NetworkMessage
FromclientToclientpublish
rates
+/-
Policies
deliver/get
Limits
甲AddanewviFtualhost
Cluster
Name:/Shopping
Description:
Tags:
吸
Addvi比ualhost
心

### 1.4.3. 设置权限

计算机生成了可选文字:
门匕匕｛t
Refreshed2019一11一1716:09:071Refreshevery5seconds
3.8.1
Erlang21.3.8.9
0Vel'VieW
ConneCtionS
ChannelS
Exchanges
Queues
心
V,rtualhost一All
Clusterrabbit@zhangyu
。Se。。dm.nl垃口口
V1rtua1H0sts
USerS
甲Allvirtualhosts
司，赢赢二二尸
Filter:
Regex?
2items,pagesizeupto
100
FeatureFlags
OVe四iew
PoliCieS
Nanle
State
USeFS?
guest
句dmin
rUnning
「Unning
MessagesNetworkMessagerates+/-
ReadyUllackedTotalFromclientToclientpublishdeliver/get
NSNNSNNSN
NSNNaNNaN
Lil,lits
点击这个虚拟主机
C!uster
Addane
画国

计算机生成了可选文字:
比Ra匕匕i
Refreshed2019一11一1716:12:57一Refreshevery5seconds,
3.8.1
Erlang21.3.8.9
V1rtualhostA}l
水l七
OVeFVieW
ConneCtionS
ChannelS
Exchanges
Queues
Admin
C1usterrabbit@zhangyu
。Se。。dm。。Logoutl
V1rtua1H0st:/5h0pp1ng
USerS
卜overview
Vi比UalHOStS
二立
peFmiSSIOnS
FeatureFlags
Currentpermissions
poliCieS
USeF
admin
ConfigureregexpWriteregexpReadregexp
L1mits
」曰．~~~~~~~~~~～气
{Clear{
C1USteF
Setpermisslon
Userzhangsan,
Configureregexp:
Writeregexp:
Readregexp:
给zhangsan用户分配虚拟
主机Ishopping的所有权限
Setpe而iSSIOn
Log。。tl

计算机生成了可选文字:
比Ra匕匕itMO
Refreshed2019一11一1716:16:30Refreshevery5seconds,
3.8.1
Erlang21.3.8.9
OVeFVieW
COnneCtionS
ChannelS
〔xchanges
Queues
Admin
v1rtuo．卜。st一A.．一
C1usterrabbit@zhangyu
use。。dm．。Logoutl
V1rtua1H0st:/5h0pp1ng
USe「S
OVeFVieW
VirtUalHOStS
peFmiSSionS
FeatureFlags
Currentpermissions
po!iCieS
USeF
admin
ConfigureregexpWriteregexpReadregexp
LimitS
(c.ea门
设置权限后当前权限列表中
多了一个zhangsan用户
C!USteF
Setpermission
Log。utl

# 2. 五种消息模型

RabbitMQ提供了6种消息模型，但是第6种其实是RPC，并不是MQ，因此不予学习。那么也就剩下5种。但是其实3、4、5这三种都属于订阅模型，只不过进行路由的方式不同。

计算机生成了可选文字:
O
,'H61!0WOFld!"
,IWo『kqueues
二l
吧二J
Thesimplestthingthatdoes
something
D1stributingtasksamong
WO「keFS
Sendingmessagestomany
COnSUmeFSatonCe
/\
卜、C
之）
Pyt.loll
－一馨
。明卜
月口口（D
祖口口（9
Pyt.lollJavak,Iby
C＃』avascriptGo
pHp
JavaR,IbyPHP
PytllollJavaRtlbyPHP
C＃」avascriptGo
C＃』avascrfptGo
．尺o"''""
口’op'"S
困卜RPC
ReceivingmessageS
selectively
Receivingmessagesbasedon
apattern
Remote
OCedUreC
implement百坳n
,.，猫艺蒸
口口口口口口口口口
Python，。v。
刃口口口口
Rolby
．月沙
．硕沙
!PHP
奄》．月沙二
、、‘
二
二同，竺．口口口口
．口，.
承笔淤
二口．.....．甲卜．
口口口口口．
七亚
PythollJOv。
加口口口口口
Rolby
．月诊
．硕沙
pHp
p
On
J己Vd
Rtlbvp城P
C#Javascript1Go
C#JavascriptGo
C＃』avascrfptGo
,.
二！l
口
口

* 我们通过一个demo工程来了解下RabbitMQ的工作方式，导入工程：

计算机生成了可选文字:
气尹
rdbbitmq一demo
》
。idea
气尹
srC
v,maIn
V〕ava
vcom.a七ulgU.rabbi七mq
》臼fanou七
》臼simPle
》臼spr:ng
》臼七。pic
》，u七11
》work
》．三re3OurCes
，亩test
V〕ava
丫com·a七gUigU.rabbi七mq.spring
己’Hqoemo
pom．洲口nl
_rabbl七mo一demo。lml
.'

* 依赖：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 <http://maven.apache.org/xsd/maven-4.0.0.xsd>">  <modelVersion>4.0.0</modelVersion>  <groupId>com.atguigu.rabbitmq</groupId>  <artifactId>rabbitmq-demo</artifactId>  <version>1.0-SNAPSHOT</version>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.9.RELEASE</version>  </parent>  <properties>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.3.2</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-amqp</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  </dependency>  </dependencies>  </project> |

* 我们抽取一个建立RabbitMQ连接的工具类，方便其他程序获取连接：

|  |
| --- |
| package com.atuigu.rabbitmq.util;  import com.rabbitmq.client.ConnectionFactory;  import com.rabbitmq.client.Connection;  import java.io.IOException;  public class ConnectionUtil {  /\*\*  \* 建立与RabbitMQ的连接  \*/  public static Connection getConnection() throws Exception {  //定义连接工厂  ConnectionFactory factory = new ConnectionFactory();  //设置服务地址  factory.setHost("192.168.137.137");  //端口  factory.setPort(5672);  //设置账号信息，用户名、密码、vhost  factory.setVirtualHost("/shopping");  factory.setUsername("zhangsan");  factory.setPassword("123456");  // 通过工程获取连接  Connection connection = factory.newConnection();  return connection;  }  public static void main(String[] args) throws Exception {  Connection con = ConnectionUtil.getConnection();  System.out.println(con);  con.close();  }  } |

## 2.1. 基本消息模型

官方介绍：

计算机生成了可选文字:
!ntF0dUCtion
RabbitMQ15anlessagebroker:itaCCeptSand
forwardsmeSSageS.YOuCanthinkaboutitasa
postoffiCe:whenyouputthemaitthatyouwdnt
p0Stingindp0Stbox.youcanbesurethatMr.
p0StmanwiLteventuattydetiverthemaittoyour
reCipient.Inthisanatogy.RabbitMQ15apostbox.
apostoffiCeondapostnlan.
Them司ordi什erencebetweenRabbitMQandthe
postoffice15thatitdoesn,tdeaLwithpaper'
insteaditaccepts.storesandforwardsbinary
btobsofdata一，刀‘55dges.

RabbitMQ是一个消息代理：它接受和转发消息。 你可以把它想象成一个邮局：当你把邮件放在邮箱里时，你可以确定邮差先生最终会把邮件发送给你的收件人。

在这个比喻中，RabbitMQ是邮政信箱，邮局和邮递员。

RabbitMQ与邮局的主要区别是它不处理纸张，而是接受，存储和转发数据消息的二进制数据块。

计算机生成了可选文字:
…
l

* P（producer/ publisher）：生产者，一个发送消息的用户应用程序。
* C（consumer）：消费者，消费和接收有类似的意思，消费者是一个主要用来等待接收消息的用户应用程序
* 队列（红色区域）：rabbitmq内部类似于邮箱的一个概念。
* 虽然消息流经rabbitmq和你的应用程序，但是它们只能存储在队列中。
* 队列只受主机的内存和磁盘限制，实质上是一个大的消息缓冲区。
* 许多生产者可以发送消息到一个队列，许多消费者可以尝试从一个队列接收数据。
* 总之：
  + 生产者将消息发送到队列，消费者从队列中获取消息，队列是存储消息的缓冲区。
  + 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。
  + 我们将详细介绍Java API中的一些细节，这是一个消息传递的“Hello World”。
  + 我们将调用我们的消息发布者（发送者）Send和我们的消息消费者（接收者）Recv。
  + 发布者将连接到RabbitMQ，发送一条消息，然后退出。

### 2.1.1. 生产者发送消息

|  |
| --- |
| package com.atuigu.rabbitmq.simple;  import com.atuigu.rabbitmq.util.ConnectionUtil;  import com.rabbitmq.client.Channel;  import com.rabbitmq.client.Connection;  public class Send {  private final static String QUEUE\_NAME = "simple\_queue";  public static void main(String[] argv) throws Exception {  // 获取到连接  Connection connection = ConnectionUtil.getConnection();  // 从连接中创建通道，使用通道才能完成消息相关的操作  Channel channel = connection.createChannel();  // 声明（创建）队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);  // 消息内容  String message = "Hello World!";  // 向指定的队列中发送消息  channel.basicPublish("", QUEUE\_NAME, null, message.getBytes());  System.out.println(" [x] Sent '" + message + "'");  //关闭通道和连接  channel.close();  connection.close();  }  } |

 API：

|  |
| --- |
| / \*\*    \* 声明队列    \* @param queue队列名称    \* @param durable 如果我们声明一个持久队列，则为true（该队列将在服务器重启后继续存在）    \* @param Exclusive 如果我们声明一个排他队列，则为true（仅限此连接）    \* @param autoDelete 如果我们声明一个自动删除队列，则为true（服务器将在不再使用它时将其删除）    \* @param arguments 参数队列的其他属性（构造参数）    \* @return一个声明确认方法来指示队列已成功声明    \* /  Queue.DeclareOk queueDeclare(String queue, boolean durable, boolean exclusive, boolean autoDelete,Map<String, Object> arguments) throws IOException; |

控制台：

计算机生成了可选文字:
卜l,
C八Java\jdkl.8.0_45\bin\ja甲a…
[x]Sent'HelloWorld!'
Processfinishedwithexitcodeo

### 2.1.2. 管理工具中查看消息

进入队列页面，可以看到新建了一个队列：simple\_queue

计算机生成了可选文字:
OVeFVieW
COnneCtionS
ChannelS
Exchanges
QueueS
Admin
Userzhangsan
〔言器〕
QueueS
,A11queues(1)
Pa91nation
Pagel,ofl一Filter:
口Regex?
D1splaying1item,pagesizeupto:
100
OVerVieW
MeSSageS
ReadyUnackedTotal
101
MeSSagerates
incomingdeliver/getack
0.00/s
+/-
VirtualhostNameType
/shoppingsimple_queueclassic
Fe己tUreSState
idle

点击队列名称，进入详情页，可以查看消息：

计算机生成了可选文字:
OVeFVieW
COnneCtionS
ChannelS
Exchanges
Queu
Admin
卜Publishmessage
Getmessages
Warn1ng:gettingmessagesfromaqueue15adestructiveaction.?
ACkMOde:
Encoding:
MessageS:
Nackmessagerequeuetrue,
Autostring/base64,?
工
GetMessage(s)
MeSSagel
Theserverreported0messagesremaining.
Exchange
RoutingKey
Redelivered
Properties
Payload
12匕ytes
Encoding:string
(AMQPdefault)
He110World!

在控制台查看消息并不会将消息消费，所以消息还在。

### 2.1.3. 消费者获取消息

|  |
| --- |
| package com.atuigu.rabbitmq.simple;  import com.atuigu.rabbitmq.util.ConnectionUtil;  import com.rabbitmq.client.\*;  import java.io.IOException;  /\*\*  \* 消费者  \*/  public class Recv {  private final static String QUEUE\_NAME = "simple\_queue";  public static void main(String[] argv) throws Exception {  // 获取到连接  Connection connection = ConnectionUtil.getConnection();  // 创建通道  Channel channel = connection.createChannel();  // 声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);  // 定义队列的消费者  DefaultConsumer consumer = new DefaultConsumer(channel) {  // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties,byte[] body) throws IOException {  // body 即消息体  String msg = new String(body);  System.out.println(" [x] received : " + msg + "!");  }  };  // 监听队列，第二个参数：是否自动进行消息确认。  channel.basicConsume(QUEUE\_NAME, true, consumer);  }  } |

控制台：

计算机生成了可选文字:
偏｝,
C八Java\jdkl.8.0_45\bin\ja,a…
[x〕received:HelloWorld!!
备弓
."

这个时候，队列中的消息就没了：

计算机生成了可选文字:
OVeFVieW
ConneCtionS
ChannelS
Exchanges
UeUeS
Admin
QueUeS
卜AI1queues(1)
OVerVieW
VirtualhostNameType
/shoppingsimple_queueclassic
Fe己tUresState
Messages
ReadyUnackedTotal
000
Messagerates
incomingdeliver/get
0.00/50.00/s
aCk
idle
0.00/s

我们发现，消费者已经获取了消息，但是程序没有停止，一直在监听队列中是否有新的消息。一旦有新的消息进入队列，就会立即打印.

### 2.1.4. 消息确认机制（ACK）

* 通过刚才的案例可以看出，消息一旦被消费者接收，队列中的消息就会被删除。
* 那么问题来了：RabbitMQ怎么知道消息被接收了呢？
* 如果消费者领取消息后，还没执行操作就挂掉了呢？或者抛出了异常？消息消费失败，但是RabbitMQ无从得知，这样消息就丢失了！
* 因此，RabbitMQ有一个ACK机制。当消费者获取消息后，会向RabbitMQ发送回执ACK，告知消息已经被接收。
* 不过这种回执ACK分两种情况：
  + 自动ACK：消息一旦被接收，消费者自动发送ACK
  + 手动ACK：消息接收后，不会发送ACK，需要手动调用
* 大家觉得哪种更好呢？
* 这需要看消息的重要性：
  + 如果消息不太重要，丢失也没有影响，那么自动ACK会比较方便
  + 如果消息非常重要，不容丢失。那么最好在消费完成后手动ACK，否则接收消息后就自动ACK，RabbitMQ就会把消息从队列中删除。如果此时消费者宕机，那么消息就丢失了。
* 我们之前的测试都是自动ACK的，如果要手动ACK，需要改动我们的代码：

|  |
| --- |
| public class Recv2 {     private final static String QUEUE\_NAME = "simple\_queue";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 创建通道         final Channel channel = connection.createChannel();         // 声明队列         channel.queueDeclare(QUEUE\_NAME, false, false, false, null);         // 定义队列的消费者         DefaultConsumer consumer = new DefaultConsumer(channel) {             // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用             @Override             public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties,                     byte[] body) throws IOException {                 // body 即消息体                 String msg = new String(body);  //int i = 1/0;                 System.out.println(" [x] received : " + msg + "!");                 // 手动进行ACK                 channel.basicAck(envelope.getDeliveryTag(), false);            }        };         // 监听队列，第二个参数false，手动进行ACK         channel.basicConsume(QUEUE\_NAME, false, consumer);    }  } |

注意到最后一行代码：

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

如果第二个参数为true，则会自动进行ACK；如果为false，则需要手动ACK。方法的声明：

计算机生成了可选文字:
／承＊
承Startanon一olocal,non一excl:15生recons：切er
承aserver一generatedconsL如eITag.
冰鱼丝旦鱼aueuet力ena功eo了t力eQ:Je:Je
wlth
日工日口SUtOACkt工LIeifthes户工r户1ShOLJldCOI]SI
acknowle心edoncedeli、·ered‘介lse工ftheser、·er
户厂口eSS日ges
劝ouldeXPect
e却1ic工tacknowle心e田ents
承鱼丝丝ca11ba比谧万州艺街几拐布弱麟百南sL皿ej’刁7j’目拐
承鱼旦卫塑theconsL切erTaggeneratedbytheser,er
承鱼五些丝jat'a．拓．IOEXc即non汀an~or招enco明tered
承鱼丝c。瓜rabb立t.Q.cl互朗t～姗职Baslc.ConsL皿e
承鱼丝conLrabb立tIDQ.cl工朗t.．姗护Ba别c.ConsL皿eok
水鱼丝二baslcCons：迎estrlng.boolean,String.boolean,boolean，价p,Cons姗er
叮
String
basicConsLuDe(String叫eue,1booleanautoAck,1ConsLuDercallback)thro节5IOExcept立on;

**2.1.4.1. 自动ACK存在的问题**

修改消费者，添加异常，如下：

计算机生成了可选文字:
@override
pub1icvo立dhandleDelivery(String
byte［〕
刀b。d,．即消，急体
Str,ngmsg二ne,String(body);
consLunerTag,Envelopeenvelope，枷QP.BasicPropertiesproperties,
body)thro,5IOExcepti。n{
主nti二1/0
:.'stem-out.println（甲［:]rece主ved：禅＋msg＋禅！禅）;
}

生产者不做任何修改，直接运行，消息发送成功：

计算机生成了可选文字:
QueueS
A11queues(1)
OVerVieW
V1rtualhostNameType
/shoppingsimple_queueclassiC
Fe己tUFeSState
FUnn1ng
Messages
ReadYUnackedTotal
101
Messagerates+/-
incomingdeliver/getack
0.00/50.00/50.00/s

运行消费者，程序抛出异常。但是消息依然被消费：

计算机生成了可选文字:
全C:\Ja,a\jdkl.8.0_45\bin\java…
备20:19:38.786〔pool一1一thread一4]ERRORcom.rabbit叫．client.i呻1.ForgivingExceptionHandler
弓java.lang.ArithmeticException:/b:.,zero
回。t。。二。tuigu.rabbit、．siole.Recv$1.handleoeli:ery（处红口立组丈边）
香atco二”bb主tmq.chent.i即1.cons~l-D始Patcher$5.rLul（一一上，,
亩I田atco二rabbit叫·client·i叩l·Constunerworkservice$WorkPoolRLumable.rm(
atjava.lang.T址ead.r明（TI二ead.lava:745)
一Consumercom.atuigu.rabbit叫．51呻le.
偏．“目旧
馨
14)<2internalcalls>

管理界面：

计算机生成了可选文字:
QueueS
卜A11queues(1)
OVerVieW
V1rtualhostNameType
/shoppingsimple_queueclassic
Fe己tUreS
State
dle
Messages
ReadyUnackedTotal
000
Messagerates+/-
incomingdeliver/getack
0.00/50.20/50.00/s

**2.1.4.2. 演示手动ACK**

修改消费者，把自动改成手动（去掉之前制造的异常）

计算机生成了可选文字:
刀定义队列的消费者
DefaultCons~rcons~r二ne贯DefaultCons~r(channel){
刀获取消息、，并且处理，这个方法类似事件监听，如果有消，自、的时候，会被自动调用
@override
publ立cvoidhandleDel主very(StringconstunerTag,Envelopeenvelope,AMQP.BasicPropertiesproperties,
byte［〕bod:..)thro份5loException{
刀bod,．即消，息体
Strln
口口口口口口口口口
//inr
憋沪气S'r'n"b。‘y'
不几em-out．页拓nn（冲［x]received：牟＋msg＋命！即）
}
};
刀监听队列，第二个参数：是否自动进行消息确科、。
chann。：.bas＊。C。nstune（。二吸～，暇思岔袱嚣1。。nstuner)

生产者不变，再次运行：

计算机生成了可选文字:
QueueS
甲Allqueues
Fi!ter:
Regex〔，
OVe四iew
VirtU己1h0St
N己n】e
FeatUreS
St己te
Ready
Messages
Un己Cked
Total
Messagerates
incomingdeliver1oet
0.00150.00/5
己仁k
/leyou
simple_queue
idle

运行消费者

计算机生成了可选文字:
偏｝,
C八Java\jdkl.8.0_45\bin\ja,a…
[x〕received:HelloWorld!!
备弓
."

但是，查看管理界面，发现：

计算机生成了可选文字:
QueueS
卜A11queues(1)
OVerVleW
VirtualhostNameType
/shoppingsimple_queueclassic
Fe己tUres
State
d!e
Messages
ReadYUnackedTotal
011
Messagerates
incomingdeliver/get
0.00/50.00/s
+/-
aCk
0.00/s

停掉消费者的程序，发现：

计算机生成了可选文字:
QueueS
A11queues(1)
OVerVieW
V1rtualhostNameType
/shoppingsimple_queueclassiC
Fe己tUreS
State
id!e
Messages
ReadyUnackedTotal
101
Messagerates+/-
incomingdeliver/getack
0.00/50.00/50.00/s

这是因为虽然我们设置了手动ACK，但是代码中并没有进行消息确认！所以消息并未被真正消费掉。

当我们关掉这个消费者，消息的状态再次称为Ready

修改代码手动ACK：

计算机生成了可选文字:
刀定义队列的消费者
DefaultConsumercons~r二ne,DefaultConsumer(channel){
刀获取消，自、，并且处理，这个方法类似事件监听，如果有消息、的时候，会被自动调用
@override
pub1ic甲0idhandleDeliver:.'(StringconstunerTag,Envelopeenvelope，枷QP.BasicPropert立espropert立es,
byte[]bod:..)t址。，5loException{
刀b。d-T．即消，息体
String
口．......
intl
msg二ne,String(body)
互丁丁1
:stem.out.prlntln（甲［x]received：尸＋二g＋尸！#)
/／手动进行ACK
口口曰口口口口口口口口口口口口口口口口口口口口口口口口口口口口口．...．口口口口口月口
channe1.basicAck(envelope.getDeliver:..Tag()
multiple:false);
}
};
刀监听队列，第二个参数介1，。，手动进厅：一
channe’·bas,cConsume(o二吸～，暇蕊岔氛二
COflSLllller

执行：

计算机生成了可选文字:
QueueS
卜A11queues(1)
OVerVieW
V1rtualhostNameType
/shoppingsimple_queueclassic
Fe己tUreS
State
dle
Messages
ReadyUnackedTotal
000
Messagerates+/-
incomingdeliver/getack
0.00/50.20/50.00/s

消息消费成功！

## 2.2. work消息模型

工作队列或者竞争消费者模式

计算机生成了可选文字:
2琪IOrkqueues
Distributingtasksanlong
\Vorkers(the
!）亡rsDattern)
」
｝』一
几，
},)
\l/
l’认
。口口口．
/
.
刁叭、
．了’一厂一＼
．泛岁
口口口口口口

在第一篇教程中，我们编写了一个程序，从一个命名队列中发送并接受消息。在这里，我们将创建一个工作队列，在多个工作者之间分配耗时任务。

工作队列，又称任务队列。主要思想就是避免执行资源密集型任务时，必须等待它执行完成。相反我们稍后完成任务，我们将任务封装为消息并将其发送到队列。 在后台运行的工作进程将获取任务并最终执行作业。当你运行许多消费者时，任务将在他们之间共享，但是**一个消息只能被一个消费者获取**。

这个概念在Web应用程序中特别有用，因为在短的HTTP请求窗口中无法处理复杂的任务。

接下来我们来模拟这个流程：

* P：生产者：任务的发布者​
* C1：消费者，领取任务并且完成任务，假设完成速度较快​
* C2：消费者2：领取任务并完成任务，假设完成速度慢

面试题：避免消息堆积？

1）采用workqueue，多个消费者监听同一队列。

2）接收到消息以后，而是通过线程池，异步消费。

### 2.2.1. 生产者

生产者与案例1中的几乎一样：

|  |
| --- |
| public class Send {     private final static String QUEUE\_NAME = "test\_work\_queue";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明队列         channel.queueDeclare(QUEUE\_NAME, false, false, false, null);         // 循环发布任务         for (int i = 0; i < 50; i++) {             // 消息内容             String message = "task .. " + i;             channel.basicPublish("", QUEUE\_NAME, null, message.getBytes());             System.out.println(" [x] Sent '" + message + "'");​        }         // 关闭通道和连接         channel.close();         connection.close();    }  } |

不过这里我们是循环发送50条消息。

### 2.2.2. 消费者1

|  |
| --- |
| // 消费者1  public class Recv {  private final static String QUEUE\_NAME = "test\_work\_queue";  public static void main(String[] argv) throws Exception {  // 获取到连接  Connection connection = ConnectionUtil.getConnection();  // 获取通道  final Channel channel = connection.createChannel();  // 声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);  // 定义队列的消费者  DefaultConsumer consumer = new DefaultConsumer(channel) {  // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties,  byte[] body) throws IOException {  // body 即消息体  String msg = new String(body);  System.out.println(" [消费者1] received : " + msg + "!");  try {  // 模拟完成任务的耗时：1000ms  Thread.sleep(1000);  } catch (InterruptedException e) {  }  // 手动ACK  channel.basicAck(envelope.getDeliveryTag(), false);  }  };  // 监听队列。  channel.basicConsume(QUEUE\_NAME, false, consumer);  }  } |

### 2.2.3. 消费者2

|  |
| --- |
| //消费者2  public class Recv2 {  private final static String QUEUE\_NAME = "test\_work\_queue";  public static void main(String[] argv) throws Exception {  // 获取到连接  Connection connection = ConnectionUtil.getConnection();  // 获取通道  final Channel channel = connection.createChannel();  // 声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);  // 定义队列的消费者  DefaultConsumer consumer = new DefaultConsumer(channel) {  // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties,  byte[] body) throws IOException {  // body 即消息体  String msg = new String(body);  System.out.println(" [消费者2] received : " + msg + "!");  try {  // 模拟完成任务的耗时：200ms  Thread.sleep(200);  } catch (InterruptedException e) {  }  // 手动ACK  channel.basicAck(envelope.getDeliveryTag(), false);  }  };  // 监听队列。  channel.basicConsume(QUEUE\_NAME, false, consumer);  }  } |

与消费者1基本类似，就是没有设置消费耗时时间。

这里是模拟有些消费者快，有些比较慢。

接下来，两个消费者一同启动，然后发送50条消息：

|  |  |
| --- | --- |
| 计算机生成了可选文字: RUn: 肋cvzl，肋cv(1)l呀send(1) tIC八Java\jdkl.8.0_45\bin\java 。一叶肖费者1〕rece,ved:task· 弓叶肖费都］received:ta,k· 画叶肖费都〕recelved:task· 香．肖费都〕reCelved:task· 市［争肖费都］reCe,ved:task· 叶肖费都〕rece主ved:task· 叶肖费者1〕received:task· 叶肖费者1]received:task· 叶肖费者1]received:task· 叶肖费者1]received:task· 叶肖费者1〕received:task· 叶肖费者1〕rece主ved:task· 叶肖费者1〕：ece主ved:task· 叶肖费都〕rece主ved:task· 叶肖费者1〕received:task· 叶肖费者1]received:task· 叶肖费者1〕received:task· 叶肖费都〕received:task· 叶肖费者1〕received:task· 叶肖费者1〕received:task· 叶肖费都〕rece主ved:task· 叶肖费者1]received:task· 叶肖费者1]received:task· 叶肖费者1〕received:task· 叶肖费者1〕received:task· 一」avaagent 1! 3! 5! Ll'OJ 偏．”口旧 些 11! 13! 15! 17! 19! 21! 23! 25! 27! 29! 31! 33! 35! 37! 39! 41! 43! 45! 47! 49! 广X, | 计算机生成了可选文字: RUn: 巴垂口『R一（，亚Se·“,,. C八Java\Jdkl.8.0_45\bin\java 叶肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. ．肖费者2〕received:task. 。肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2]received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:ta:k. ．肖费者2〕received:task. ．肖费者2〕rece:ved:task. ．肖费者皿1rece,ved:task. ．肖费者21received:task. 氏肖费者21received:task. ．肖费者2〕received:task. ，肖费者2〕received:task. 叶肖费者2〕received:task. 叶肖费者2〕received:task. 一）avaagent 0! 伪， 4! 6! 8! 10! 12! 14! 16! ，备 偏 弓已香一画 .“口旧 馨 户X, OOn甘01连占么〕OUnU口1连‘民U八己n甘勺工A占叹UOU 11臼1口1勺一勺一口一Q〕八jQ口Q〕八J连占连占连占连‘连山 巨皿口 |

可以发现，两个消费者各自消费了25条消息，而且各不相同，这就实现了任务的分发。

### 2.2.4. 能者多劳

* 刚才的实现有问题吗？
  + 消费者1比消费者2的效率要低，一次任务的耗时较长
  + 然而两人最终消费的消息数量是一样的
  + 消费者2大量时间处于空闲状态，消费者1一直忙碌
* 现在的状态属于是把任务平均分配，正确的做法应该是消费越快的人，消费的越多。
* 怎么实现呢？
  + 我们可以使用basicQos方法和prefetchCount = 1设置。
  + 这告诉RabbitMQ一次不要向工作人员发送多于一条消息。
  + 或者换句话说，不要向工作人员发送新消息，直到它处理并确认了前一个消息。
  + 相反，它会将其分派给不是仍然忙碌的下一个工作人员。

计算机生成了可选文字:
刀声明队列
channe1．叫eueDeclare(O泥呱－刀摊卫百，durable:false
exclusive:false
autoDelete:false,arguments:null);
〔Oq甘八〕，上
，人1人乃一乃1
DefaultConsLuDer
刀获取消，自、，
consLuner二ne,DefaultConsumer(channel){
并且处理，这个方法类似事件监听，如果有消息、的时候，会被自动调用
勺白Q〕月份
勺白乃1伪〕

再次测试：

|  |  |
| --- | --- |
| 计算机生成了可选文字: ReCVZ ，肋cv(1)l下send(1) C八Java\jdkl.8.0_45\bin\java ［消费者1〕received:task. ［消费者1〕rec。ived:ta:k. 〔消费者1〕received:task. 仁消费者1〕rece:ved:task. 仁消费者1]received:task. ［消费者1]rece主ved:task. 一」avaagent 0! 4! 11! 19! 30! 44! | 计算机生成了可选文字: 。RecvZ）呼Recv｛卫一Send(1)__ C八Java\jdkl.8.0_45\bin\java一javaagent ［消费者2]received:task二1! ［消费者2]received:ta:k二2! ［消费者2〕received:task二3! ［消费者2〕received:task二5! 氏肖费者2]received:task二6! ．肖费者2]received:task二7! 氏肖费者2]received:task二8! ［消费者2]received:task二9! ［消费者2]received:task二10! ［消费者2]received:task二12! ［消费者2〕received:task二13! 氏肖费者2〕rece主ved:task二14! 氏肖费者2〕rece主ved:task二15! 叶肖费者2〕received:task二16! 叶肖费者2〕received:task二17! ［消费者2]received:task二18! ［消费者2]received:task二20! ［消费者2]received:task二21! ［消费者2〕received:task二22! ．肖费者2〕received:task二23! ．肖费者2]received:task二24! ．肖费者2]received:task二25! ［消费都〕received:task二26! 叶肖费都］received:task二27! ［消费者2]received:task二28! ［消费者2]received:task二29! 〔消费者2〕received:task二31! ［消费者2〕received:task二32! 氏肖费者2]received:task二33! 叶肖费都］received:task二34! 叶肖费者2〕received:task二35! ［消费都〕received:task二36! ［消费者2]received:task二37! ［消费者2]received:task二38! ．肖费者2〕received:task二39! ．肖费者2]rece主ved:task二40! ．肖费者2〕received:task二41! 叶肖费者2]received:task二42! 叶肖费者2〕received:task二43! ［消费者2]received:task二45! 〔消费者2]received:task二46! ［消费者2]received:task二47! ［消费者2〕received:task二48! 氏肖费者2〕rece主ved:task二49! |

## 2.3. 订阅模型分类

在之前的模式中，我们创建了一个工作队列。 工作队列背后的假设是：每个任务只被传递给一个工作人员。

在这一部分，我们将做一些完全不同的事情 - 我们将会传递一个信息给多个消费者。 这种模式被称为“发布/订阅”。

订阅模型示意图：

计算机生成了可选文字:
amq.gen一RQ6…
amq.gen一58…
口口口．
口口口口

解读：

1、1个生产者，多个消费者

2、每一个消费者都有自己的一个队列

3、生产者没有将消息直接发送到队列，而是发送到了交换机

4、每个队列都要绑定到交换机

5、生产者发送的消息，经过交换机到达队列，实现一个消息被多个消费者获取的目的

X（Exchanges）：

交换机一方面：接收生产者发送的消息。

另一方面：知道如何处理消息，例如递交给某个特别队列、递交给所有队列、或是将消息丢弃。到底如何操作，取决于Exchange的类型。

**Exchange（交换机）只负责转发消息，不具备存储消息的能力**，因此如果没有任何队列与Exchange绑定，或者没有符合路由规则的队列，那么消息会丢失！

Exchange类型有以下几种：

* Fanout：广播，将消息交给所有绑定到交换机的队列
* Direct：定向，把消息交给符合指定routing key 的队列
* Topic：通配符，把消息交给符合routing pattern（路由模式） 的队列

## 2.4. 订阅模型-Fanout

Fanout，也称为广播。

流程图：

计算机生成了可选文字:
amq.gen一RQ6…
①
口口口口
1.....

在广播模式下，消息发送流程是这样的：

1） 可以有多个消费者

2） 每个**消费者有自己的queue**（队列）

3） 每个**队列都要绑定到Exchange**（交换机）

4） **生产者发送的消息，只能发送到交换机**，交换机来决定要发给哪个队列，生产者无法决定。

5） 交换机把消息发送给绑定过的所有队列

6） 队列的消费者都能拿到消息。实现一条消息被多个消费者消费

### 2.4.1. 生产者

两个变化：

1） 声明Exchange，不再声明Queue

2） 发送消息到Exchange，不再发送到Queue

|  |
| --- |
| public class Send {     private final static String EXCHANGE\_NAME = "fanout\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明exchange，指定类型为fanout         channel.exchangeDeclare(EXCHANGE\_NAME, "fanout");         // 消息内容         String message = "Hello everyone";         // 发布消息到Exchange         channel.basicPublish(EXCHANGE\_NAME, "", null, message.getBytes());         System.out.println(" [生产者] Sent '" + message + "'");         channel.close();         connection.close();    }  } |

### 2.4.2. 消费者1

|  |
| --- |
| public class Recv {     private final static String QUEUE\_NAME = "fanout\_exchange\_queue\_1";     private final static String EXCHANGE\_NAME = "fanout\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明队列         channel.queueDeclare(QUEUE\_NAME, false, false, false, null);         // 绑定队列到交换机         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "");         // 定义队列的消费者         DefaultConsumer consumer = new DefaultConsumer(channel) {             // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用             @Override             public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties, byte[] body) throws IOException {                 // body 即消息体                 String msg = new String(body);                 System.out.println(" [消费者1] received : " + msg + "!");            }        };         // 监听队列，自动返回完成         channel.basicConsume(QUEUE\_NAME, true, consumer);    }  } |

### 2.4.3. 消费者2

|  |
| --- |
| public class Recv2 {     private final static String QUEUE\_NAME = "fanout\_exchange\_queue\_2";     private final static String EXCHANGE\_NAME = "fanout\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明队列         channel.queueDeclare(QUEUE\_NAME, false, false, false, null);         // 绑定队列到交换机         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "");         // 定义队列的消费者         DefaultConsumer consumer = new DefaultConsumer(channel) {             // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用             @Override             public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties,                     byte[] body) throws IOException {                 // body 即消息体                 String msg = new String(body);                 System.out.println(" [消费者2] received : " + msg + "!");            }        };         // 监听队列，手动返回完成         channel.basicConsume(QUEUE\_NAME, true, consumer);    }  } |

### 2.4.4. 测试

**我们应该先启动生产者，否则，先启动消费者时，由于要绑定交换机，此时，交换机并不存在所以会报错。**

我们运行两个消费者，然后发送1条消息：

计算机生成了可选文字:
C八Java\jdkl.
［生产者］Sent
8.0_45\b主n\java…
He110everyone
Processfinishedwithexltcodeo

计算机生成了可选文字:
C八Ja,a\jdkl.8.0_45\bin\ja,a…
氏肖费者1]received:Hell。ever:one!

计算机生成了可选文字:
C八Java\jdkl.8.0_45\bin\java…
氏肖费者2]received:Hell。ever:one!

## 2.5. 订阅模型-Direct

* 有选择性的接收消息
* 在订阅模式中，生产者发布消息，所有消费者都可以获取所有消息。
* 在路由模式中，我们将添加一个功能 - 我们将只能订阅一部分消息。

例如，我们只能将重要的错误消息引导到日志文件（以节省磁盘空间），同时仍然能够在控制台上打印所有日志消息。

* 但是，在某些场景下，我们希望不同的消息被不同的队列消费。这时就要用到Direct类型的Exchange。
* 在Direct模型下，队列与交换机的绑定，不能是任意绑定了，而是要指定一个RoutingKey（路由key）
* 消息的发送方在向Exchange发送消息时，也必须指定消息的routing key。

计算机生成了可选文字:
amqp.gen一Sgb…
eFrOF
C.
type=dire吐
info
amqp.gen一Agl…
P
eFrOF
C:
Warnlng
．口口．
自口．

* P：生产者，向Exchange发送消息，发送消息时，会指定一个routing key。
* X：Exchange（交换机），接收生产者的消息，然后把消息递交给 与routing key完全匹配的队列
* C1：消费者，其所在队列指定了需要routing key 为 error 的消息
* C2：消费者，其所在队列指定了需要routing key 为 info、error、warning 的消息

### 2.5.1. 生产者

此处我们模拟商品的增删改，发送消息的RoutingKey分别是：insert、update、delete

|  |
| --- |
| public class Send {     private final static String EXCHANGE\_NAME = "direct\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明exchange，指定类型为direct         channel.exchangeDeclare(EXCHANGE\_NAME, "**direct**");         // 消息内容         String message = "商品新增了， id = 1001";         // 发送消息，并且指定routing key 为：insert ,代表新增商品         channel.basicPublish(EXCHANGE\_NAME, "**insert**", null, message.getBytes());         System.out.println(" [商品服务：] Sent '" + message + "'");         channel.close();         connection.close();    }  } |

​

### 2.5.2. 消费者1

我们此处假设消费者1只接收两种类型的消息：更新商品和删除商品。

|  |
| --- |
| public class Recv {     private final static String QUEUE\_NAME = "direct\_exchange\_queue\_1";     private final static String EXCHANGE\_NAME = "direct\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明队列         channel.queueDeclare(QUEUE\_NAME, false, false, false, null);         // 绑定队列到交换机，同时指定需要订阅的routing key。假设此处需要update和delete消息         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "**update**");         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "**delete**");         // 定义队列的消费者         DefaultConsumer consumer = new DefaultConsumer(channel) {             // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用             @Override             public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties,                     byte[] body) throws IOException {                 // body 即消息体                 String msg = new String(body);                 System.out.println(" [消费者1] received : " + msg + "!");            }        };         // 监听队列，自动ACK         channel.basicConsume(QUEUE\_NAME, true, consumer);    }  } |

### 2.5.3. 消费者2

我们此处假设消费者2接收所有类型的消息：新增商品，更新商品和删除商品。

|  |
| --- |
| public class Recv2 {     private final static String QUEUE\_NAME = "direct\_exchange\_queue\_2";     private final static String EXCHANGE\_NAME = "direct\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明队列         channel.queueDeclare(QUEUE\_NAME, false, false, false, null);         // 绑定队列到交换机，同时指定需要订阅的routing key。订阅 insert、update、delete         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "**insert**");         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "**update**");         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "**delete**");         // 定义队列的消费者         DefaultConsumer consumer = new DefaultConsumer(channel) {             // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用             @Override             public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties,                     byte[] body) throws IOException {                 // body 即消息体                 String msg = new String(body);                 System.out.println(" [消费者2] received : " + msg + "!");            }        };         // 监听队列，自动ACK         channel.basicConsume(QUEUE\_NAME, true, consumer);    }  } |

### 2.5.4. 测试

我们分别发送增、删、改的RoutingKey，发现结果：

计算机生成了可选文字:
C八Java\jdkl.8.0_45\bin\java
〔消费者1]reoeived：商品删除了
〔消费者1]received：商品更新了
ld二
id二
1001!
1001!

计算机生成了可选文字:
C八Ja,a\jdkl.8.0_45\bin\java…
［消费者2]received：商品删除了，id二1001!
［消费者2]received：商品新增了，id二1001!
［消费者2〕received：商品更新了，id二1001!

## 2.6. 订阅模型-Topic

* Topic类型的Exchange与Direct相比，都是可以根据RoutingKey把消息路由到不同的队列。只不过Topic类型Exchange可以让队列在绑定Routing key 的时候使用通配符！
* Routingkey 一般都是有一个或多个单词组成，多个单词之间以”.”分割，例如： item.insert
* 通配符规则：
  + `#`：匹配**0或多个词**​（含零个）
  + `\*`：匹配不多不少恰好1个词（不含零个）
* 举例：
  + `audit.#`：能够匹配`audit.irs.corporate` 或者 `audit.irs`​
  + `audit.\*`：只能匹配`audit.irs`

计算机生成了可选文字:
，人
Q
type=topic*.orange.*
‘一

* 在这个例子中，我们将发送所有描述动物的消息。消息将使用由三个字（两个点）组成的routing key发送。路由关键字中的第一个单词将描述速度，第二个颜色和第三个种类：“**<speed>.<color>.<species>**”。
* 我们创建三个绑定：

Q1绑定了绑定键“\* .orange.\**”，Q2绑定了“\**.\*.rabbit”和“lazy.＃”。

* Q1匹配所有的橙色动物。
* Q2匹配关于兔子以及懒惰动物的消息。
* 练习，生产者发送如下消息，会进入那个队列：
  + quick.orange.rabbit
  + lazy.orange.elephant
  + quick.orange.fox
  + lazy.pink.rabbit
  + quick.brown.fox
  + quick.orange.male.rabbit
  + orange

### 2.6.1. 生产者

使用topic类型的Exchange，发送消息的routing key有3种： item.isnert、item.update、item.delete：

|  |
| --- |
| public class Send {     private final static String EXCHANGE\_NAME = "topic\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明exchange，指定类型为topic         channel.exchangeDeclare(EXCHANGE\_NAME, "topic");         // 消息内容         String message = "**新增商品** : id = 1001";         // 发送消息，并且指定routing key 为：insert ,代表新增商品         channel.basicPublish(EXCHANGE\_NAME, "**item.insert**", null, message.getBytes());         System.out.println(" [商品服务：] Sent '" + message + "'");         channel.close();         connection.close();    }  } |

### 2.6.2. 消费者1

我们此处假设消费者1只接收两种类型的消息：更新商品和删除商品

|  |
| --- |
| public class Recv {     private final static String QUEUE\_NAME = "topic\_exchange\_queue\_1";     private final static String EXCHANGE\_NAME = "topic\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明队列         channel.queueDeclare(QUEUE\_NAME, false, false, false, null);         // 绑定队列到交换机，同时指定需要订阅的routing key。需要 update、delete         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "**item.update**");         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "**item.delete**");         // 定义队列的消费者         DefaultConsumer consumer = new DefaultConsumer(channel) {             // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用             @Override             public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties,                     byte[] body) throws IOException {                 // body 即消息体                 String msg = new String(body);                 System.out.println(" [消费者1] received : " + msg + "!");            }        };         // 监听队列，自动ACK         channel.basicConsume(QUEUE\_NAME, true, consumer);    }  } |

### 2.6.3. 消费者2

我们此处假设消费者2接收所有类型的消息：新增商品，更新商品和删除商品。

|  |
| --- |
| /\*\*  \* 消费者2  \*/  public class Recv2 {     private final static String QUEUE\_NAME = "topic\_exchange\_queue\_2";     private final static String EXCHANGE\_NAME = "topic\_exchange\_test";     public static void main(String[] argv) throws Exception {         // 获取到连接         Connection connection = ConnectionUtil.getConnection();         // 获取通道         Channel channel = connection.createChannel();         // 声明队列         channel.queueDeclare(QUEUE\_NAME, false, false, false, null);         // 绑定队列到交换机，同时指定需要订阅的routing key。订阅 insert、update、delete         channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "**item.\***");         // 定义队列的消费者         DefaultConsumer consumer = new DefaultConsumer(channel) {             // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用             @Override             public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties,                     byte[] body) throws IOException {                 // body 即消息体                 String msg = new String(body);                 System.out.println(" [消费者2] received : " + msg + "!");            }        };         // 监听队列，自动ACK         channel.basicConsume(QUEUE\_NAME, true, consumer);    }  } |

### 2.6.4.测试

计算机生成了可选文字:
C八Ja,a\jdkl.8.0_45\b主n\ja,a…
〔消费者1]received：删除商品：id二
〔消费者1]received：更新商品：id二
1001!
1001!

计算机生成了可选文字:
C八Java\jdkl.8.0_45\bin\java…
〔消费者2]received：新增商品：id二1001!
［消费者2]received：删除商品：id二1001!
氏肖费者2〕received:xxxx商品：id二1001!
［消费者2〕received：更新商品：id二1001!

## 2.7. 持久化

* 如何避免消息丢失？

1） 消费者的ACK机制。可以防止消费者丢失消息。

2） 但是，如果在消费者消费之前，MQ就宕机了，消息就没了。

* 如何将消息进行持久化呢？

要将消息持久化，前提是：队列、Exchange都持久化

### 2.7.1. 交换机持久化和消息持久化

计算机生成了可选文字:
pub1icclassSend{
prl甲atefinalstatlcStringEXC枯心昭E-－刀摊卫百二命topic_exchange_test牟
pub1icstatic甲。主doin(String［〕argy少thro,5Exception{
刀获取到连接
Connect主onconnection二ConnectionUtil.getConnect工on();
刀获取通道
Channe1channel二connection.createChannel():
刀声明。x动ang。，指定类型为top：。
channe1.exchangeDeclare（万万亡枯心昭E-－刀摊盆瓦
type：布t。p主c",1aurabie:tru
刀消，息、内容
Stringmessage二
”更新商品
id二1001禅
刀发送消J自、，并且指定工。LJt:ngke犷为：
channe1.bas主cPubl:sh（刃万亡枯心佑E-－刃她月双
Inselt，代表新增商品
routingKey:'item.upd。te",IMessagePr。pert主es．尸万左SIST朋tTEX几PLA班lmessage.getB:.'tes());
System.out.println("［商品服务：]Sent
即＋message＋禅，牟）;
channe1.close();
connection.close()
}
}

### 2.7.2. 队列持久化

计算机生成了可选文字:
pub1icclassRecv{
privatefinalstatlcString
privatefinalstaticString
O呱泥七刀沁卫百二尸top立c_exchange_queue_1牟
EX阴洲刘rGE-－四摊赵百二护topic_exchange_test布
pub1icstaticvoidmain(String[]argy)thro,5Exception{
刀获取到连接
Connectionco朋ection二ConnectionUtil.getConnect工on();
刀获取通间
Channe1channel二connection.createChannel();
/／声明队列
channe1．明eueDeclare(O泥姐－刀沁卫百．ld七，r"b'le:true,1exclusive:false,autoDelete:false,arguments:null)
刀绑定队列到交换机，同时指定需要订阅的IoLJtjng1'e,'o需要即da,e、del,te
cha朋e1．叩eueBind(O泥四－刃摊创瓦
cha朋e1.qLleueB主nd(O泥四－刃摊盆瓦
routingKey：尸主tem.update#);
routingKey:”主tem.deiete介）;
刀定义队列的消费者
DefaultConstunercons~r二ne,DefaultConstuner(channel){
刀获取消息、，并且处理，这个方法类似事件监听，如果有消息、的时候，会被自动调用
@override
pub1icvoidhandleDeliver:.,(StringconstunerTag.Envelopeenvelope，枷QP.BasicPropertiesproperties,
byte［〕body)t址。，5loExceptlon{
/／为。协·即消，自、体
String此g二ne,String(body);
Syste二out.println("［消费者1]received:"+msg+"!,);
}
};
刀监听队列，自动AcK
cha朋e1.basicConstune(O泥姐－四只创百，autoAck:true,constuner);
}
}

# 3. Spring AMQP

## 3.1. 简介

Spring有很多不同的项目，其中就有对AMQP的支持：

计算机生成了可选文字:
C白．spring.io/projects
☆昌e:
令一
令
SpplNGpESTOOCS
SpplNCMO日ILE
DocumentRESTfulservicesby
combininghand一wrltten
dOCUment日tionWithaUtO-
generatedSnippetSproduCed
withSpringMVCTestorREST
ASSUred.
ApplieSCOreSpringCOnCeptSto
thedeVelopmentofAMQP-
basedmessagingsolutions.
5impli石esthedevelopmentOf
mobilewebappsthrough
deVICedeteCtion日nd
。rogressiverenderlngoptions.

Spring AMQP的页面：<http://spring.io/projects/spring-amqp>

计算机生成了可选文字:
SpringAMQp,
O
TheSPringAMQPprOJectaPp!iescoreSPllllgco，、cePtStothedeveloPmentofAMQP一based
meSSagingSolutionS一ItproVidesa''template"asahigh一IevelabstraCtionforSendingandreceiVing
meSSages.ItaISOprOVideSSupportforMeSSage一driVenPOJOSwitha"IIStenerContainer，卜These
1ibrariesfacilitatemanagementofAMQPresourceSwhilepromotingtheuseofdependency
in)eCtionanddeClaratiVecon朽guration.InalloftheseCaseS,youwillSeeSimilaritieStotheJMS
SupportintheSpringFramework.
TheprojectconsistSoftwoparts;spring一amqp15thebaseabstraction,andspring一rabbit15the
RabbitMQimplementation.

注意这里一段描述：

Spring-amqp是对AMQP协议的抽象实现，而spring-rabbit 是对协议的具体实现，也是目前的唯一实现。底层使用的就是RabbitMQ。

## 3.2. 依赖和配置

添加AMQP的启动器：

|  |
| --- |
| <dependency>     <groupId>org.springframework.boot</groupId>     <artifactId>spring-boot-starter-amqp</artifactId>  </dependency> |

在application.yml中添加RabbitMQ地址：

|  |
| --- |
| spring:  rabbitmq:  host: 192.168.137.137  username: zhangsan  password: 123456  virtual-host: /shopping  port: 5672 |

## 3.3. 监听者

在SpringAmqp中，对消息的消费者进行了封装和抽象，一个普通的JavaBean中的普通方法，只要通过简单的注解，就可以成为一个消费者。

|  |
| --- |
| @Component  public class Listener {     @RabbitListener(bindings = @QueueBinding(             value = @Queue(value = "spring.test.queue", durable = "true"),             exchange = @Exchange(                     value = "spring.test.exchange",                     ignoreDeclarationExceptions = "true",                     type = ExchangeTypes.TOPIC            ),             key = {"#.#"}))     public void listen(String msg){         System.out.println("接收到消息：" + msg);    }  } |

* @Componet：类上的注解，注册到Spring容器
* @RabbitListener：方法上的注解，声明这个方法是一个消费者方法，需要指定下面的属性：
  + bindings：指定绑定关系，可以有多个。值是@QueueBinding的数组。@QueueBinding包含下面属性：
  + value：这个消费者关联的队列。值是@Queue，代表一个队列
  + exchange：队列所绑定的交换机，值是@Exchange类型
  + key：队列和交换机绑定的RoutingKey

类似listen这样的方法在一个类中可以写多个，就代表多个消费者。

## 3.4. AmqpTemplate

Spring最擅长的事情就是封装，把他人的框架进行封装和整合。

Spring为AMQP提供了统一的消息处理模板：AmqpTemplate，非常方便的发送消息，其发送方法：

计算机生成了可选文字:
岭On,ertAndsend(obj匹主
卜On,ertAndsend(S,r,ng
黑器器糕
convertAndsend(String
con,ertAndsend(String
message)J口口口口口口肠日口口口口口口
routingKey,objectmessage)
exchange,StringroutingKey,objectmessage)
message,MessageP
ostProcessor
丽药丽砰丽而石氏ssor)
rout:ngKey,objectmessage,MessagePostProcessormessagePostProcessor)
exchange,StringroutingKe:.',objectmessage,MessagePostProcessormessagePostProcessor

红框圈起来的是比较常用的3个方法，分别是：

* 指定消息
* 指定RoutingKey和消息，会向默认的交换机发送消息
* 指定交换机、RoutingKey和消息体

## 3.5. 测试代码

|  |
| --- |
| @RunWith(SpringRunner.class)  @SpringBootTest(classes = Application.class)  public class MqDemo {     @Autowired     private **AmqpTemplate** amqpTemplate;     @Test     public void testSend() throws InterruptedException {         String msg = "hello, Spring boot amqp";         this.amqpTemplate.convertAndSend("spring.test.exchange","a.b", msg);         // 等待10秒后再结束         Thread.sleep(10000);    }  } |

运行后查看日志：

计算机生成了可选文字:
／、、／_
(()\_
\\/_)
}_
::Spring
____(_)___＿、、、、
'_}'_}}'＿、／_'｝、、、、
l_)}}}}｝日（_}}))))
._l_}l_l_}｝八＿,}////
＿卜二二二二二二二二二二二二二｝_/=／一／一／一／
Boot::(v2.1.9.RELEASE)
nnnnn
ma姗姗mama
2019一11一1809:47:59
2019一11一1809:47:59
2019一11一1809:48:00
2019一11一1809:48:00
2019一11一1809:48:00
口口口口口口口口口门口口口．．口
接收到消，息、：hello,
.9丁811）万32968
.98011{F32968
.659工｝下32968
．丁41工｝下C.32968
.80711下C.32968
口口口口口口口口．口口口口口口
Springbootalnqp
－一〔
－一〔
－一〔
－一〔
－一〔