

RabbitMQ 文档

RabbitMQ 文档

第一章 RabbitMQ 简介

第二章 安装 RabbitMQ

2.1 环境准备

2.2 erlang安装

2.3 rabbitmq安装

2.4 安装bug

2.5 启动管理页面

2.6 命令模式添加账户

2.8学生做题

3.2.1 命令模式

3.2.2.1 访问web

3.3 学生做题

第四章 消息

4.1消息模式种类

4.1.1 pom&log4j.properties

4.1.2 工具类ConnectionUtil

4.2 简单模式

4.2.1 生产者

4.2.2 消费者

4.2.3 测试

4.2.4学生做题

4.3 work 模式

4.3.1 发送者

4.3.2 消费者1

4.3.3 消费者2

4.3.4 测试

4.3.5学生做题

4.4 消息的确认模式

4.5 订阅模式

4.5.1 生产者

4.5.2 消费者1

4.5.3 消费者2

4.5.4 测试

4.5.5学生做题

4.6 路由模式

4.6.1 生产者

4.6.2 消费者1

4.6.3 消费者2

4.6.4 测试

4.6.5学生做题

第五章 整合 spring

5.2学生做题

第六章 整合 springboot

6.1 pom.xml

6.2 yml 配置

6.3 sender 发送者

6.4 消费者的监听器

6.5 starter&config

6.6 测试

6.7 学生做题

第七章 RabbitMQ集群

7.1 配置hosts文件

7.2 两个节点分别启动rabbitmq-server

7.3 拷贝erlang.cookie

7.4 将mq02作为内存节点加入mq01节点集群中.

7.5 查看集群状态

7.6 登录rabbitmq web管理控制台

7.7 RabbitMQ镜像集群配置

7.7.1. 创建rabbitmq策略

7.7.2 分别登陆mq02、mq03两个节点的控制台

7.7.3 添加队列

7.7.4 创建消息

7.7.5 破坏性测试

7.8 学生做题

第八章 消息确认与延迟机制

8.1 AMQP事务

8.1.1 事务发送者

8.1.2 消费者

8.1.3 测试

8.2 confirm模式

8.2.1 普通confirm模式

8.2.2 批量confirm模式

8.2.3 消费者

8.2.4 测试

8.3 延迟队列

8.3.1 延迟队列概念

8.3.2 延迟消息发送

8.3.3 延迟消息消费

8.3.4 测试

8.4 学生做题

第九章 MQ面试

9.1 rabbitmq与kafka对比

9.2 为什么使用消息队列

9.3 如何保证消息队列的高可用

9.4 如何保证消息不被重复消费啊（幂等性）

9.5 如何保证消息的可靠性传输（消息丢失问题）

9.6 如何保证消息的顺序性

第一章 RabbitMQ 简介

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。其中较为成熟的MQ产品有IBM WEBSPPHERE MQ等等。

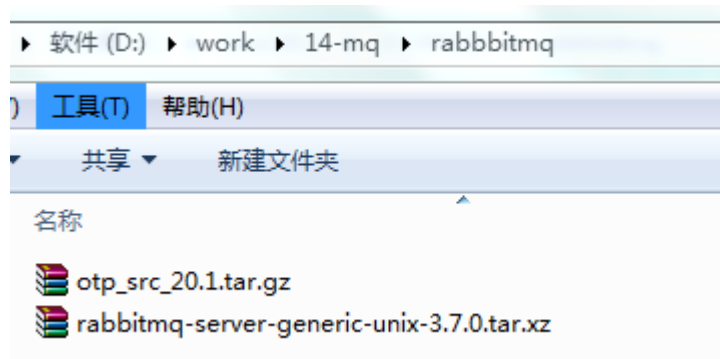
RabbitMQ是一个在AMQP基础上完成的，可复用的企业消息系统。他遵循Mozilla Public License开源协议。

AMQP，即Advanced Message Queuing Protocol,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/中间件不同产品，不同的开发语言等条件的限制。Erlang中的实现有 RabbitMQ等。

第二章 安装 RabbitMQ

CentOS 7 安装 RabbitMQ 3.7

本地安装，减少网络的依赖。



如果之前安装过erlang,先删除原有的yum remove erlang*

2.1 环境准备

```
yum -y install make gcc gcc-c++ kernel-devel m4 ncurses-devel openssl-devel unixODBC unixODBC-devel  
httpd python-simplejson
```

2.2 erlang安装

下载:wget http://www.erlang.org/download/otp_src_20.1.tar.gz

```
[root@ChengGeGe opt]# ll  
total 96432  
-rw-r--r-- 1 root root 87342296 May 27 20:22 otp_src_20.1.tar.gz  
-rw-r--r-- 1 root root 11396700 May 27 20:02 rabbitmq-server-generic-unix-3.7.0.tar.xz
```

```
tar -xvf otp_src_20.1.tar.gz
```

```
cd otp_src_20.1
```

```
[root@ChengGeGe opt]# cd otp_src_20.1
[root@ChengGeGe otp_src_20.1]# ll
total 608
-rw-rw-r-- 1 421 wheel  90488 Sep 26  2017 aclocal.m4
-rw-rw-r-- 1 421 wheel    601 Sep 26  2017 AUTHORS
drwxrwxr-x 2 421 wheel  4096 Sep 26  2017 bin
drwxrwxr-x 3 421 wheel  4096 Sep 26  2017 bootstrap
-rwxrwxr-x 1 421 wheel 152109 Sep 26  2017 configure
-rw-rw-r-- 1 421 wheel  15316 Sep 26  2017 configure.in
-rw-rw-r-- 1 421 wheel   4474 Sep 26  2017 CONTRIBUTING.md
-rw-rw-r-- 1 421 wheel   1146 Sep 26  2017 erl-build-tool-vars.sh
drwxrwxr-x 18 421 wheel  4096 Sep 26  2017 erts
drwxrwxr-x 2 421 wheel  4096 Sep 26  2017 HOWTO
drwxrwxr-x 47 421 wheel  4096 Sep 26  2017 lib
-rw-rw-r-- 1 421 wheel 10175 Sep 26  2017 LICENSE.txt
```

指定安装目录：

./configure --prefix=/usr/local/erlang --enable-smp-support --enable-threads --enable-sctp --enable-kernel-poll
 --enable-hipe --with-ssl --without-javac

```
config.status: creating ../lib/otp_src_20.1/x86_64-unknown-linux-gnu/Makefile
config.status: creating ../lib/runtime_tools/c_src/x86_64-unknown-linux-gnu/Makefile
config.status: creating ../lib/tools/c_src/x86_64-unknown-linux-gnu/Makefile
config.status: creating x86_64-unknown-linux-gnu/config.h
config.status: creating include/internal/x86_64-unknown-linux-gnu/ethread_header_config.h
config.status: creating include/x86_64-unknown-linux-gnu/erl_int_sizes_config.h
*****
***** APPLICATIONS DISABLED *****
*****
jinterface      : Java compiler disabled by user

*****
***** APPLICATIONS INFORMATION *****
*****
wx              : wxWidgets not found, wx will NOT be usable

*****
***** DOCUMENTATION INFORMATION *****
*****
documentation  :
                 xsltproc is missing.
                 fop is missing.
                 The documentation can not be built.
*****
```

编译与安装

make && make install

```
/usr/bin/install -c -m 644 "/opt/otp_src_20.1/OTP_VERSION" "/usr/local/erlang/lib/erlang/releases/20"
cd /usr/local/erlang/bin
rm -f erl
rm -f erlc
rm -f epmd
rm -f run_erl
rm -f to_erl
rm -f dialyzer
rm -f escript
rm -f ct_run
ln -s ../lib/erlang/bin/erl erl
ln -s ../lib/erlang/bin/erlc erlc
ln -s ../lib/erlang/bin/epmd epmd
ln -s ../lib/erlang/bin/run_erl run_erl
ln -s ../lib/erlang/bin/to_erl to_erl
ln -s ../lib/erlang/bin/dialyzer dialyzer
ln -s ../lib/erlang/bin/escript escript
ln -s ../lib/erlang/bin/ct_run ct_run
```

配置erlang环境变量:

vim /etc/profile

文件最后放入: export PATH=\$PATH:/usr/local/erlang/bin

```
export JAVA_HOME=/usr/local/software/jdk1.8.0_151
export JRE_HOME=/usr/local/software/jdk1.8.0_151/jre
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:/usr/local/erlang/bin
```

//执行下面命令修改配置文件立即生效 **source /etc/profile**

验证:

erl

```
[root@ChengGeGe otp_src_20.1]# erl
Erlang/OTP 20 [erts-9.1] [source] [64-bit] [smp:1:1]

Eshell V9.1 (abort with ^G)
1> 
```

2.3 rabbitmq安装

下载 wget <https://github.com/rabbitmq/rabbitmq-server/releases/download/v3.7.0/rabbitmq-server-generic-unix-3.7.0.tar.xz>

```
[root@ChengGeGe opt]# ll
total 96432
-rw-r--r-- 1 root root 87342296 May 27 20:22 otp_src_20.1.tar.gz
-rw-r--r-- 1 root root 11396700 May 27 20:02 rabbitmq-server-generic-unix-3.7.0.tar.xz
```

xz -d rabbitmq-server-generic-unix-3.7.0.tar.xz

tar -xvf rabbitmq-server-generic-unix-3.7.0.tar

解压就可以直接启动使用

cd rabbitmq_server-3.7.0/sbin

```
[root@ChengGeGe rabbitmq_server-3.7.0]# cd sbin/
[root@ChengGeGe sbin]# ll
total 508
-rwxr-xr-x 1 root root 468026 Nov 30 2017 cuttlefish
-rwxr-xr-x 1 root root 1228 Nov 30 2017 rabbitmqctl
-rwxr-xr-x 1 root root 2050 Nov 30 2017 rabbitmq-defaults
-rwxr-xr-x 1 root root 1237 Nov 30 2017 rabbitmq-diagnostics
-rwxr-xr-x 1 root root 12969 Nov 30 2017 rabbitmq-env
-rwxr-xr-x 1 root root 1256 Nov 30 2017 rabbitmq-plugins
-rwxr-xr-x 1 root root 13266 Nov 30 2017 rabbitmq-server
```

启动: `./rabbitmq-server -detached`

```
[root@ChengGeGe sbin]# ./rabbitmq-server -detached
Warning: PID file not written; -detached was passed.
```

查看进程: `ps`

```
[root@ChengGeGe sbin]# ps -aux |grep rabbit
root      32634  2.3  3.8 2340636 71868 ?        S1   20:43   0:02 /usr/local/erlang/lib/erlang/erts-9.1/bin
000 -K true -- -root /usr/local/erlang/lib/erlang -programe erl -- -home /root -- -pa /opt/rabbitmq_server
ngGeGe -boot start_sasl -kernel inet_default_connect_options [{nodelay,true}] -sasl errlog_type error -sas
q_server-3.7.0/var/log/rabbitmq" -rabbit lager_default_file "/opt/rabbitmq_server-3.7.0/var/log/rabbitmq/r
_server-3.7.0/var/log/rabbitmq/rabbit@ChengGeGe_upgrade.log" -rabbit enabled_plugins_file "/opt/rabbitmq_s
"/opt/rabbitmq_server-3.7.0/plugins" -rabbit plugins_expand_dir "/opt/rabbitmq_server-3.7.0/var/lib/rabbit
up false -os_mon start_disksup false -os_mon start_memsup false -mnesia dir "/opt/rabbitmq_server-3.7.0/va
en_min 25672 -kernel inet_dist_listen_max 25672 -noshell -noinput
```

2.4 安装bug

注意: 不要轻易修改主机名

bug1:ERROR: epmd error for host "yourhostname": timeout

原因是: 主机名和ip不匹配, 需要更改hostname或者/etc/hosts文件修改主机名: `#hostname yourhostname` 要跟/etc/hosts文件中一致。

```
root@ChengGeGe:/opt/rabbitmq_server-3.7.0/sbin
127.0.0.1    localhost localhost.localdomain localh
127.0.0.1    mycomputer
::1         localhost localhost.localdomain localh
101.132.99.151 chengdi9782.xyz
101.132.99.151 caigege.vip
127.0.0.1    ChengGeGe
```

bug2:主机名为bogon的设置账户会有失败的情况 (bogon是个非正常账户)

解决方案:

`vi /etc/hosts`

`127.0.0.1 bogon`

2.5 启动管理页面

./rabbitmq-plugins enable rabbitmq_management

```
[root@ChengGeGe sbin]# ./rabbitmq-plugins enable rabbitmq_management
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@ChengGeGe...
The following plugins have been enabled:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
started 3 plugins.
```

2.6 命令模式添加账户

配置用户:

./rabbitmqctl add_user admin admin

```
[root@ChengGeGe sbin]# ./rabbitmqctl add_user admin admin
Adding user "admin" ...
```

设置权限

./rabbitmqctl set_user_tags admin administrator

#设置admin为administrator级别

```
[root@ChengGeGe sbin]# ./rabbitmqctl set_user_tags admin administrator
Setting tags for user "admin" to [administrator] ...
```

###2.7放行端口

网络指令安装yum install net-tools (如果没有)

```
[root@iZ2ze3lulvta6b0e67qi0zZ plugins]# netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:25672           0.0.0.0:*               LISTEN      23398/beam.smp
tcp        0      0 0.0.0.0:8009           0.0.0.0:*               LISTEN      6530/java
tcp        0      0 0.0.0.0:3690           0.0.0.0:*               LISTEN      23760/svnserve
tcp        0      0 0.0.0.0:8080           0.0.0.0:*               LISTEN      21310/java
tcp        0      0 0.0.0.0:4369           0.0.0.0:*               LISTEN      23571/epmd
tcp        0      0 0.0.0.0:8081           0.0.0.0:*               LISTEN      6530/java
tcp        0      0 0.0.0.0:22             0.0.0.0:*               LISTEN      3418/sshd
tcp        0      0 0.0.0.0:15672           0.0.0.0:*               LISTEN      23398/beam.smp
tcp        0      0 0.0.0.0:127.0.0.1:8005  0.0.0.0:*               LISTEN      6530/java
tcp6       0      0 :::5672                :::*                   LISTEN      23398/beam.smp
tcp6       0      0 :::4369                :::*                   LISTEN      23571/epmd
```

```
firewall-cmd --add-port=15672/tcp --permanent
```

```
firewall-cmd --add-port=5672/tcp --permanent
```

```
[root@localhost plugins]# firewall-cmd --add-port=15672/tcp --permanent  
success  
[root@localhost plugins]# firewall-cmd --add-port=5672/tcp --permanent  
success
```

可以直接关闭防火墙:

关闭: `systemctl stop firewalld`

开机禁用: `systemctl disable firewalld`

查看状态: `systemctl status firewalld`

阿里云控制台:

添加安全组规则 ⓘ 添加安全组规则 ×

网卡类型:	内网 ▼
规则方向:	入方向 ▼
授权策略:	允许 ▼
协议类型:	自定义 TCP ▼
* 端口范围:	5672/5672 ⓘ
优先级:	1 ⓘ
授权类型:	IPv4地址段访问 ▼
* 授权对象:	0.0.0.0/0 ⓘ 教我设置

2.8学生做题

搭建rabbitmq环境

##第三章 添加用户

###3.1 账号级别

1. 超级管理员administrator,可以登录控制台,查看所有信息,可以对用户和策略进行操作
2. 监控者monitoring,可以登录控制台,可以查看节点的相关信息,比如进程数,内存磁盘使用情况
3. 策略制定者policymaker ,可以登录控制台,制定策略,但是无法查看节点信息
4. 普通管理员 management 仅能登录控制台
5. 其他, 无法登录控制台,一般指的是提供者 and 消费者

###3.2 添加账号

3.2.1 命令模式

rabbitmqctl add_user luke luke #添加账号 luke 密码是 luke

```
[root@iZ2ze3lulvta6b0e67qi0zZ plugins]# rabbitmqctl add_user qf qf
Adding user "qf" ...
[root@iZ2ze3lulvta6b0e67qi0zZ plugins]#
```

rabbitmqctl set_user_tags luke administrator #设置 luke 为administrator级别

```
[root@iZ2ze3lulvta6b0e67qi0zZ plugins]# rabbitmqctl set_user_tags qf administrator
Setting tags for user "qf" to [administrator] ...
[root@iZ2ze3lulvta6b0e67qi0zZ plugins]#
```

####3.2.2 web 方式

可以通过 web 页面添加账号,此方式需要开启 web 访问

3.2.2.1 访问web

防火墙关闭:

查看下防火墙的状态: systemctl status firewalld

关闭: systemctl stop firewalld

开机禁用: systemctl disable firewalld

访问:

<http://192.168.3.227:15672/> 此处 ip 是本人 ip, 实际中请以实际 ip 为准

101.201.236.229:15672



Username: =

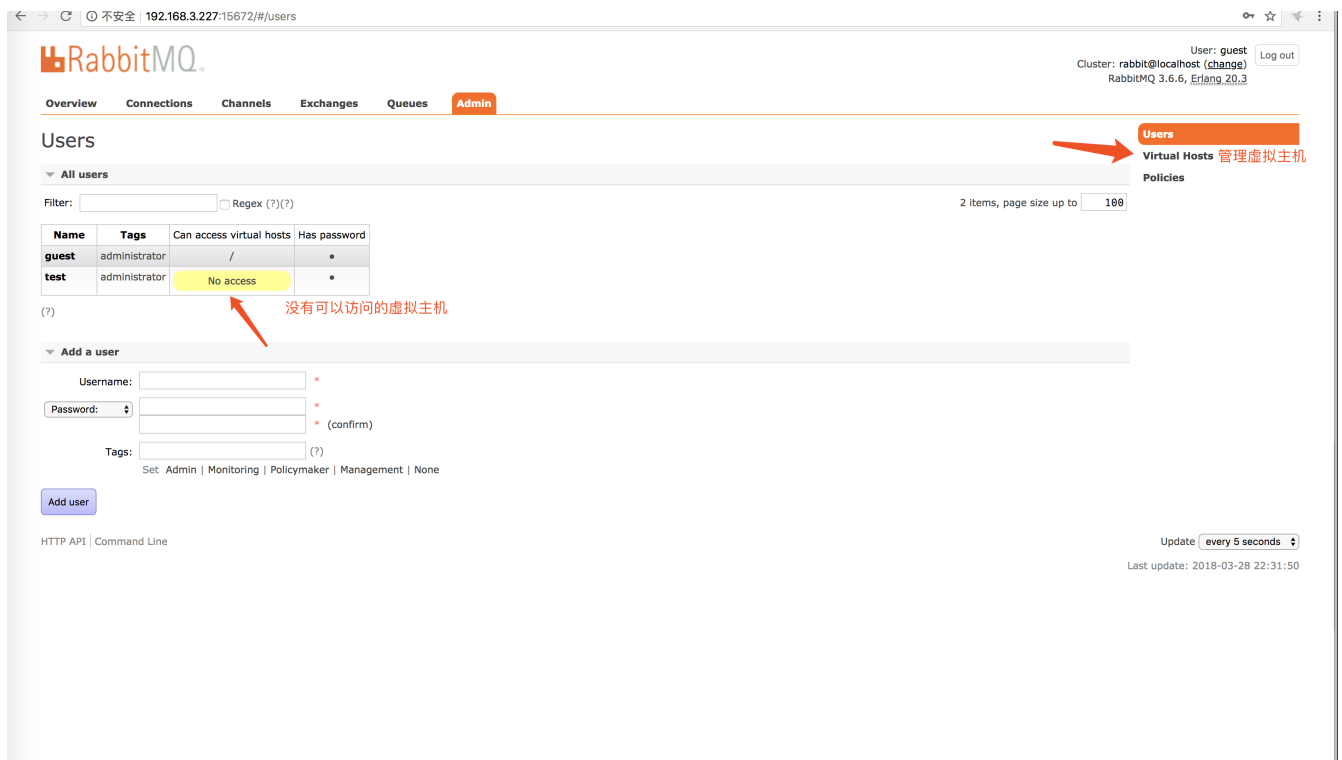
Password: =

使用 guest guest 登录 guest 具有最高权限

#####3.2.2.2 添加用户

#####3.2.2.3 分配可以访问的虚拟主机

默认情况下没有任何可以访问的,我们可以添加一个主机(相当于添加一个数据库),然后分配权限



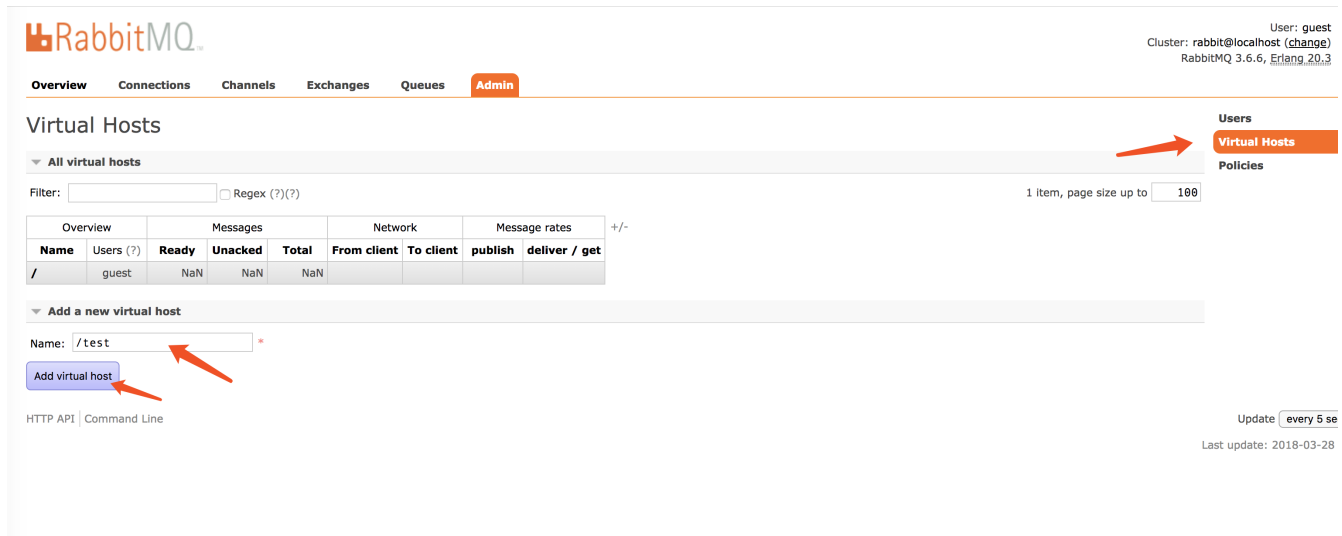
The screenshot shows the RabbitMQ Admin UI for the 'Users' page. The top navigation bar includes 'Overview', 'Connections', 'Channels', 'Exchanges', 'Queues', and 'Admin'. The 'Users' page is active, showing a list of users. A red arrow points to the 'Users' tab in the top right. Another red arrow points to the 'No access' status for the 'test' user, with a note '没有可以访问的虚拟主机' (No virtual hosts can be accessed). The 'Add a user' form is visible below the list.

Name	Tags	Can access virtual hosts	Has password
guest	administrator	/	*
test	administrator	No access	*

Filter: ☐ Regex (?) (?) 2 items, page size up to 100

Update: every 5 seconds Last update: 2018-03-28 22:31:50

#####3.2.2.4 创建虚拟主机



The screenshot shows the RabbitMQ Admin UI for the 'Virtual Hosts' page. The top navigation bar includes 'Overview', 'Connections', 'Channels', 'Exchanges', 'Queues', and 'Admin'. The 'Virtual Hosts' page is active, showing a list of virtual hosts. A red arrow points to the 'Virtual Hosts' tab in the top right. Another red arrow points to the 'Add virtual host' button. The 'Add a new virtual host' form is visible below the list.

Overview	Messages	Network	Message rates					
Name	Users (?)	Ready	Unacked	Total	From client	To client	publish	deliver / get
/	guest	NaN	NaN	NaN				

Filter: ☐ Regex (?) (?) 1 item, page size up to 100

Update: every 5 seconds Last update: 2018-03-28

#####3.2.2.5 给虚拟主机分配权限

Virtual Hosts

▼ All virtual hosts

Filter: ☐ Regex (?)

Overview		Messages			Network		Message rates		
Name	Users (?)	Ready	Unacked	Total	From client	To client	publish	deliver / get	+/-
/	guest	NaN	NaN	NaN					
/test	No users	NaN	NaN	NaN					

▼ Add a new virtual host

Name: *

Add virtual host

HTTP API | Command Line

#####3.2.2.6 给指定用户分配权限

Virtual Host: /test

Overview

Queued messages (chart: last minute) (?)

Waiting for data...

Message rates (chart: last minute) (?)

Currently idle

Data rates (chart: last minute)

Waiting for data...

Details

Tracing enabled: ☐

▼ Permissions

Current permissions

User	Configure regexp	Write regexp	Read regexp
test	.*	.*	.*

Clear

Set permission

User:

Configure regexp:

Write regexp:

Read regexp:

Set permission

设置完成后，回到用户界面确认：

Users

▼ All users

Filter: ☐ Regex ?

Name	Tags	Can access virtual hosts	Has
guest	administrator	/, /qf, /test	
qf	administrator	/qf	
qf2	administrator	No access	

3.3 学生做题

RabbitMQ中添加个人账户

第四章 消息

<http://www.rabbitmq.com/getstarted.html>

消息测试都在一个项目中,不同包下做测试

4.1消息模式种类

1 "Hello World!"

The simplest thing that does
something



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

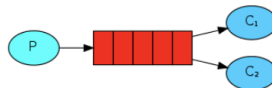
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

2 Work queues

Distributing tasks among
workers (the [competing
consumers pattern](#))



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

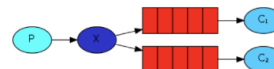
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

3 Publish/Subscribe

Sending messages to many
consumers at once



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

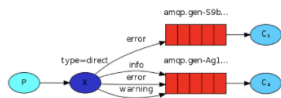
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

4 Routing

Receiving messages selectively



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

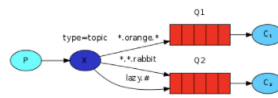
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

5 Topics

Receiving messages based on a pattern (topics)



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

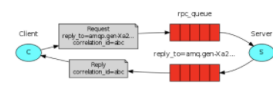
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

6 RPC

[Request/reply pattern](#)
example



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

[Spring AMQP](#)

4.1.1 pom&log4j.properties

创建pom项目，加入依赖。

```
<dependencies>
  <dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>4.5.0</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.25</version>
  </dependency>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.3.2</version>
  </dependency>
<!-- https://mvnrepository.com/artifact/org.springframework.amqp/spring-rabbit
```

整合 spring 时使用,amqp 只对 rabbitmq 做了支持

-->

```
<dependency>
  <groupId>org.springframework.amqp</groupId>
  <artifactId>spring-rabbit</artifactId>
  <version>1.7.6.RELEASE</version>
```

```
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>4.3.7.RELEASE</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
</dependencies>
```

log4j.properties

```
log4j.rootLogger=DEBUG,A1
log4j.logger.com.taotao = DEBUG
log4j.logger.org.mybatis = DEBUG

log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n
```

4.1.2 工具类ConnectionUtil

注意导包

```
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
public class ConnectionUtil {

    public static Connection getConnection() throws Exception {
        //定义连接工厂
        ConnectionFactory factory = new ConnectionFactory();
        //设置服务地址
        factory.setHost("192.168.3.227");
        //端口
        factory.setPort(5672);
        //设置账号信息, 用户名、密码、vhost
        factory.setVirtualHost("/test");
        factory.setUsername("test");
        factory.setPassword("test");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        return connection;
    }
}
```

Users

▼ All users

Filter: ☐ Regex ?

Name	Tags	Can access virtual hosts	Has password
admin	administrator	No access	•
guest	administrator	/, /qf	•
qf	administrator	/qf	•

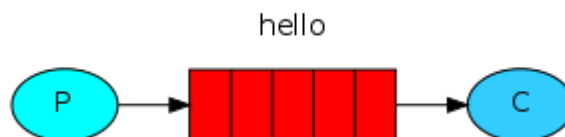
?

4.2 简单模式

<http://www.rabbitmq.com/tutorials/tutorial-one-java.html>

简单模式就是我们的生产者将消息发到队列,消费者从队列中取消息

一条消息对应一个消费者



4.2.1 生产者

```

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
public class Send {

    private final static String QUEUE_NAME = "test_queue";

    public static void main(String[] argv) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();//相当于数据库中的创建连接
        // 从连接中创建通道
        Channel channel = connection.createChannel();//相当于数据库中的 statement

        // 声明（创建）队列,如果存在就不创建,不存在就创建
        //参数1 队列名,
        //参数2 durable: 是否持久化, 队列的声明默认是存放到内存中的, 如果rabbitmq重启会丢失, 如果想重
        启之后还存在就要使队列持久化, 保存到Erlang自带的Mnesia数据库中, 当rabbitmq重启之后会读取该数据库
    }
  
```



```
//exclusive: 是否排外的, 有两个作用, 一: 当连接关闭时connection.close()该队列是否会自动删除;
二: 该队列是否是私有的private, 如果不是排外的, 可以使用两个消费者都访问同一个队列, 没有任何问题, 如果是排外的, 会对当前队列加锁, 其他通道channel是不能访问的, 如果强制访问会报异常:
com.rabbitmq.client.ShutdownSignalException: channel error; protocol method:
#method<channel.close>(reply-code=405, reply-text=RESOURCE_LOCKED - cannot obtain exclusive
access to locked queue 'queue_name' in vhost '/', class-id=50, method-id=20)一般等于true的话
用于一个队列只能有一个消费者来消费的场景

//autoDelete: 是否自动删除, 当最后一个消费者断开连接之后队列是否自动被删除, 可以通过RabbitMQ
Management, 查看某个队列的消费者数量, 当consumers = 0时队列就会自动删除

//arguments: 参数
channel.queueDeclare(QUEUE_NAME, false, false, false, null);

// 消息内容
String message = "Hello world!";
//参数1 交换机, 此处无
//参数2 发送到哪个队列
//参数3 属性 参数4 内容
channel.basicPublish("", QUEUE_NAME, null, message.getBytes()); //将消息发动到数据库
System.out.println(" 发送数据 " + message + "");

//关闭通道和连接
channel.close();
connection.close();
}
}
```

后台查看:

Overview
Connections
Channels
Exchanges
Queues
Admin

Queues

▼ All queues (1)

Pagination

Page 1 ▼ of 1 - Filter: ☐ Regex ?

Overview				Messages			Message rates		
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/qf	qf_queue		idle	1	0	1	0.00/s		

4.2.2 消费者

```
package com.qf.rabbitmq;

import com.rabbitmq.client.*;

import java.io.IOException;

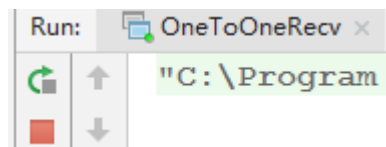
public class Recv {
    private final static String QUEUE_NAME = "qf_queue";
```

```
public static void main(String[] argv) throws Exception {
    // 获取到连接以及mq通道
    Connection connection = ConnectionUtil.getConnection();
    Channel channel = connection.createChannel();
    //参数1 队列的名字
    //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存
    放到数据库中,当服务器重启后会重新读取数据
    //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有
    了,只能由一个消费者能访问该队列
    //参数4 autoDelete: 是否自动删除
    channel.queueDeclare(QUEUE_NAME, false, false, false, null);
    Consumer consumer = new DefaultConsumer(channel){
        @Override
        public void handleDelivery(String consumerTag, Envelope envelope,
    AMQP.BasicProperties properties, byte[] body) throws IOException {
            // 捕获消息内容
            String message = new String(body, "UTF-8");
            System.out.println(message);
            //应答,告诉服务器我收到消息了
            channel.basicAck(envelope.getDeliveryTag(), false);
        }
    };
    //参数2 true表示自动应答,当收到消息的时候自动告诉服务器我已经收到消息,这样的话 服务器就不会一直给
    我们推送消息
    channel.basicConsume(QUEUE_NAME, true, consumer); //使用当前消费者处理指定队列的消息
}
}
```

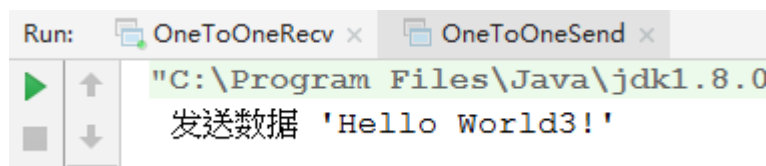
4.2.3 测试

运行 send, 和 recv 测试可以收发消息

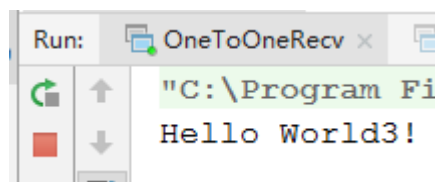
开启消费:



开启发送:



消费者收到消息:



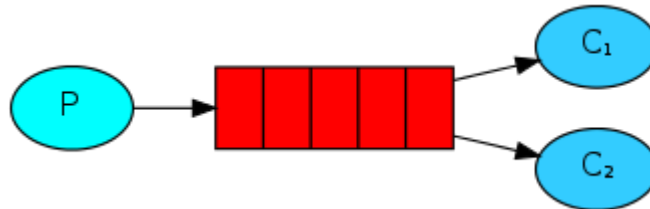
4.2.4 学生做题

测试简单模式

4.3 work 模式

一条消息可以被多个消费者尝试接收,但是最终只能有一个消费者能获取

<https://www.rabbitmq.com/tutorials/tutorial-two-java.html>



4.3.1 发送者

```
public class Send {  
  
    private final static String QUEUE_NAME = "test_queue_work";  
  
    public static void main(String[] argv) throws Exception {  
        // 获取到连接以及mq通道  
        Connection connection = ConnectionUtil.getConnection();  
        Channel channel = connection.createChannel();  
  
        // 声明队列  
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);  
  
        for (int i = 0; i < 100; i++) { // 循环发送消息,但是每条消息的时间间隔越来越长  
            // 消息内容  
            String message = "" + i;  
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes());  
            System.out.println(" 发送消息 '" + message + "'");  
  
            Thread.sleep(i * 10); // 休眠  
        }  
  
        channel.close();  
        connection.close();  
    }  
}
```

4.3.2 消费者1

```
public class Recv {  
    private static final String QUEUENAME = "workqueue";  
  
    public static void main(String[] args) throws Exception {
```

```
Connection connection = ConnectionUtils.getConnection();//获取连接
Channel channel = connection.createChannel();//相当于我们连接数据库的时候的 statement\
//参数1 队列的名字
//参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据
//参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有
了,只能由一个消费者能访问该队列
channel.queueDeclare(QUEUENAME, false, false , false, null);
//注释掉后可以获取多条消息,但是会一条一条处理
channel.basicQos(1);//声明每次处理一个消息,只有处理完成并应答服务器之后才会处理下一条消息

//创建消费者,并重写处理消息的方法,此方法为阻塞的,不需要后面 while 循环
DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
        System.out.println("消费者1收到的内容是:"+new String(body));
        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //应答,告诉服务器我收到消息了
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};
// 监听队列, 手动返回完成,参数2手动确认模式
channel.basicConsume(QUEUENAME, false, defaultConsumer);
}
}
```

4.3.3 消费者2

```
public class Recv2 {
private static final String QUEUENAME = "workqueue";

    public static void main(String[] args) throws Exception {

        Connection connection = ConnectionUtils.getConnection();//获取连接
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的 statement\
        //参数1 队列的名字
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有
        了,只能由一个消费者能访问该队列
        channel.queueDeclare(QUEUENAME, false, false , false, null);
        channel.basicQos(1);//声明每次处理一个消息
        DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者2222收到的内容是:"+new String(body));
            }
        };
        channel.basicConsume(QUEUENAME, false, defaultConsumer);
    }
}
```

```

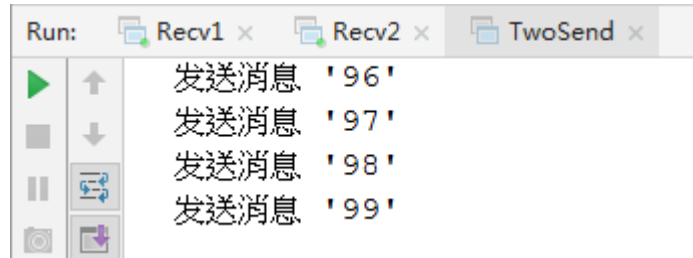
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //应答
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};

channel.basicConsume(QUEUENAME, false, defaultConsumer);
}
}

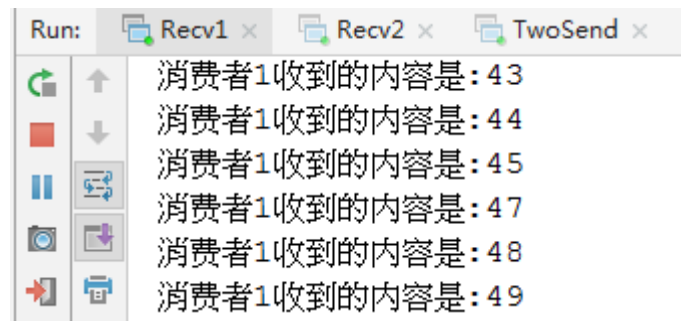
```

4.3.4 测试

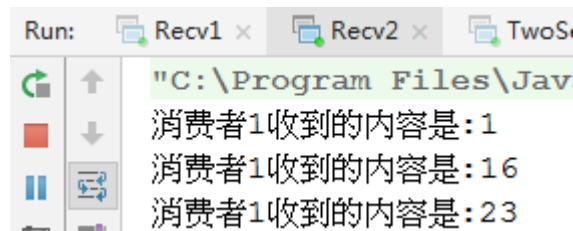
启动消费者1,消费者2,发送者



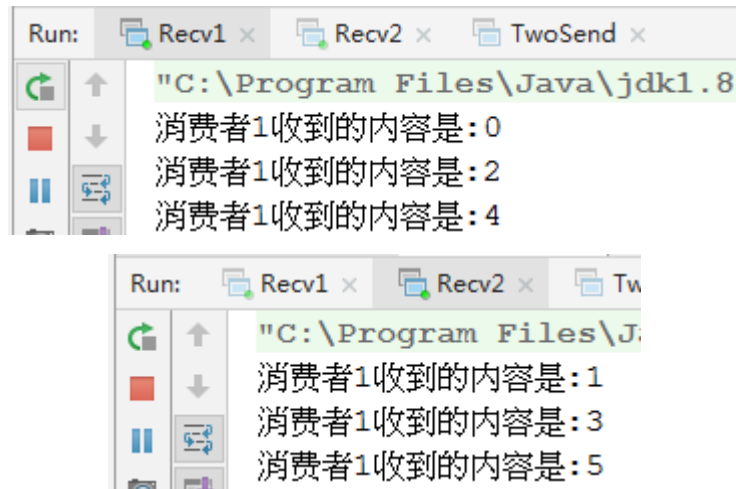
channel.basicQos(1);代码打开后,发现消费者1可以获取到更多数据,因为消费者的处理时间端,处理快,所以可以获取到更多的消息



消费者2明显消费慢:



在channel.basicQos(1);代码注释掉的情况下,我们发现两个消费者获取到的消息数量是一致的,会轮流从队列取消息



4.3.5 学生做题

测试work模式

4.4 消息的确认模式

当我们发送消息后,服务端如何知道消息已经被消费

模式1:自动模式,不管消费者获取到消息后是否是成功处理消息,服务端都认为是成功的

模式2:手动模式,消费者获取到消息后,服务器会将消息标记为不可用,等待消费者反馈,如果不反馈,则一直标记为不可用

4.5 订阅模式

<https://www.rabbitmq.com/tutorials/tutorial-three-java.html>

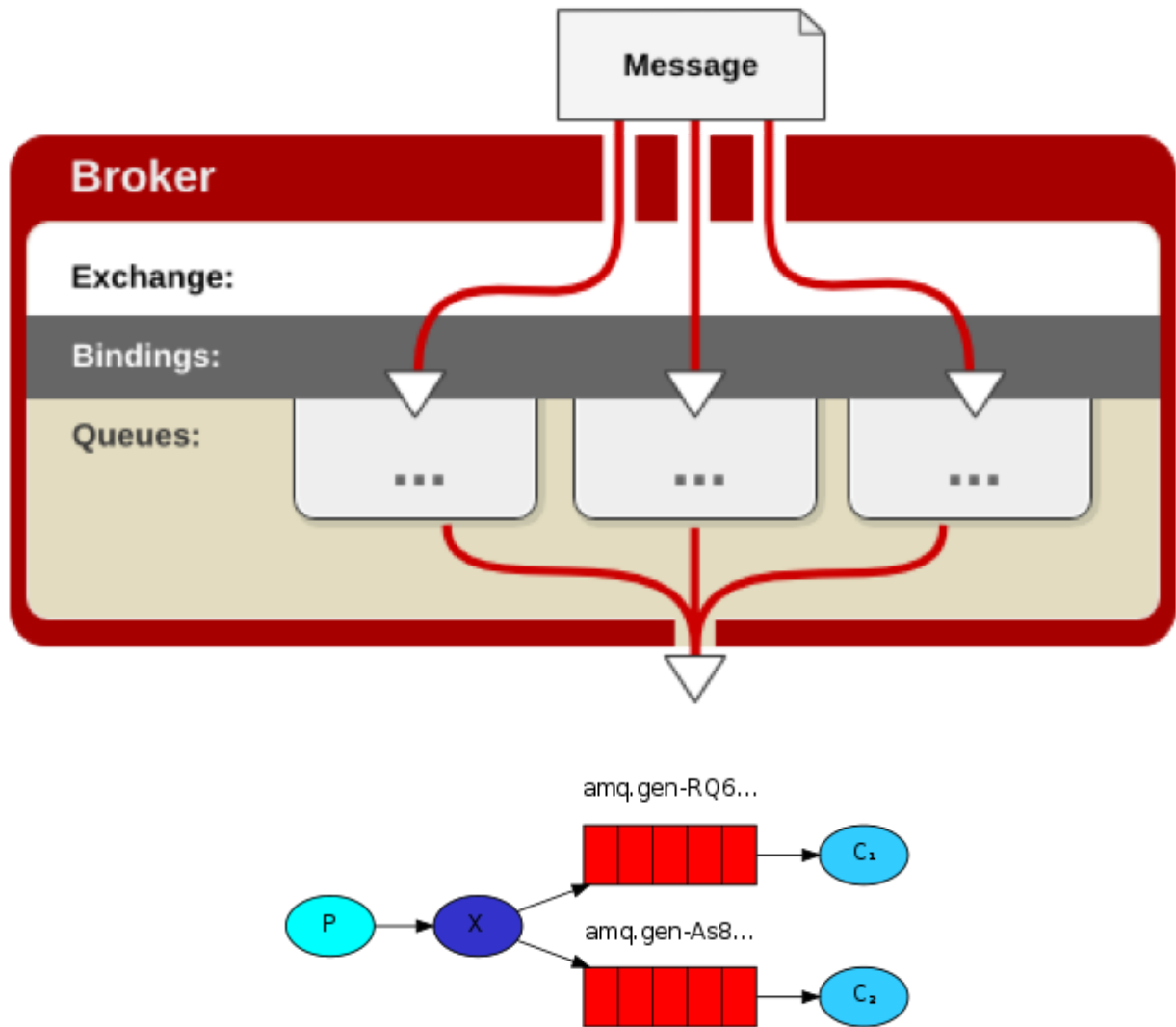
一条消息可以被多个消费者同时获取

生产者将消息发送到交换机

消费者将自己对应的队列注册到交换机

当发送消息后 所有注册的队列的消费者都可以收到消息

Fanout Exchange



4.5.1 生产者

```
public class Send {

    private final static String EXCHANGE_NAME = "test_exchange_fanout";

    public static void main(String[] argv) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明exchange
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

        // 消息内容
        String message = "Hello world!";
        //将消息发送到交换机,如果此时没有队列绑定,则消息会丢失,因为交换机没有存储消息的能力
        channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
        System.out.println(" 发送消息 '" + message + "'");
    }
}
```

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

4.5.2 消费者1

```
public class Recv {

    private static final String EXCHANGENAME = "fanoutexchange222222";
    private static final String QUEUENAME = "subqueue1";

    public static void main(String[] args) throws Exception {
        Connection connection = ConnectionUtils.getConnection();//获取连接
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的 statement\
        //交换机必须先声明后使用,不管是谁声明的,队列必须先存在
        channel.exchangeDeclare(EXCHANGENAME, "fanout");
        //参数1 队列的名字
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放
        //到数据库中,当服务器重启后会重新读取数据
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有
        //了,只能由一个消费者能访问该队列
        channel.queueDeclare(QUEUENAME, false, false, false, null);
        channel.queueBind(QUEUENAME, EXCHANGENAME, ""); //将制定的队列绑定到指定的交换机上面
        channel.basicQos(1); //声明每次处理一个消息,只有处理完成并应答服务器之后才会处理下一条消息
        DefaultConsumer defaultConsumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
                AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者1收到的内容是:" + new String(body));
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                //应答,告诉服务器我收到消息了
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };

        channel.basicConsume(QUEUENAME, false, defaultConsumer);
    }
}
```

4.5.3 消费者2

```
public class Recv2 {

    private static final String EXCHANGENAME = "fanoutexchange222222";
    private static final String QUEUENAME = "subqueue2";
```



```
public static void main(String[] args) throws Exception {

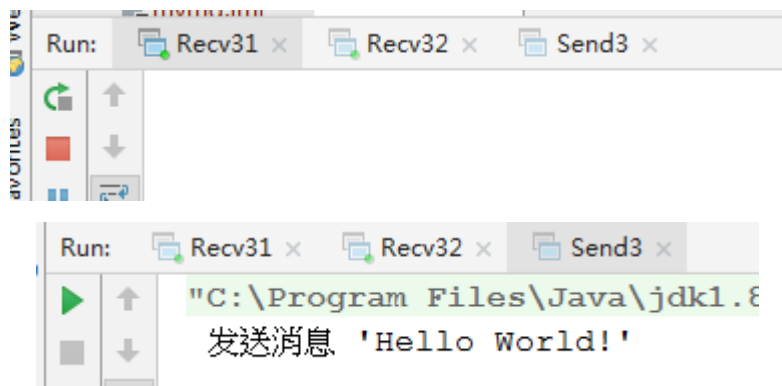
    Connection connection = ConnectionUtils.getConnection();//获取连接
    Channel channel = connection.createChannel();//相当于我们连接数据库的时候的 statement\
//交换机必须先声明后使用,不管是谁声明的,队列必须先存在
    channel.exchangeDeclare(EXCHANGENAME, "fanout");
    //参数1 队列的名字
    //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放
    到数据库中,当服务器重启后会重新读取数据
    //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有
    了,只能由一个消费者能访问该队列
    channel.queueDeclare(QUEUENAME,false,false ,false,null);
    channel.queueBind(QUEUENAME, EXCHANGENAME, "");//将制定的队列绑定到指定的交换机上面
    channel.basicQos(1);//声明每次处理一个消息
    DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
        @Override
        public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
            System.out.println("消费者2222收到的内容是:"+new String(body));

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            //应答
            channel.basicAck(envelope.getDeliveryTag(), false);
        }
    };

    channel.basicConsume(QUEUENAME, false, defaultConsumer);
}
}
```

4.5.4 测试

启动消费者1,2 生产者测试



发送后, 每个消费者都可以订阅到一样的消息。

服务端队列确认:

Overview
Connections
Channels
Exchanges
Queues

Queues

▼ All queues (4)

Pagination

Page 1 ▼ of 1 - Filter: ☐ Regex ?

Overview				Messages			M
Virtual host	Name	Features	State	Ready	Unacked	Total	in
/qf	qf_queue		idle	0	0	0	
/qf	qf_queue2		idle	0	0	0	
/qf	subqueue1		idle	0	0	0	
/qf	subqueue2		idle	0	0	0	

4.5.5 学生做题

测试订阅模式

4.6 路由模式

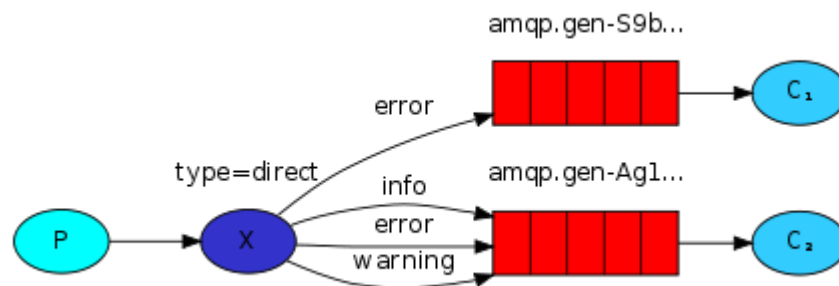
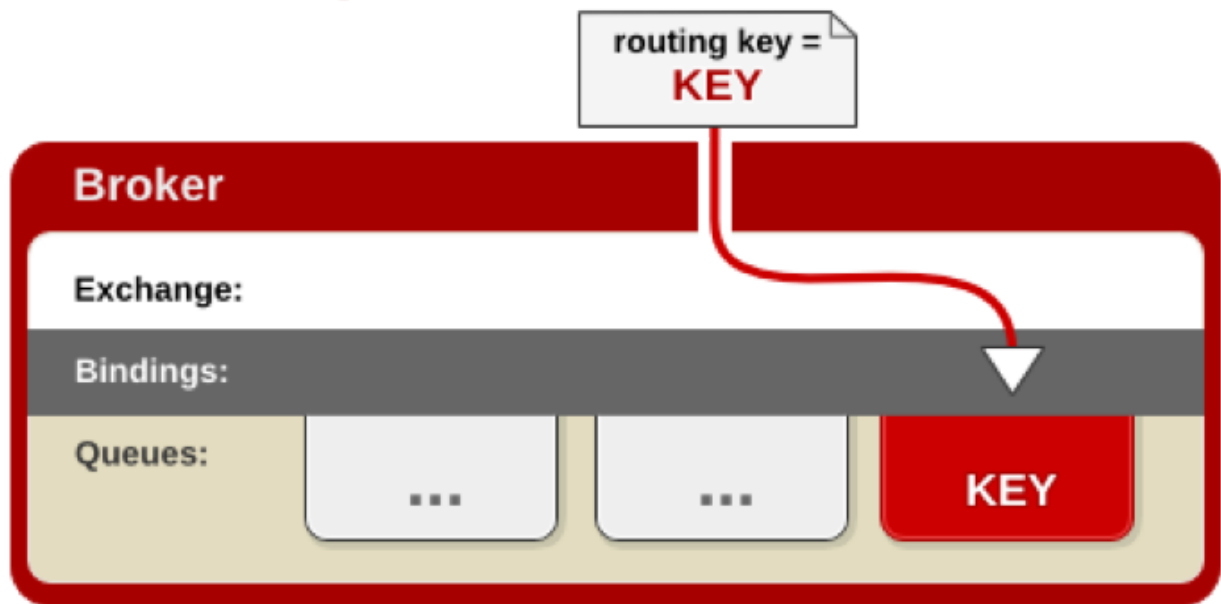
<https://www.rabbitmq.com/tutorials/tutorial-four-java.html>

生产者将消息发送到了 type 为 direct 模式的交换机

消费者的队列在将自己绑定到路由的时候会给自己绑定一个 key

只有消费者发送对应 key 格式的消息时候 队列才会收到消息

Direct Exchange



4.6.1 生产者

```
public class Send {

    private final static String EXCHANGE_NAME = "test_exchange_direct";//路由名字

    public static void main(String[] argv) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 声明exchange, type 是direct
        channel.exchangeDeclare(EXCHANGE_NAME, "direct");

        // 消息内容
        String message = "Hello world!";
        channel.basicPublish(EXCHANGE_NAME, "abc", null, message.getBytes()); //发送 key 为
        // abc的消息
        System.out.println(" 发送消息'" + message + "'");

        channel.close();
    }
}
```

```
        connection.close();  
    }  
}
```

4.6.2 消费者1

```
public class Recv {  
    private static final String EXCHANGENAME = "test_exchange_direct";  
    private static final String QUEUENAME = "directqueue1";  
  
    public static void main(String[] args) throws Exception {  
        Connection connection = ConnectionUtils.getConnection();//获取连接  
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的 statement\  
        //交换机必须先声明后使用,不管是谁声明的,队列必须先存在  
        channel.exchangeDeclare(EXCHANGENAME, "direct");  
        //参数1 队列的名字  
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放  
        //到数据库中,当服务器重启后会重新读取数据  
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有  
        //了,只能由一个消费者能访问该队列  
        channel.queueDeclare(QUEUENAME, false, false, false, null);  
        // 绑定队列到交换机,绑定自己的关键字 key 为key,注意在绑定到指定路由(交换机)的时候,该路由必须存在,  
        //也就是我们必须先由发送者创建一个路由才可以  
        channel.queueBind(QUEUENAME, EXCHANGENAME, "abc");  
        //如果要绑定多个 key 多次执行即可  
        channel.queueBind(QUEUENAME, EXCHANGENAME, "abcd");//将制定的队列绑定到指定的交换机上面  
        channel.queueBind(QUEUENAME, EXCHANGENAME, "asdf");//将制定的队列绑定到指定的交换机上面  
        channel.basicQos(1);//声明每次处理一个消息,只有处理完成并应答服务器之后才会处理下一条消息  
        DefaultConsumer defaultConsumer=new DefaultConsumer(channel){  
            @Override  
            public void handleDelivery(String consumerTag, Envelope envelope,  
AMQP.BasicProperties properties, byte[] body) throws IOException {  
                System.out.println("消费者1收到的内容是:"+new String(body));  
  
                try {  
                    Thread.sleep(10);  
                } catch (InterruptedException e) {  
                    e.printStackTrace();  
                }  
                //应答,告诉服务器我收到消息了  
                channel.basicAck(envelope.getDeliveryTag(), false);  
            }  
        };  
        // 监听队列, 手动返回完成  
        channel.basicConsume(QUEUENAME, false, defaultConsumer);  
    }  
}
```

4.6.3 消费者2

```
public class Recv2 {

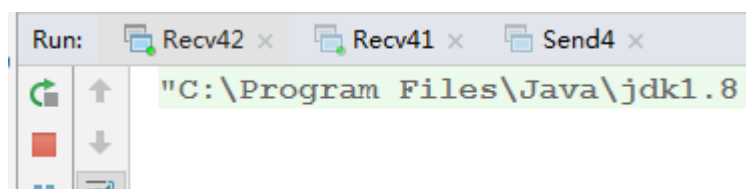
    private static final String EXCHANGENAME = "test_exchange_direct";
    private static final String QUEUENAME = "directqueue2";

    public static void main(String[] args) throws Exception {
        Connection connection = ConnectionUtils.getConnection();//获取连接
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的 statement\
        //交换机必须先声明后使用,不管是谁声明的,队列必须要先存在
        channel.exchangeDeclare(EXCHANGENAME, "direct");
        //参数1 队列的名字
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放
        //到数据库中,当服务器重启后会重新读取数据
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有
        //了,只能由一个消费者能访问该队列
        channel.queueDeclare(QUEUENAME, false, false, false, null);
        // 绑定队列到交换机,绑定自己的关键字 key 为key,注意在绑定到指定路由(交换机)的时候,该路由必须存在,
        //也就是我们必须先由发送者创建一个路由才可以
        channel.queueBind(QUEUENAME, EXCHANGENAME, "abc");
        channel.basicQos(1);//声明每次处理一个消息,只有处理完成并应答服务器之后才会处理下一条消息
        DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
            AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者2222收到的内容是:"+new String(body));
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                //应答,告诉服务器我收到消息了
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };
        // 监听队列, 手动返回完成
        channel.basicConsume(QUEUENAME, false, defaultConsumer);
    }
}
```

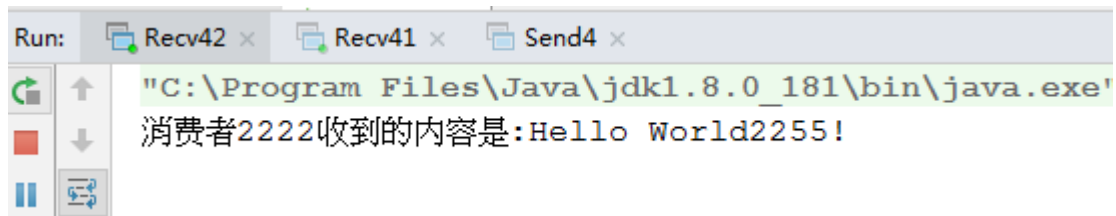
4.6.4 测试

启动消费者1,消费者2

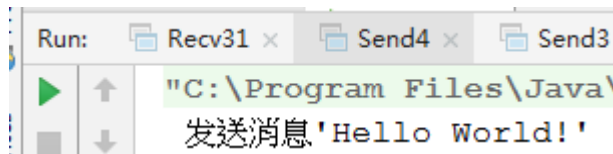
key和发送者key不相同, 收不到消息。



通过修改发送者代码中的 key 来多次执行测试,发现可以分别收到不同消息,如果监听了相同的 key 可以一起收到消息



启动发送者创建路由



服务端查看:

Overview	Connections	Channels	Exchanges	Queues	Admin
/	amq.direct	direct	D		
/	amq.fanout	fanout	D		
/	amq.headers	headers	D		
/	amq.match	headers	D		
/	amq.rabbitmq.trace	topic	D I		
/	amq.topic	topic	D		
/qf	(AMQP default)	direct	D	0.00/s	0.00/s
/qf	amq.direct	direct	D		
/qf	amq.fanout	fanout	D		
/qf	amq.headers	headers	D		
/qf	amq.match	headers	D		
/qf	amq.rabbitmq.trace	topic	D I		
/qf	amq.topic	topic	D		
/qf	qf_exchange	fanout		0.00/s	0.00/s
/qf	qf_exchange222	fanout			
/qf	test_exchange_direct	direct		0.00/s	

4.6.5 学生做题

测试路由模式

第五章 整合 spring

spring 对 amqp 做了支持,但是当前只实现了 rabbitmq

###5.1 spring 自动模式

####5.1.1 spring 配置文件

rabbitmq-context.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:rabbit="http://www.springframework.org/schema/rabbit"
    xsi:schemaLocation="http://www.springframework.org/schema/rabbit
        http://www.springframework.org/schema/rabbit/spring-rabbit-1.7.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd">

    <!-- 定义RabbitMQ的连接工厂 -->
    <rabbit:connection-factory id="connectionFactory"
        host="192.168.3.227" port="5672" username="test" password="test"
        virtual-host="/test" />
    <!-- 定义Rabbit模板, 指定连接工厂以及定义exchange
    如果要将消息发送到队列而不是交换机,则声明queue="" 而不是exchange=""
    -->
    <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
exchange="fanoutExchange" />
    <!-- <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"
        exchange="fanoutExchange" routing-key="foo.bar" /> -->
    <!-- MQ的管理, 包括队列、交换器等 -->
    <rabbit:admin connection-factory="connectionFactory" />
    <!-- 定义队列, 自动声明 -->
    <rabbit:queue name="myQueue" auto-declare="true"/>
    <!-- 定义交换器, 自动声明 -->
    <rabbit:fanout-exchange name="fanoutExchange" auto-declare="true" >
        <rabbit:bindings>
            <!--将下列队列绑定到当前交换机-->
            <rabbit:binding queue="myQueue"/>
        </rabbit:bindings>
    </rabbit:fanout-exchange>
    <!-- 通配符模式 -->
    <!--
    <rabbit:topic-exchange name="myExchange">
        <rabbit:bindings>
            <rabbit:binding queue="myQueue" pattern="foo.*" />
        </rabbit:bindings>
    </rabbit:topic-exchange>
    -->
    <!--路由设置 将队列绑定, 属于direct类型
    <rabbit:direct-exchange id="directExchange"
        name="directExchange" durable="true" auto-delete="false">
        <rabbit:bindings>
            <rabbit:binding queue="myQueue" key="${rabbitmq.system.out.log.error.mail}" />
        </rabbit:bindings>
    </rabbit:direct-exchange>
    -->
    <!-- 队列监听
    acknowledge = "manual" 属性为手动应答
```

```
-->
<rabbit:listener-container connection-factory="connectionFactory">
    <!--指定对应队列myQueue的监听为 foo 中的 listen 方法-->
    <rabbit:listener ref="foo" method="listen" queue-names="myQueue" />
</rabbit:listener-container>
<bean id="foo" class="com.qianfeng.rabbitmq.spring.Foo" />
</beans>
```

####5.1.2 接收者

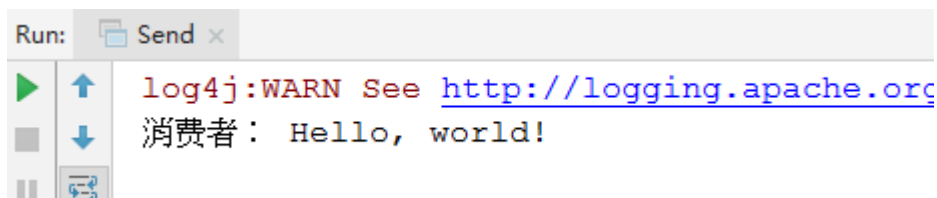
```
/**
 * 消费者类,任意类都可以
 */
public class Foo {

    //具体执行业务的方法
    public void listen(String foo) {
        System.out.println("消费者: " + foo);
    }
}
```

####5.1.3 测试类

```
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class TestMain {
    public static void main(final String... args) throws Exception {
        AbstractApplicationContext ctx = new ClassPathXmlApplicationContext(
            "classpath:spring/rabbitmq-context.xml");
        //RabbitMQ模板
        RabbitTemplate template = ctx.getBean(RabbitTemplate.class);
        //发送消息
        template.convertAndSend("Hello, world!");
        Thread.sleep(1000); // 休眠1秒
        ctx.destroy(); //容器销毁
    }
}
```

####5.1.4 启动测试



发送后消息监听在控制台打印消费内容。

5.2 学生做题

Spring整合RabbitMQ

第六章 整合 springboot

6.1 pom.xml

```
<?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.qianfeng</groupId>
    <artifactId>rabbitmqdemo</artifactId>
    <version>1.0</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.6.RELEASE</version>
    </parent>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-amqp -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>

    </dependencies>
</project>
```

6.2 yml 配置

```
spring:
  rabbitmq:
    host: rabbitmq1.qfjava.cn
    port: 5672
    username: gp02
    password: gp02
    virtual-host: /gp02
```

6.3 sender 发送者

```
@Component
public class Sender {
    @Autowired
    private AmqpTemplate amqpTemplate;

    public void send(String message) {
        amqpTemplate.convertAndSend("wxpayqueue", message);
    }

    public void sendtoExchange(String message) {
        amqpTemplate.convertAndSend("wxpayfanout", "", message);
    }

    public void sendtoDirectExchange(String message) {
        amqpTemplate.convertAndSend("directgp02", "abc", message);
    }
}
```

6.4 消费者的监听器

springboot 通过 listener 来监听消息,不需要单独写连接等

```
@Component
@RabbitListener(queues = "wxpayqueue")//标记当前类是用来处理来自于wxpayqueue这个队列的消息的
public class MessageListener {
    @RabbitHandler//标记当前方法是用来处理消息的
    public void aaaaa(String message) {
        System.out.println("收到交换机上面发送的内容:  =>" + message);
    }
}
```

6.5 starter&config

```
import org.springframework.amqp.core.*;
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
@SpringBootApplication
public class StartApp {

    @Bean//创建了一个队列,队列在rabbitmq中的名字是dongwei,在spring中的名字叫xuezhi
    public Queue xuezhi() {
        Queue queue = new Queue("wxpayqueue");
    }
}
```

```
        return queue;
    }

    @Bean
    public ConnectionFactory connectionFactory() {
        CachingConnectionFactory cachingConnectionFactory = new CachingConnectionFactory();
        cachingConnectionFactory.setHost("rabbitmq1.qfjava.cn");
        cachingConnectionFactory.setPort(5672);
        cachingConnectionFactory.setUsername("gp02");
        cachingConnectionFactory.setPassword("gp02");
        cachingConnectionFactory.setVirtualHost("/gp02");
        return cachingConnectionFactory;
    }

    /**
     * 声明交换机,fanout 类型
     * @return
     */
    @Bean
    public FanoutExchange fanoutExchange() {
        FanoutExchange fanoutExchange = new FanoutExchange("wxpayfanout");
        return fanoutExchange;
    }

    /**
     * 将队列和交换机绑定
     * 此处使用的是一个队列,如果有多个队列,请注意方法队列的参数名字区分
     * @return
     */
    @Bean
    public Binding bindingFanoutExchange(Queue queue, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(queue).to(fanoutExchange);
    }

    /**
     * 声明交换机,direct 类型
     * @return
     */
    @Bean
    public DirectExchange directExchange() {
        DirectExchange directExchange = new DirectExchange("directgp02");
        return directExchange;
    }

    /**
     * 将队列和交换机绑定
     * 此处使用的是一个队列,如果有多个队列,请注意方法队列的参数名字区分
     * @return
     */
    @Bean
    public Binding bindingDirectExchange(Queue queue, DirectExchange directExchange) {
        return BindingBuilder.bind(queue).to(directExchange).with("abc");
    }
}
```

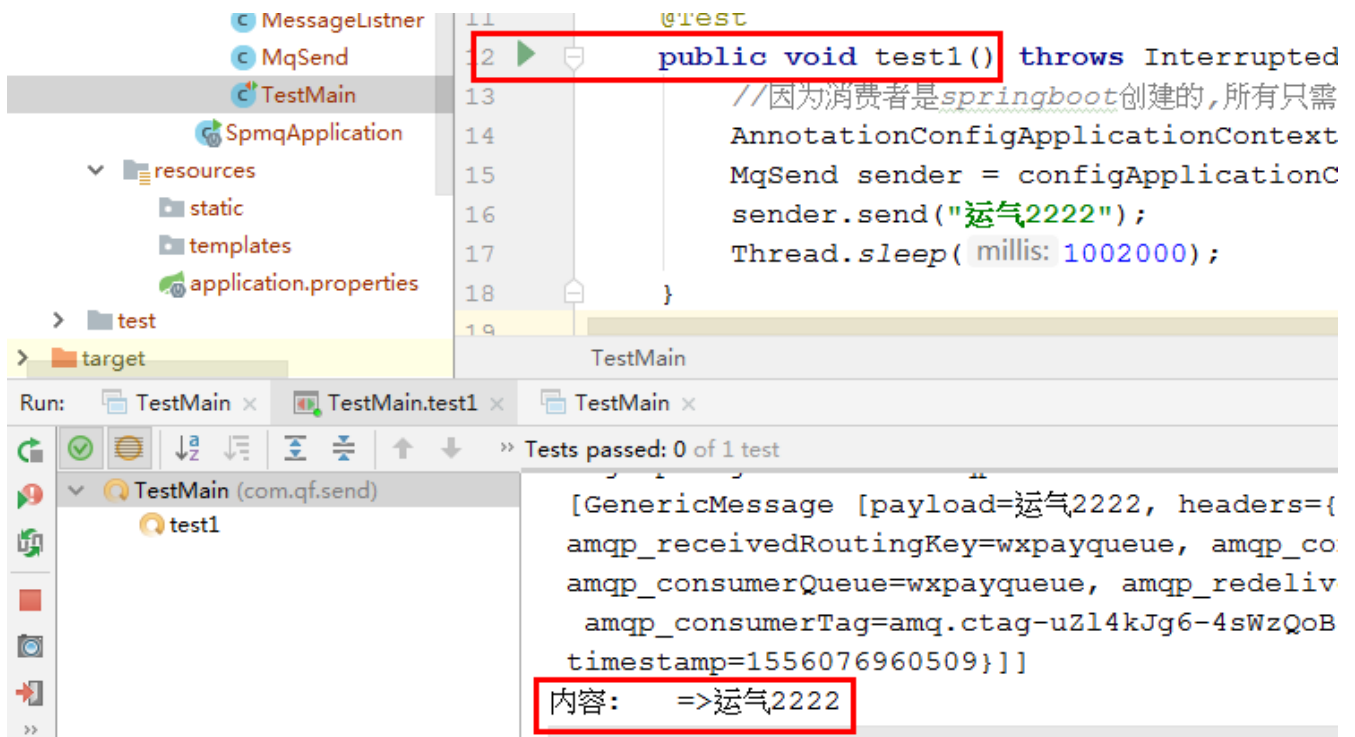
6.6 测试

```
import com.qf.SpmqApplication;
import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class TestMain {
    //测试消费发送与消费
    @Test
    public void test1() throws InterruptedException {
        //因为消费者是springboot创建的,所有只需要加载spring的容器就可以了,会自动创建对象,自动去链接服务器
        AnnotationConfigApplicationContext configApplicationContext = new
AnnotationConfigApplicationContext(StartApp.class);
        Sender sender = configApplicationContext.getBean(Sender.class);
        sender.send("运气2222");
        Thread.sleep(1002000);
    }

    @Test
    public void test2() throws InterruptedException {
        AnnotationConfigApplicationContext configApplicationContext = new
AnnotationConfigApplicationContext(StartApp.class);
        Sender sender = configApplicationContext.getBean(Sender.class);
        sender.sendtoExchange("测试1");
        Thread.sleep(1002000);
    }

    @Test
    public void test3() throws InterruptedException {
        AnnotationConfigApplicationContext configApplicationContext = new
AnnotationConfigApplicationContext(StartApp.class);
        Sender sender = configApplicationContext.getBean(Sender.class);
        sender.sendtoDirectExchange("测试2");
        Thread.sleep(1002000);
    }
}
```

控制台输出:



6.7 学生做题

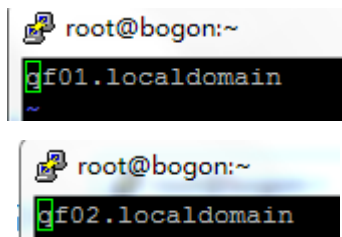
Springboot整合RabbitMQ

第七章 RabbitMQ集群

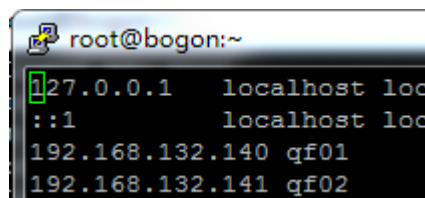
7.1 配置hosts文件

更改三台MQ节点的计算机名分别为mq01、mq02 和mq03，然后修改hosts配置文件

vim /etc/hostname //其他两台相同 mq01.localdomain



vi /etc/hosts 192.168.100.143 mq01 //注意不能带.注意-主机名称也要更改 192.168.100.144 mq02



执行reboot重启，即可看到主机名更改成功

```
root@qf01:~  
login as: root  
root@192.168.132.140's pass  
Last login: Sat Apr 27 17:40:5  
[root@qf01 ~]# ping qf01
```

再ping名称:(能通说明正常)

```
[root@qf01 ~]# ping qf01  
PING qf01 (127.0.0.1) 56(84) bytes of data.  
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.168 ms  
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.074 ms  
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.074 ms
```

7.2两个节点分别启动rabbitmq-server

启动服务端:

cd /opt/rabbitmq_server-3.7.0/sbin

./rabbitmq-server -detached

```
[root@qf01 sbin]# ./rabbitmq-server -detached  
Warning: PID file not written; -detached was passed.  
[root@qf01 sbin]# ps -aux |grep rabbit  
root      47557 45.9   4.1 2349932 76656 ?        Ssl   17:45   0:05 /usr/lib64/erlang/erts-9.3.3.6/bin/beam.smp -W w -A 64 -P 1048576 -t 5000000  
b -zdbbl 128000 -K true -- -root /usr/lib64/erlang -programe erl -- -home /root -- -pa /opt/rabbitmq_server-3.7.0/ebin -noshell -noinput -s rab  
t -sname rabbit@qf01 -boot start_sasl -kernel inet default_connect_options [{(nodelay,true)] -sasl errlog_type error -sasl sasl_error_logger fal  
bit lager_log_root "/opt/rabbitmq_server-3.7.0/var/log/rabbitmq" -rabbit lager default_file "/opt/rabbitmq_server-3.7.0/var/log/rabbitmq/rabbit  
og" -rabbit lager upgrade_file "/opt/rabbitmq_server-3.7.0/var/log/rabbitmq/rabbit@qf01 upgrade.log" -rabbit enabled_plugins_file "/opt/rabbitmq  
r-3.7.0/etc/rabbitmq/enabled_plugins" -rabbit plugins_dir "/opt/rabbitmq_server-3.7.0/plugins" -rabbit plugins_expand_dir "/opt/rabbitmq_server  
var/lib/rabbitmq/mnesia/rabbit@qf01-plugins-expand" -os_mon start_cpu_sup false -os_mon start_diskup false -os_mon start_memsup false -mnesia  
pt/rabbitmq_server-3.7.0/var/lib/rabbitmq/mnesia/rabbit@qf01" -kernel inet_dist_listen_min 25672 -kernel inet_dist_listen_max 25672 -noshell -n  
root      47656 0.0   0.0 112724   988 pts/0    R+    17:45   0:00 grep --color=auto rabbit
```

启动管理界面:

./rabbitmq-plugins list //查看插件安装情况

```
[root@qf01 sbin]# ./rabbitmq-plugins list  
Configured: E = explicitly enabled; e = implicitly enabled  
| Status: * = running on rabbit@qf01  
|/  
[ ] rabbitmq_amqp1_0          3.7.0  
[ ] rabbitmq_auth_backend_cache 3.7.0  
[ ] rabbitmq_auth_backend_http 3.7.0  
[ ] rabbitmq_auth_backend_ldap 3.7.0  
[ ] rabbitmq_auth_mechanism_ssl 3.7.0  
[ ] rabbitmq_consistent_hash_exchange 3.7.0  
[ ] rabbitmq_event_exchange    3.7.0  
[ ] rabbitmq_federation         3.7.0  
[ ] rabbitmq_federation_management 3.7.0  
[ ] rabbitmq_ims_topic_exchange 3.7.0  
[ ] rabbitmq_management         3.7.0  
[ ] rabbitmq_management_agent   3.7.0
```

启用rabbitmq_management服务插件,监控的端口是15672,查看监听端口如果有就不需再启用

./rabbitmq-plugins enable rabbitmq_management

```
[root@qf01 sbin]# ./rabbitmq-plugins enable rabbitmq_management
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@qf01...
The following plugins have been enabled:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
started 3 plugins.
```

7.3拷贝erlang.cookie

Rabbitmq的集群是依附于erlang的集群来工作的,所以必须先构建起erlang的集群景象。Erlang的集群中各节点是由过程一个magic cookie来实现的,这个cookie存放在\$HOME/.erlang.cookie中, 文件是400的权限。所以必须保证各节点cookie一致,不然节点之间就无法通信。

说明:

使用解压缩安装方式(二进制安装或者编译安装), 那么该文件存在位置为\$HOME目录下。即\$HOME/.erlang.cookie。使用root安装, 则位置为: /root/.erlang.cookie, 其他用户为/home/用户名/.erlang.cookie。

使用rpm包方式安装, 那么这个文件会存在于/var/lib/rabbitmq目录下。

启动rabbitmq之后会在/root下生成一个.erlang.cookie隐藏文件

```
cat /root/.erlang.cookie
```

```
[root@qf01 rabbitmq]# cat /root/.erlang.cookie
RAJHHPAYFRANODYLFLQI[root@qf01 rabbitmq]#
```

用scp的方式将mq01节点的.erlang.cookie的值复制到其他1-2个节点中。

```
scp /root/.erlang.cookie root@192.168.132.141:/root/
```

```
[root@qf01 rabbitmq]# cat /root/.erlang.cookie
RAJHHPAYFRANODYLFLQI[root@qf01 rabbitmq]# scp /root/.erlang.cookie root@192.168.132.141:/root/
root@192.168.132.141's password:
.erlang.cookie
```

服务器二的cookie:(确认两台服务器的cookie一致)

```
[root@qf02 rabbitmq_server-3.7.0]# cat /root/.erlang.cookie
RAJHHPAYFRANODYLFLQI[root@qf02 rabbitmq_server-3.7.0]#
```

关闭服务器二的rabbitmq,然后重启(加载了新cookie)。重启后查看状态:

./rabbitmqctl status 如果这步报错, 说明修改的cookie有问题。

```
[root@qf02 sbin]# ./rabbitmqctl status
Status of node rabbit@qf02 ...
[{pid,58711},
 {running_applications,
  [{rabbitmq_management,"RabbitMQ Management Console","3.7.0"},
   {rabbitmq_management_agent,"RabbitMQ Management Agent","3.7.0"},
   {amqp_client,"RabbitMQ AMQP Client","3.7.0"},
   {rabbitmq_web_dispatch,"RabbitMQ Web Dispatcher","3.7.0"},
   {rabbit,"RabbitMQ","3.7.0"},
   {rabbit_common,
    "Modules shared by rabbitmq-server and rabbitmq-erlang-client",
    "3.7.0"}],
```

7.4将mq02作为内存节点加入mq01节点集群中.

在mq02执行如下命令:

cd /opt/rabbitmq_server-3.7.0/sbin

在第二台服务器中加入到集群中来:

停止服务 ./rabbitmqctl stop_app

```
[root@qf02 sbin]# ./rabbitmqctl stop_app
Stopping rabbit application on node rabbit@qf02 ...
```

加入集群:./rabbitmqctl join_cluster rabbit@qf01

或./rabbitmqctl join_cluster --ram rabbit@qf01 //加入到磁盘节点

```
[root@qf02 sbin]# ./rabbitmqctl join_cluster rabbit@qf01
Clustering node rabbit@qf02 with rabbit@qf01
```

启动rabbit应用:./rabbitmqctl start_app

```
[root@qf02 sbin]# ./rabbitmqctl start_app
Starting node rabbit@qf02 ...
completed with 3 plugins.
```

(1) 默认rabbitmq启动后是磁盘节点, 在这个cluster命令下, mq02是内存节点, mq01是磁盘节点。(2) 如果要使mq02都是磁盘节点, 去掉--ram参数即可。(3) 如果想要更改节点类型, 可以使用命令**rabbitmqctl change_cluster_node_type disc(ram)**,前提是必须停掉rabbit应用

7.5查看集群状态

./rabbitmqctl cluster_status

```
[root@qf02 sbin]# ./rabbitmqctl cluster_status
Cluster status of node rabbit@qf02 ...
[{nodes, [{disc, [rabbit@qf01, rabbit@qf02]}]},
 {running_nodes, [rabbit@qf01, rabbit@qf02]},
 {cluster_name, <<"rabbit@qf01">>},
 {partitions, []},
 {alarms, [{rabbit@qf01, []}, {rabbit@qf02, []}]}]
[root@qf02 sbin]#
```

7.6登录rabbitmq web管理控制台

确保防火墙关闭或放行


关闭: `systemctl stop firewalld`

开机禁用: `systemctl disable firewalld`

查看状态: `systemctl status firewalld`

打开浏览器输入<http://192.168.132.140:15672/>, 输入默认的Username: guest, 输入默认的Password: guest

⚠ 不安全 | 192.168.132.140:15672



Username:

Password:

Login

rabbitmq从3.3.0开始禁止使用guest/guest权限通过除localhost外的访问

新增加的账户没有具体的虚拟主机可以操作:

Overview	Connections	Channels	Exchanges	Queues	Admin
Users					
▼ All users					
Filter: <input type="text"/> <input type="checkbox"/> Regex ?					
Name	Tags	Can access virtual hosts	Has password		
admin	administrator	No access	•		
guest	administrator	/	•		

新创建一个虚拟主机:

OverviewConnectionsChannelsExchangesQueuesAdmin

Cluster rabbitmq
User admin Lo

Virtual Hosts

▼ All virtual hosts

Filter: ☐ Regex ?

2 items, page size up to 100

Overview			Messages			Network		Message rates		+/-
Name	Users ?	State	Ready	Unacked	Total	From client	To client	publish	deliver / get	
/	guest	running	NaN	NaN	NaN					
/admin	admin	running	NaN	NaN	NaN					

▼ Add a new virtual host

Name:

Add virtual host

Users

Virtual Hosts

Policies

Limits

Cluster

第二个节点已同步添加:

← → ↻ ⓘ 不安全 | 192.168.132.141:15672/#/users

RabbitMQ™

3.7.0 Erlang 20.3.8.14

- Overview
- Connections
- Channels
- Exchanges
- Queues
- Admin**

Users

▼ All users

Filter: ☐ Regex ?

Name	Tags	Can access virtual hosts	Has password
admin	administrator	/admin	•
guest	administrator	/	•

登录后首页看到集群节点:

← → ↻ ⓘ 不安全 | 192.168.132.140:15672/#/

RabbitMQ™

3.7.0 Erlang 20.3.8.14

- Overview**
- Connections
- Channels
- Exchanges
- Queues
- Admin

Overview

▼ Totals

Queued messages **last minute** ?

Currently idle

Message rates **last minute** ?

Currently idle

Global counts ?

Connections: 0 Channels: 0 Exchanges: 8 Queues: 0 Consumers: 0

▼ Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space
rabbit@qf01	35 1024 available	0 829 available	371 1048576 available	83MB 729MB high watermark 48MB low watermark	5.7GB
rabbit@qf02	28 1024 available	0 829 available	364 1048576 available	70MB 729MB high watermark 48MB low watermark	5.1GB

根据界面提示创建一条队列

Overview Connections Channels Exchanges **Queues** Admin

Overview					Messages			Message rates		
Virtual host	Name	Node	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/admin	java01	rabbit@qf01	D	idle	0	0	0			

▼ Add a new queue

Virtual host: /admin 选择对应用户可以操作的虚拟主机

Name: java01 输入队列名称，后面的设置默认

Durability: Durable

Node: rabbit@qf01

Auto delete: ? No

Arguments: = String

Add Message TTL ? | Auto expire ? | Max length ? | Max length bytes ? | Overflow behaviour ? | Dead letter exchange ? | Dead letter routing key ? | Maximum priority ? | Lazy mode ? | Master locator ?

Add queue

在服务器二上，同样的队列也出现:

← → ↺ ⓘ 不安全 | 192.168.132.141:15672/#/queues

RabbitMQ 3.7.0 Erlang 20.3.8.14

Overview Connections Channels Exchanges **Queues** Admin

Queues

▼ All queues (1)

Pagination

Page 1 ▼ of 1 - Filter: ☐ Regex ?

Overview					Messages			Message rates		
Virtual host	Name	Node	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/admin	java01	rabbit@qf01	D	idle	0	0	0			

在RabbitMQ集群集群中，必须至少有一个磁盘节点，否则队列元数据无法写入到集群中，当磁盘节点宕掉时，集群将无法写入新的队列元数据信息。

7.7 RabbitMQ镜像集群配置

上面已经完成RabbitMQ默认集群模式，但并不保证队列的高可用性，尽管交换机、绑定这些可以复制到集群里的任何一个节点，但是队列内容不会复制。虽然该模式解决一项目组节点压力，但队列节点宕机直接导致该队列无法应用，只能等待重启，所以要想在队列节点宕机或故障也能正常应用，就要复制队列内容到集群里的每个节点，必须要创建镜像队列。

镜像队列是基于普通的集群模式的，然后再添加一些策略，所以你还是得先配置普通集群，然后才能设置镜像队列，我们就以上面的集群接着做。

设置的镜像队列可以通过开启的网页的管理端，也可以通过命令，这里说的是其中的网页设置方式。

7.7.1.创建rabbitmq策略

在mq01节点的控制台上创建策略

- (1) 点击admin菜单->右侧的Policies选项->左侧最下下边的Add/update a policy。
- (2) 按照图中的内容根据自己的需求填写。

▼ Add / update a policy

Virtual host: 用户可以访问的虚拟主机

Name: 名称自定义

Pattern:

Apply to:

Priority:

Definition: =

HA **HA mode** ? HA params ? | HA sync mode ?

Federation Federation upstream set ? | Federation upstream ?

Queues Message TTL | Auto expire | Max length | Max length bytes | Overflow behaviour
Dead letter exchange | Dead letter routing key
Lazy mode | Master Locator

Exchanges Alternate exchange ?

Add policy

成功后:

Overview Connections Channels Exchanges Queues **Admin**

User admin Log out

Policies

▼ User policies

Filter: ☐ Regex ? 1 item, page size up to 100

Virtual Host	Name	Pattern	Apply to	Definition	Priority
/admin	qfpol	^	all	ha-mode: all	0

Users

Virtual Hosts

Policies

Limits

Cluster

- Name:策略名称
- Pattern: 匹配的规则, ^a表示匹配a开头的队列, 如果是匹配所有的队列, 那就是^.
- Definition:使用ha-mode模式中的all, 也就是同步所有匹配的队列。问号链接帮助文档。

7.7.2分别登陆mq02、mq03两个节点的控制台

可以看到上面添加的这个策略，如图所示：

<http://192.168.132.141:15672>

Name	Pattern	Apply to	Definition	Priority
mypol	^	all	ha-mode: all	0

7.7.3添加队列

在mq01节点的控制台上添加队列（1）点击Queues菜单->左侧下边的Add a new queue（2）输入Name和Arguments参数的值，别的值默认即可

Add a new queue

Virtual host: /admin

Name: java02 *

Durability: Durable

Node: rabbit@qf01

Auto delete: No

Arguments: x-ha-policy = all

String

String

Add Message TTL ? | Auto expire ? | Max length ? | Max length bytes ?

Dead letter exchange ? | Dead letter routing key ? | Maximum priority ?

Lazy mode ? Master locator ?

Add queue

- Name: 队列名称
- Durability: 队列是否持久化: Durable (持久化保存), Transient (即时保存)
- Node: 消息队列的节点
- Auto delete: 自动删除
- Arguments: 使用的策略类型 x-ha-policy=all
镜像队列将会在整个集群中复制。当一个新的节点加入后, 也会在这个节点上复制一份。

Overview

Connections

Channels

Exchanges

Queues

Admin

All queues (2)

Pagination

Page

1

of 1

- Filter:

☐ Regex ?

Overview					Messages			Message rates		
Virtual host	Name	Node	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/admin	java01	rabbit@qf01	+1 D qfpol	idle	0	0	0			
/admin	java02	rabbit@qf01	+1 D Args qfpol	idle	0	0	0			

将鼠标指向+1 | +2可以显示出另外一台 | 两台消息节点。

说明: qf01创建的队列也同步到了qf02

java02	rabbit@qf01	+1	D Args mypol	idle	0
Add a new queue Synchronised mirrors: rabbit@qf02					

7.7.4创建消息

(1) 服务器一上点击java02队列按钮

Overview	Connections	Channels	Exchanges	Queues
Queue java02				
Overview				

(2) 拖动滚动条, 点击publish message

- ▶ Consumers
- ▶ Bindings
- ▶ **Publish message**
- ▶ Get messages
- ▶ Move messages
- ▶ Delete

(3) 填写相关内容

▼ Publish message

Message will be published to the default exchange with routing key **java02**, routing it to this queue.

Delivery mode: 2 - Persistent ▼

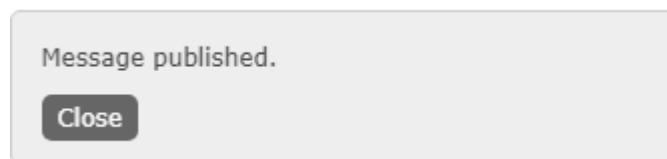
Headers: ?	<input type="text" value="a"/>	=	<input type="text" value="123"/>	String ▼
	<input type="text"/>	=	<input type="text"/>	String ▼

Properties: ?	<input type="text" value="message_id"/>	=	<input type="text" value="1"/>
	<input type="text"/>	=	<input type="text"/>

Payload:

- 2-Persistent:表示持久化
- Headers:随便填写即可
- Properties:点击问号, 选择一个消息ID号
- Payload:消息内容

(4) 点击Publish message按钮



点击queue按钮, 发现java02队列的Ready和Total中多了一条消息记录。

Overview

Connections

Channels

Exchanges

Queues

Admin

Page

1

of 1

- Filter:

☐

Regex

?

Overview					Messages			Message rate	
Name	Node	Features	State	Ready	Unacked	Total	incoming	deliver	
java01	rabbit@qf01	+1	D mypol	idle	0	0	0		
java02	rabbit@qf01	+1	D Args mypol	idle	1	0	1	0.00/s	

qf02服务器也会同步。

7.7.5破坏性测试

(1) 将mq01节点的服务关闭，再通过mq02和mq03查看消息记录是否还存在。

```
./rabbitmqctl stop_app //停掉mq01的rabbit应用
```

```
[root@qf01 sbin]# ./rabbitmqctl stop_app
Stopping rabbit application on node rabbit@qf01 ...
[root@qf01 sbin]#
```

qf02节点:

从中可以看到java02队列已经从之前的+1显示成没了，只是变成了一个节点,而且消息记录是存在的。

OverviewConnectionsChannelsExchangesQueuesAdmin

Queues

▼ All queues (2)

Pagination

Page 1 ▼ of 1 - Filter: ☐ Regex ?

Overview				Messages			Message rates	
Name	Node	Features	State	Ready	Unacked	Total	incoming	deliver / ge
java01	rabbit@qf02	D mypol	idle	0	0	0		
java02	rabbit@qf02	D Args mypol	idle	1	0	1		

(2) 将mq01的服务再启动起来

```
./rabbitmqctl start_app //启动mq01、mq02的rabbit应用
```



```
[root@qf01 sbin]# ./rabbitmqctl start_app
Starting node rabbit@qf01 ...
completed with 3 plugins.
[root@qf01 sbin]#
```

查看队列:

← → ↻ ⓘ 不安全 | 192.168.132.140:15672/#/queues

RabbitMQ™

3.7.4 Erlang 20.3.8.14

Overview Connections Channels Exchanges **Queues** Admin

Page 1 of 1 - Filter: ☐ Regex ?

Overview					Messages	
Name	Node	Features	State	Ready	Unacked	T
java01	rabbit@qf02 +1	D mypol	idle	0	0	
java02	rabbit@qf02 +0 +1	D Args mypol	idle	1	0	

+1又变成了蓝色,集群成功。鼠标指上去只是有异步状态。

Overview					Messages	
Name	Node	Features	State	Ready	Unacked	T
java01	rabbit@qf02 +1	D mypol	idle	0	0	
java02	rabbit@qf02 +0 +1	D Args mypol	idle	1	0	

► Add a new queue Unsynchronised mirrors: rabbit@qf01

非同步的Slave (unsynchronised slave)

在rabbitmq中同步(synchronised)是用来描述master和slave之间的数据状态是否一致的。

节点重新加入(rejoin) 到镜像队列时, 也会出现非同步的情况。

采取的解决办法是选择在mq02节点上执行同步命令。

```
./rabbitmqctl sync_queue java02 -p '/admin' //同步java02队列
```

```
[root@qf02 sbin]# ./rabbitmqctl sync_queue java02
Synchronising queue 'java02' in vhost '/' ...
Error:
not_found
[root@qf02 sbin]# ./rabbitmqctl sync_queue java02 -p '/admin'
Synchronising queue 'java02' in vhost '/admin' ...
[root@qf02 sbin]#
```

注意:创建的用户需要指定虚拟主机。

再次查看,变为同步状态。

Overview
Connections
Channels
Exchanges
Queues

Queues

▼ All queues (2)

Pagination

Page 1 ▼ of 1 - Filter: ☐ Regex ?

Overview				Messages	
Name	Node	Features	State	Ready	Unacked
java01	rabbit@qf02 +1	D mypol	idle	0	0
java02	rabbit@qf02 +1	D Args mypol	idle	3	0

► Add a new queue

Synchronised mirrors: rabbit@qf01

7.8学生做题

搭建RabbitMQ集群

第八章 消息确认与延迟机制

我们知道可以通过持久化（交换机、队列和消息持久化）来保障我们在服务器崩溃时，重启服务器消息数据不会丢失。但是我们无法确认当消息的发布者在将消息发送出去之后，消息到底有没有正确到达Broker代理服务器呢？如果不进行特殊配置的话，默认情况下发布操作是不会返回任何信息给生产者的，也就是默认情况下我们的生产者是不知道消息有没有正确到达Broker的。如果在消息到达Broker之前已经丢失的话，持久化操作也解决不了这个问题，因为消息根本就未到达代理服务器，这个是没有办法进行持久化的，那么当我们遇到这个问题又该如何去解决呢？

这里就是我们讲解到的RabbitMQ中的消息确认机制，通过消息确认机制我们可以确保我们的消息可靠送达到我们的用户手中，即使消息丢失掉，我们也可以通过进行重复分发确保用户可靠收到消息。

我们讲解的RabbitMQ消息确认机制，主要包括两个方面，因为RabbitMQ为我们提供了两种方式：

通过AMQP事务机制实现，这也是AMQP协议层面提供的解决方案；

通过将channel设置成confirm模式来实现；

8.1 AMQP事务

我们知道事务可以保证消息的传递，使得可靠消息最终一致性。接下来我们先来探究一下RabbitMQ的事务机制。

RabbitMQ中与事务有关的主要有三个方法：

txSelect() 将当前channel设置为transaction模式

txCommit() 提交当前事务

txRollback() 事务回滚

当我们使用txSelect提交开始事务之后，我们就可以发布消息给Broker代理服务器，如果txCommit提交成功了，则消息一定到达了Broker了，如果在txCommit执行之前Broker出现异常崩溃或者由于其他原因抛出异常，这个时候我们便可以捕获异常通过txRollback方法进行回滚事务了。

8.1.1事务发送者

```
package com.qf.tx;

import com.qf.util.RabConnFactory;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

public class TxSend {
    public static void main(String[] args) throws IOException, TimeoutException {
        String QUEUE_NAME = "test_queue_tx";
        //得到链接
        Connection conn = RabConnFactory.getConn();
        //根据链接创建通道
        Channel channel = conn.createChannel();
        // 声明一个队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        try{
            channel.txSelect(); // 开启事务
            // 发送消息
            String message = "hello, tx message";
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
            System.out.println(" [x] Sent message : '" + message + "'");
            //测试异常
            //int i = 1/0;
            channel.txCommit(); // 提交事务
        }catch (Exception e){
            channel.txRollback(); // 回滚事务
            System.out.println("send message txRollback");
        }
        channel.close();
        conn.close();
    }
}
```

8.1.2消费者

```
package com.qf.tx;

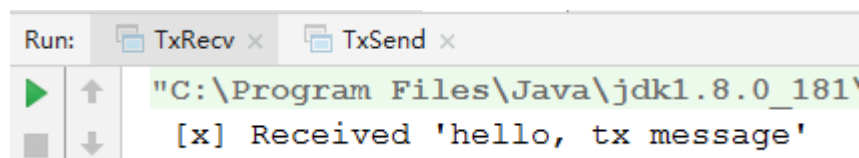
import com.qf.util.RabConnFactory;
import com.rabbitmq.client.*;
```

```
import java.io.IOException;
import java.util.concurrent.TimeoutException;
public class TxRecv {
    public static void main(String[] args) throws IOException, TimeoutException {
        String QUEUE_NAME = "test_queue_tx";
        //得到链接
        Connection conn = RabConnFactory.getConn();
        //根据链接创建通道
        Channel channel = conn.createChannel();
        // 申明要消费的队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
        // 消费消息
        channel.basicConsume(QUEUE_NAME, false, new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                // 接收到的消息
                String message = new String(body);
                System.out.println(" [x] Received '" + message + "'");
            }
        });
    }
}
```

8.1.3测试

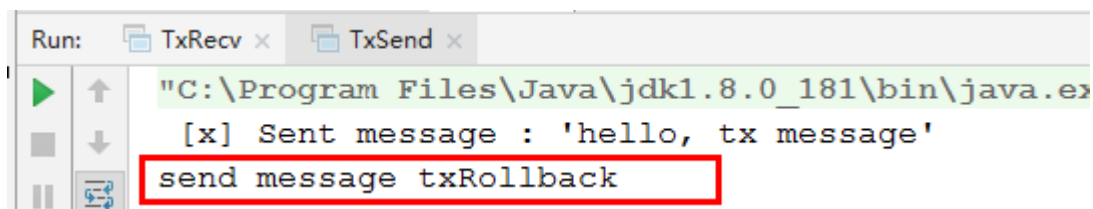
开始消费者

开启发送者：正常收发



生产者加上异常代码:

发送失败,消费者并没有接收到消息。说明生产者的消息回滚了，事务生效。



缺点:

事务确实能够解决producer与broker之间消息确认的问题，只有消息成功被broker接受，事务提交才能成功，否则我们便可以在捕获异常进行事务回滚操作同时进行消息重发，但是使用事务机制的话会降低RabbitMQ的性能，那么有没有更好的方法既能保障producer知道消息已经正确送到，又能基本上不带来性能上的损失呢？从AMQP协议的层面看是没有更好的方法，但是RabbitMQ提供了一个更好的方案，即将channel信道设置成confirm模式。

8.2 confirm模式

开启confirm模式的方法：

生产者通过调用channel的confirmSelect方法将channel设置为confirm模式，(注意一点，已经在transaction事务模式的channel是不能再设置成confirm模式的，即这两种模式是不能共存的)，如果没有设置no-wait标志的话，broker会返回confirm.select-ok表示同意发送者将当前channel信道设置为confirm模式

生产者实现confirm模式有两种编程方式：

(1): 普通confirm模式，每发送一条消息，调用waitForConfirms()方法等待服务端confirm，这实际上是一种串行的confirm，每publish一条消息之后就等待服务端confirm，如果服务端返回false或者超时时间内未返回，客户端进行消息重传；

(2): 批量confirm模式，每发送一批消息之后，调用waitForConfirms()方法，等待服务端confirm，这种批量确认的模式极大的提高了confirm效率，但是如果一旦出现confirm返回false或者超时的情况，客户端需要将这一批次的消息全部重发，这会带来明显的重复消息，如果这种情况频繁发生的话，效率也会不升反降；

8.2.1 普通confirm模式

生产者每发送一条消息就等待broker回应确认消息

```
package com.qf.confirm;

import com.qf.util.RabConnFactory;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.MessageProperties;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

public class ConfirmSend {

    public static void main(String[] args) throws IOException, TimeoutException,
        InterruptedException {
        String exchangeName = "confirmexchange";
        String queueName = "confirm_queue1";
        String routingKey = "qf";
        //得到链接
        Connection conn = RabConnFactory.getConn();
        //根据链接创建通道
        Channel channel = conn.createChannel();
        //创建exchange
        channel.exchangeDeclare(exchangeName, "direct", true, false, null);
        //创建队列
        channel.queueDeclare(queueName, true, false, false, null);
        //绑定exchange和queue
        channel.queueBind(queueName, exchangeName, routingKey);
        channel.confirmSelect();
        //发送持久化消息
        String msg = "收藏功能2";
        //第一个参数是exchangeName(默认情况下代理服务器端是存在一个""名字的exchange的,
        //因此如果不创建exchange的话我们可以直接将该参数设置成"",如果创建了exchange的话
```

```
//我们需要将该参数设置成创建的exchange的名字),第二个参数是路由键
channel.basicPublish(exchangeName, routingKey, MessageProperties.PERSISTENT_BASIC,
msg.getBytes());
if(channel.waitForConfirms())
{
    System.out.println("发送成功");
}
conn.close();
}
}
```

8.2.2 批量confirm模式

这种模式生产者不是每发送一条就等待broker确认，而是发送一批

```
package com.qf.confirm;

import com.qf.util.RabConnFactory;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.MessageProperties;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

public class ConfirmSends {

    public static void main(String[] args) throws IOException, TimeoutException,
InterruptedException {
        String exchangeName = "confirmexchange";
        String queueName = "confirm_queue1";
        String routingKey = "qf";
        //得到链接
        Connection conn = RabConnFactory.getConn();
        //根据链接创建通道
        Channel channel = conn.createChannel();
        //创建exchange
        channel.exchangeDeclare(exchangeName, "direct", true, false, null);
        //创建队列
        channel.queueDeclare(queueName, true, false, false, null);
        //绑定exchange和queue
        channel.queueBind(queueName, exchangeName, routingKey);
        channel.confirmSelect();
        //发送持久化消息
        String msg = "收藏功能";
        for (int i = 0; i < 5; i++) {
            //第一个参数是exchangeName(默认情况下代理服务器端是存在一个""名字的exchange的,
            //因此如果不创建exchange的话我们可以直接将该参数设置成"",如果创建了exchange的话
            //我们需要将该参数设置成创建的exchange的名字),第二个参数是路由键key
            channel.basicPublish(exchangeName,
routingKey, MessageProperties.PERSISTENT_BASIC, (msg+i).getBytes());
        }
        channel.waitForConfirmsOrDie();
    }
}
```

```
        conn.close();
    }
}
```

8.2.3消费者

```
package com.qf.confirm;

import com.qf.util.RabConnFactory;
import com.rabbitmq.client.*;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

public class ConfirmRecv {

    public static void main(String[] args) throws IOException, TimeoutException {
        //定义交换机, 必须一致
        String excName = "confirmexchange";
        //定义队列的名称
        String queueName = "confirm_queuecon1";
        String routingKey = "qf";
        //得到链接
        Connection conn = RabConnFactory.getConn();
        //根据链接得到通道
        final Channel channel = conn.createChannel();
        //声明交换机
        channel.exchangeDeclare(excName, "direct", true, false, null);
        //声明队列
        channel.queueDeclare(queueName, true, false, false, null);
        //把队列绑定到交换机, 同时指定key
        channel.queueBind(queueName, excName, routingKey);
        //声明处理消息的
        channel.basicQos(1);
        //创建消费对象
        DefaultConsumer consumer = new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                //接受消息
                System.out.println("消费者"+new String(body,"utf-8"));
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                //应答
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };
        //开始监听
        channel.basicConsume(queueName, false, consumer);
    }
}
```

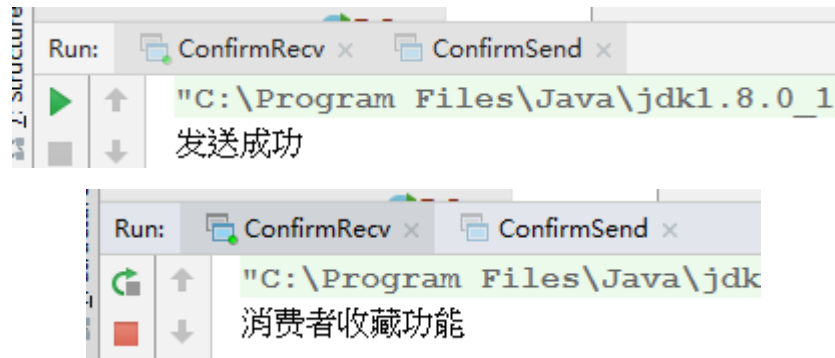
```
}  
}
```

8.2.4测试

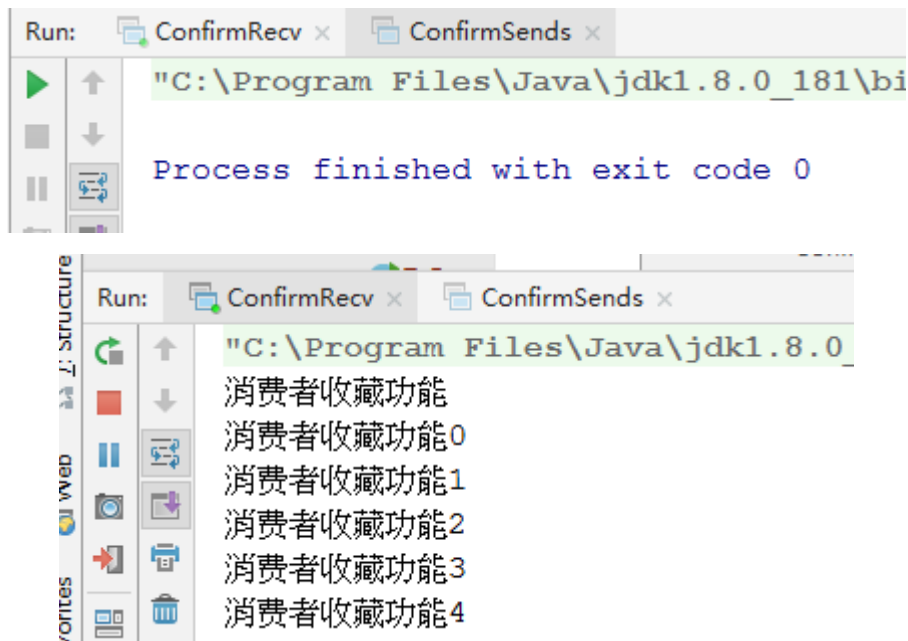
开启消费者

开启发送者

单个发送



多个发送:



8.3延迟队列

8.3.1延迟队列概念

延迟队列存储的对象肯定是对应的延时消息，所谓“延时消息”是指当消息被发送以后，并不想让消费者立即拿到消息，而是等待指定时间后，消费者才拿到这个消息进行消费。

RabbitMQ如何实现延迟队列？

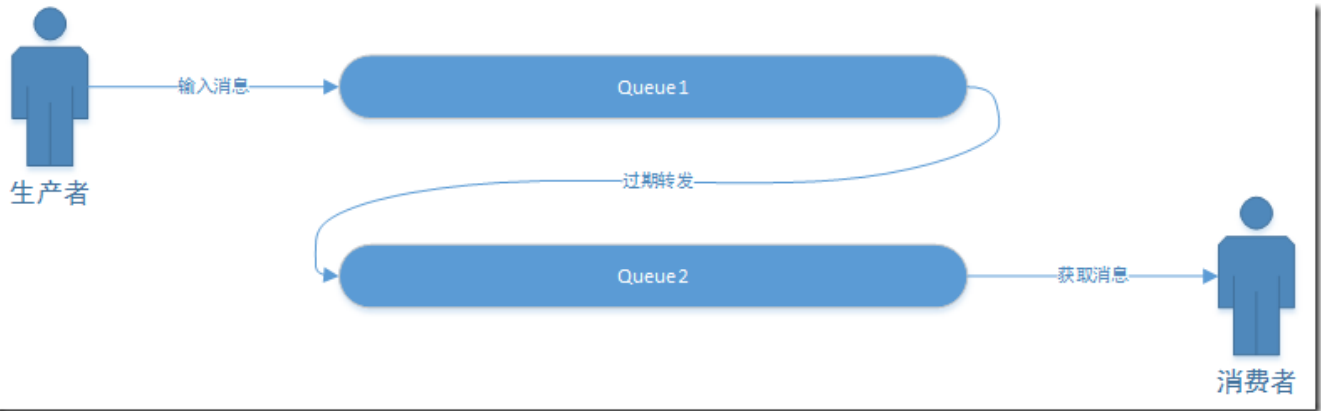
AMQP协议和RabbitMQ队列本身没有直接支持延迟队列功能，但是可以通过以下特性模拟出延迟队列的功能。

消息的TTL (Time To Live)

消息的TTL就是消息的存活时间。RabbitMQ可以对队列和消息分别设置TTL。对队列设置就是队列没有消费者连着的保留时间，也可以对每一个单独的消息做单独的设置。超过了这个时间，我们认为这个消息就死了，称之为死信。可以通过设置消息的expiration字段或者x-message-ttl属性来设置时间。

实现延迟队列

延迟任务通过消息的TTL来实现。我们需要建立2个队列，一个用于发送消息，一个用于消息过期后的转发目标队列。



生产者输出消息到Queue1，并且这个消息是设置有有效时间的，比如60s。消息会在Queue1中等待60s，如果没有消费者收掉的话，它就被转发到Queue2，Queue2有消费者，收到，处理延迟任务。

8.3.2延迟消息发送

```
package com.qf.service.delay;

import com.qf.service.util.RabConnFactory;
import com.rabbitmq.client.AMQQP;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.TimeoutException;

public class DelaySend {
    public static void main(String[] args) throws IOException, TimeoutException {
        String queueName="mydelaytopic3";
        Connection conn = RabConnFactory.getConn();
        Channel channel = conn.createChannel();
        Map<String, Object> arguments = new HashMap<>();
        arguments.put("x-dead-letter-exchange", "");
        arguments.put("x-dead-letter-routing-key", "deadLetterQueue");
        //通过队列属性设置，队列中所有消息都有相同的过期时间。
        arguments.put("x-message-ttl", 10000); //延时主要是这个参数
        channel.queueDeclare(queueName, true, false, false, arguments);
        //死信队列
        channel.queueDeclare("deadLetterQueue", true, false, false, null);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
```

```
String msg = "延时消息 发送时间:"+sdf.format(new Date());
AMQP.BasicProperties properties = new AMQP.BasicProperties();
properties.builder().expiration("10000");
channel.basicPublish("",queueName,properties,msg.getBytes());
channel.close();
conn.close();
}
}
```

8.3.3延迟消息消费

```
package com.qf.service.delay;

import com.qf.service.util.RabConnFactory;
import com.rabbitmq.client.*;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeoutException;

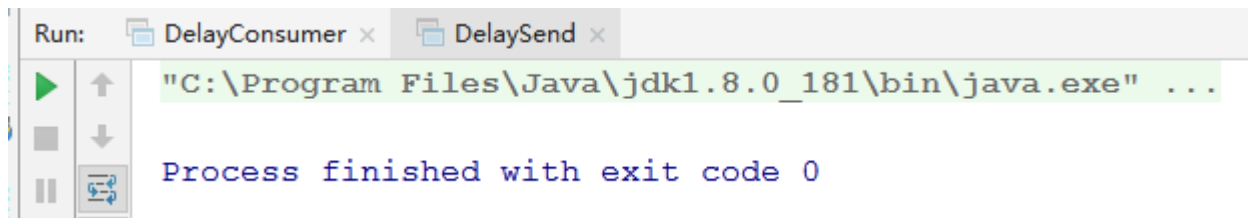
public class DelayConsumer {

    public static void main(String[] args) throws IOException, TimeoutException {
        String queueName="deadLetterQueue";
        Connection conn = RabConnFactory.getConn();
        final Channel channel = conn.createChannel();
        channel.queueDeclare(queueName,true,false,false,null);
        DefaultConsumer consumer = new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                System.out.println("消费者"+new String(body,"utf-8")+"接受时间"+sdf.format(new
Date()));

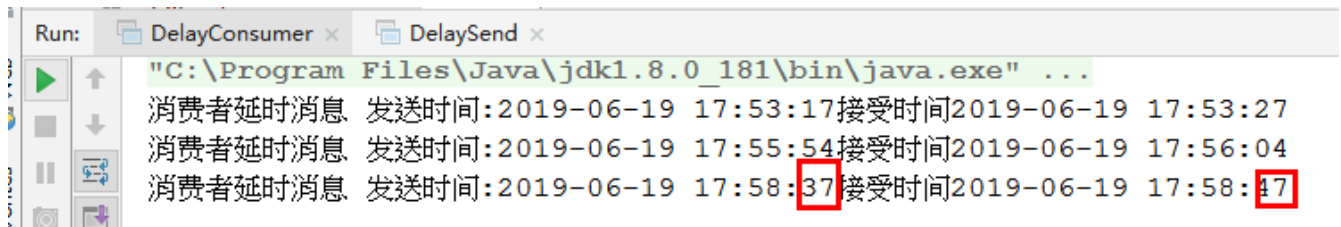
                //应答
                channel.basicAck(envelope.getDeliveryTag(),false);
            }
        };
        //开始监听
        channel.basicConsume(queueName,false,consumer);
    }
}
```

8.3.4测试

开启消费,开启发送:



过期时间到后:



8.4 学生做题

测试AMQP事务消息确认机制

测试confirm模式消息确认机制

测试延迟队列消息

第九章 MQ面试

9.1 rabbitmq与kafka对比

1)在应用场景方面

RabbitMQ,遵循AMQP协议, 由内在高并发的erlanng语言开发, 用在实时的对可靠性要求比较高的消息传递上。

kafka是Linkedin于2010年12月份开源的消息发布订阅系统,它主要用于处理活跃的流式数据,大数据量的数据处理上。

2)在架构模型方面

RabbitMQ遵循AMQP协议, RabbitMQ的broker由Exchange,Binding,queue组成, 其中exchange和binding组成了消息的路由键; 客户端Producer通过连接channel和server进行通信, Consumer从queue获取消息进行消费(长连接, queue有消息会推送到consumer端, consumer循环从输入流读取数据)。rabbitMQ以broker为中心; 有消息的确认机制。

kafka遵从一般的MQ结构, producer, broker, consumer, 以consumer为中心, 消息的消费信息保存的客户端consumer上, consumer根据消费的点, 从broker上批量pull数据; 无消息确认机制。

3)在吞吐量

kafka具有高的吞吐量, 内部采用消息的批量处理, zero-copy机制, 数据的存储和获取是本地磁盘顺序批量操作, 具有O(1)的复杂度, 消息处理的效率很高。

rabbitMQ在吞吐量方面稍逊于kafka, 他们的出发点不一样, rabbitMQ支持对消息的可靠的传递, 支持事务, 不支持批量的操作; 基于存储的可靠性的要求存储可以采用内存或者硬盘。

4)在可用性方面

rabbitMQ支持miror的queue，主queue失效，miror queue接管。

kafka的broker支持主备模式。

5)在集群负载均衡方面

kafka采用zookeeper对集群中的broker、consumer进行管理，可以注册topic到zookeeper上；通过zookeeper的协调机制，producer保存对应topic的broker信息，可以随机或者轮询发送到broker上；并且producer可以基于语义指定分片，消息发送到broker的某分片上。

rabbitMQ的负载均衡需要单独的loadbalancer进行支持。

9.2为什么使用消息队列

比较核心的有3个：解耦、异步、削峰

解耦：现场画个图来说明一下，A系统发送个数据到BCD三个系统，接口调用发送，那如果E系统也要这个数据呢？那如果C系统现在不需要了呢？现在A系统又要发送第二种数据了呢？

异步：现场画个图来说明一下，A系统接收一个请求，需要在自己本地写库，还需要在BCD三个系统写库，自己本地写库要3ms，BCD三个系统分别写库要300ms、450ms、200ms。最终请求总延时是 $3 + 300 + 450 + 200 = 953\text{ms}$ ，接近1s，用户体验非常慢。

削峰：每天0点到11点，A系统风平浪静，每秒并发请求数量就100个。结果每次一到11点~1点，每秒并发请求数量突然会暴增到1万条。但是系统最大的处理能力就只能是每秒钟处理1000个请求...

9.3如何保证消息队列的高可用

镜像集群模式,这种模式，才是所谓的rabbitmq的高可用模式，跟普通集群模式不一样的是，你创建的queue，无论元数据还是queue里的消息都会存在于多个实例上，然后每次你写消息到queue的时候，都会自动把消息到多个实例的queue里进行消息同步。

这样的话，好处在于，你任何一个机器宕机了，没事儿，别的机器都可以用。

9.4如何保证消息不被重复消费啊（幂等性）

首先就是比如rabbitmq、rocketmq、kafka，都有可能会出现消费重复消费的问题，正常。因为这问题通常不是mq自己保证的，是给你保证的。

幂等性，就一个数据，或者一个请求，给你重复来多次，你得确保对应的数据是不会改变的，不能出错。

其实还是得结合业务来思考：

(1) 比如你拿个数据要写库，你先根据主键查一下，如果这数据都有了，你就别插入了，update一下好吧

(2) 比如你是写redis，那没问题了，反正每次都是set，天然幂等性

(3) 比如你不是上面两个场景，那做的稍微复杂一点，你需要让生产者发送每条数据的时候，里面加一个全局唯一的id，类似订单id之类的东西，然后你这里消费到了之后，先根据这个id去比如redis里查一下，之前消费过吗？如果没有消费过，你就处理，然后这个id写redis。如果消费过了，那你就别处理了，保证别重复处理相同的消息即可。

9.5如何保证消息的可靠性传输（消息丢失问题）

1) 生产者弄丢了数据

生产者将数据发送到rabbitmq的时候，可能数据就在半路给搞丢了，因为网络啥的问题，都有可能。

此时可以选择用rabbitmq提供的事务功能，就是生产者发送数据之前开启rabbitmq事务（channel.txSelect），然后发送消息，如果消息没有成功被rabbitmq接收到，那么生产者会收到异常报错，此时就可以回滚事务（channel.txRollback），然后重试发送消息；如果收到了消息，那么可以提交事务（channel.txCommit）。但是问题是，rabbitmq事务机制一搞，基本上吞吐量会下来，因为太耗性能。

所以一般来说，如果你要确保说写rabbitmq的消息别丢，可以开启confirm模式，在生产者那里设置开启confirm模式之后，你每次写的消息都会分配一个唯一的id，然后如果写入了rabbitmq中，rabbitmq会给你回传一个ack消息，告诉你这个消息ok了。如果rabbitmq没能处理这个消息，会回调你一个nack接口，告诉你这个消息接收失败，你可以重试。而且你可以结合这个机制自己在内存里维护每个消息id的状态，如果超过一定时间还没接收到这个消息的回调，那么你可以重发。

事务机制和confirm机制最大的不同在于，事务机制是同步的，你提交一个事务之后会阻塞在那儿，但是confirm机制是异步的，你发送个消息之后就可以发送下一个消息，然后那个消息rabbitmq接收了之后会异步回调你一个接口通知你这个消息接收到了。

所以一般在生产者这块避免数据丢失，都是用confirm机制的。

2) rabbitmq弄丢了数据

就是rabbitmq自己弄丢了数据，这个你必须开启rabbitmq的持久化，就是消息写入之后会持久化到磁盘，哪怕是rabbitmq自己挂了，恢复之后会自动读取之前存储的数据，一般数据不会丢。除非极其罕见的是，rabbitmq还没持久化，自己就挂了，可能导致少量数据会丢失的，但是这个概率较小。

设置持久化有两个步骤，第一个是创建queue的时候将其设置为持久化的，这样就可以保证rabbitmq持久化queue的元数据，但是不会持久化queue里的数据；第二个是发送消息的时候将消息的deliveryMode设置为2，就是将消息设置为持久化的，此时rabbitmq就会将消息持久化到磁盘上去。必须要同时设置这两个持久化才行，rabbitmq哪怕是挂了，再次重启，也会从磁盘上重启恢复queue，恢复这个queue里的数据。

而且持久化可以跟生产者那边的confirm机制配合起来，只有消息被持久化到磁盘之后，才会通知生产者ack了，所以哪怕是在持久化到磁盘之前，rabbitmq挂了，数据丢了，生产者收不到ack，你也是可以自己重发的。

3) 消费端弄丢了数据

rabbitmq如果丢失了数据，主要是因为消费的时候，刚消费到，还没处理，结果进程挂了，比如重启了，那么就尴尬了，rabbitmq认为你都消费了，这数据就丢了。

这个时候得用rabbitmq提供的ack机制，简单来说，就是你关闭rabbitmq自动ack，可以通过一个api来调用就行，然后每次你自己代码里确保处理完的时候，再程序里ack一把。这样的话，如果你还没处理完，不就没有ack？那rabbitmq就认为你还没处理完，这个时候rabbitmq会把这个消费分配给别的consumer去处理，消息是不会丢的。

9.6如何保证消息的顺序性

错乱场景：一个queue，多个consumer，这明显乱了

rabbitmq：拆分多个queue，每个queue一个consumer，就是多一些queue而已，确实是麻烦点；或者就一个queue但是对应一个consumer，然后这个consumer内部用内存队列做排队，然后分发给底层不同的worker来处理。