课程目标

1、RabbitMQ 介绍；

2、RabbitMQ 架构原理介绍；

3、RabbitMQ 消息分发机制介绍；

4、RabbitMQ 持久化与内存管理；

5、RabbitMQ 插件管理

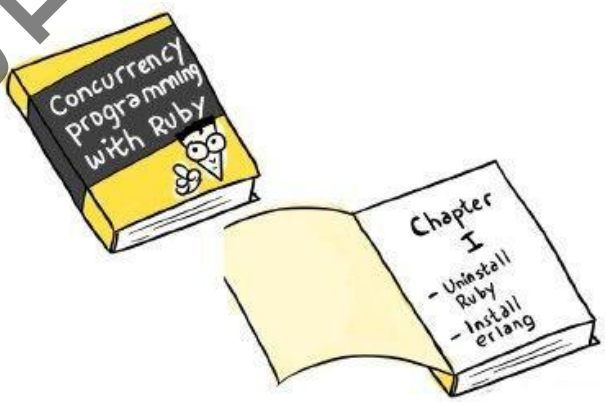
6、RabbitMQ 的 Java API 编程

7、Spring 集成 RabbitMQ 与 SpringBoot API

1、RabbitMQ 简介

官网 https://www.rabbitmq.com/getstarted.html

2007 年，Rabbit 技术公司基于 AMQP 开发了 RabbitMQ 1.0。为什么要用 Erlang 语言呢？因为 Erlang 是作者 Matthias 擅长的开发语言。第二个就是 Erlang 是为电话交换机编写的语言，天生适合分布式和高并发。



为什么要取 Rabbit Technologies 这个名字呢？因为兔子跑得很快，而且繁殖起来很疯狂。

从最开始用在金融行业里面，现在 RabbitMQ 已经在世界各地的公司中遍地开花。

国内的绝大部分大厂都在用 RabbitMQ，包括头条，美团，滴滴（TMD），去哪儿，艺龙，淘宝也有用。

RabbitMQ 和 Spring 家族属于同一家公司：Pivotal。

当然，除了 AMQP 之外，RabbitMQ 支持多种协议，STOMP、MQTT、HTTP、WebSockets。

2、环境安装

具体安装过程在预习资料中已经介绍了，重点说明几个安装中遇到的问题：

2.1 yum 方式安装 erlang

官方文档地址：https://www.erlang-solutions.com/resources/download.html

1.1.1.1、添加存储库条目

在 目录 /etc/yum.repos.d/ 下存储了 yum 常用的源，这里我们自己建立一个，使用 vi 指令来创建一个 rabbitmq_erlang.repo 文件。

```
vi /etc/yum.repos.d/rabbitmq_erlang.repo
```

在这个文件中，写入下列内容。

```
# 写入以下，保存退出
[rabbitmq_erlang]
name=rabbitmq_erlang
baseurl=https://packagecloud.io/rabbitmq/erlang/el/7/$basearch
repo_gpgcheck=1
gpgcheck=0
enabled=1
gpgkey=https://packagecloud.io/rabbitmq/erlang/gpgkey
sslverify=1
```

```
sslcacert=/etc/pki/tls/certs/ca-bundle.crt

metadata_expire=300

[rabbitmq_erlang-source]

name=rabbitmq_erlang-source

baseurl=https://packagecloud.io/rabbitmq/erlang/el/7/SRPMS

repo_gpgcheck=1

gpgcheck=0

enabled=1

gpgkey=https://packagecloud.io/rabbitmq/erlang/gpgkey

sslverify=1

sslcacert=/etc/pki/tls/certs/ca-bundle.crt

metadata_expire=300
```

1.1.1.2、安装

```
  yum install -y erlang
```

1.1.1.3、验证是否安装成功

```
  erl -version
```

2.2 安装报错

warning: rabbitmq-server-3.8.11-1.el8.noarch.rpm: Header V4 RSA/SHA256 Signature, key ID 6026dfca: NOKEY
error: Failed dependencies:
socat is needed by rabbitmq-server-3.8.11-1.el8.noarch

解决方法：执行安装 socat 命令安装即可：

```
yum install socat
```

2.3 设置开机启动

```
sudo systemctl enable rabbitmq-server
```

2.4 防火墙添加端口

RabbitMQ 服务启动后，还不能进行外部通信，需要将端口添加都防火墙

1. 添加端口

```
sudo firewall-cmd --zone=public --add-port=4369/tcp --permanent
sudo firewall-cmd --zone=public --add-port=5672/tcp --permanent
sudo firewall-cmd --zone=public --add-port=25672/tcp --permanent
sudo firewall-cmd --zone=public --add-port=15672/tcp --permanent
```

2. 重启防火墙

```
sudo firewall-cmd --reload
```

如果是阿里云服务器记得在安全策略组添加开放以上端口，如果是本地虚拟机，可以临时关闭防火墙。

```
sudo systemctl stop firewalld
```

也可以直接永久关闭防火墙，当然，生产环境不建议关闭防火墙。

停止 firewall

```
systemctl stop firewalld.service
```

禁止 firewall 开机启动

```
systemctl disable firewalld.service
```

2.6、UI 管理界面的使用

RabbitMQ 可以通过命令（RabbitMQ CLI）、HTTP API 管理，也可以通过可视化的界面去管理，这个网页就是 managment 插件。

2.6.1 启用管理插件

Windows 启用管理插件

```
cd C:\Program Files\RabbitMQ Server\rabbitmq_server-3.6.6\sbin

rabbitmq-plugins.bat enable rabbitmq_management
```

Linux 启用管理插件

```
cd /usr/lib/rabbitmq/bin

./rabbitmq-plugins enable rabbitmq_management
```

2.6.2 管理界面访问端口

默认端口是 15672，默认用户 guest，密码 guest。

guest 用户默认只能在本机访问，远程用户需要创建其他的用户。

2.6.3 虚拟主机

在 Admin 选项卡中：

默认的虚拟机是 /，可以创建自定义的虚拟机。

2.6.7 创建用户，权限

例如创建用户 admin，密码 admin，授权访问所有的 Vhost

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

firewall-cmd --reload
```

```
rabbitmqctl add_user admin admin

rabbitmqctl set_user_tags admin administrator

rabbitmqctl set_permissions -p / admin ".*" ".*" ".*"
```

以上是 RabbitMQ 的基础使用，如果仅仅是发送普通的消息，掌握以上的知识已经够用了。不过，对于一些特殊的业务场景，我们用到特殊的消息类型或者管理方式，下面我们来学习一下 RabbitMQ 的一些高级功能。

2.7、彻底卸载 RabbitMQ

最简单的方式是用 yum 安装

1、停止 rabbitmq 服务

```
systemctl stop rabbitmq-server
```

2、卸载 erlang

查看 erlang 安装的相关列表

```
yum list|grep erlang
```

卸载 erlang 所有内容

```
yum -y remove erlang-*
```

删除 erlang 目录

```
rm -rf /usr/lib64/erlang
```

3、卸载 rabbitmq

查看 rabbitmq 安装的相关列表

```
yum list|grep rabbitmq
```

卸载 rabbitmq 所有内容

```
yum -y remove rabbitmq-server.noarch
```

查找并删除 rabbitmq 相关目录

```
find / -name rabbit*
```

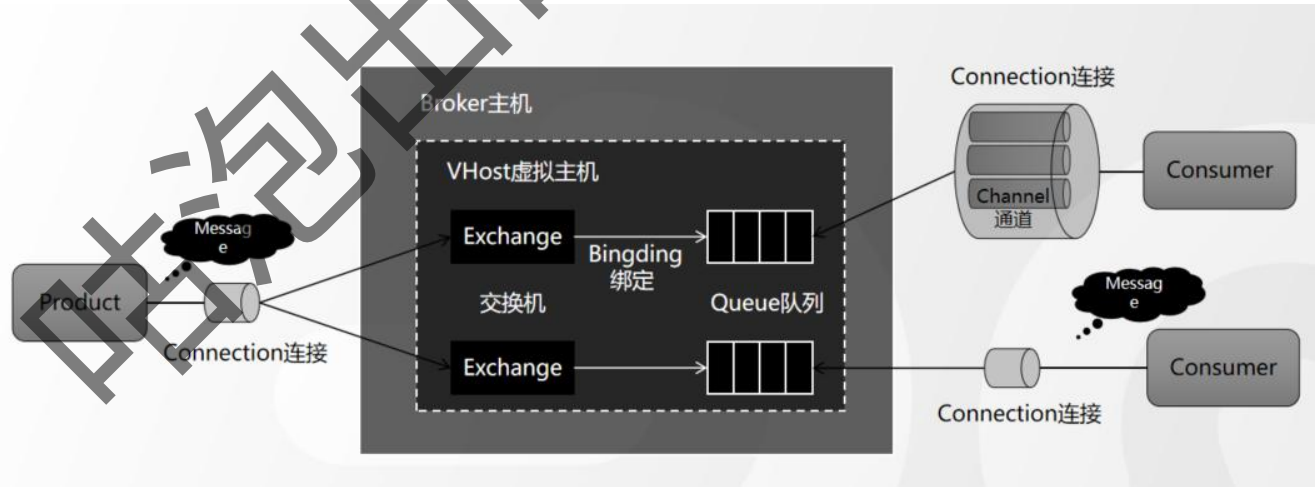
依次删除对应目录：rm -rf 路径

例如：

```
rm -rf /var/lib/rabbitmq

rm -rf /usr/lib/rabbitmq
```

3、RabbitMQ 架构原理

由于 RabbitMQ 实现了 AMQP 协议，所以 RabbitMQ 的工作模型也是基于 AMQP 的。理解这张图片至关重要。



3.1 Broker 中介

我们要使用 RabbitMQ 来收发消息，必须要安装一个 RabbitMQ 的服务，可以安

装在 Windows 上面也可以安装在 Linux 上面，默认是 5672 的端口。这台 RabbitMQ 的服务器我们把它叫做 Broker，中文翻译是代理/中介，因为 MQ 服务器帮助我们做的事情就是存储、转发消息。

3.2 Connection 连接

无论是生产者发送消息，还是消费者接收消息，都必须要跟 Broker 之间建立一个连接，这个连接是一个 TCP 的长连接。

3.3 Channel 通道

https://www.rabbitmq.com/api-guide.html#concurrency

https://www.rabbitmq.com/channels.html

https://stackoverflow.com/questions/18418936/rabbitmq-and-relationship-between-channel-and-connection

如果所有的生产者发送消息和消费者接收消息，都直接创建和释放 TCP 长连接的话，对于 Broker 来说肯定会造成很大的性能损耗，也会浪费时间。

所以在 AMQP 里面引入了 Channel 的概念，它是一个虚拟的连接。我们把它翻译成通道，或者消息信道。这样我们就可以在保持的 TCP 长连接里面去创建和释放 Channel，大大了减少了资源消耗。

不同的 Channel 是相互隔离的，每个 Channel 都有自己的编号。对于每个客户端线程来说，Channel 就没必要共享了，各自用自己的 Channel。

另外一个需要注意的是，Channel 是 RabbitMQ 原生 API 里面的最重要的编程接口，也就是说我们定义交换机、队列、绑定关系，发送消息，消费消息，调用的都是 Channel 接口上的方法。

3.4 Queue 队列

连接到 Broker 以后，就可以收发消息了。

在 Broker 上有一个对象用来存储消息,在 RabbitMQ 里面这个对象叫做 Queue。实际上 RabbitMQ 是用数据库来存储消息的，这个数据库跟 RabbitMQ 一样是用 Erlang 开发的，名字叫 Mnesia。我们可以在磁盘上找到 Mnesia 的存储路径。

Windows 系统保存在用户目录下:

```
C:\Users\用户名\AppData\Roaming\RabbitMQ\db\rabbit@用户名-mnesia
```

CentOS 保存在/var/lib 目录下:

```
/var/lib/rabbitmq/mnesia
```

队列也是生产者和消费者的纽带，生产者发送的消息到达队列，在队列中存储。消费者从队列消费消息。

字段	说明
type	此 queue 的类型，默认为 classic 主队列，也可以设置为 quorum 从队列
name	此 queue 的名称
durability	queue 中的消息是否要持久化到硬盘
auto delete	如果此 queue 没有绑定到任何一个 exchange，是否自动删除此 queue
arguments	设置一些其它参数 x-expires：队列在多长时间没有被使用后删除 x-max-length：最大消息条数。先进先出原则，超过后，后面的消息会顶替前面的消息 x-max-length-bytes：最大容量 x-dead-letter-exchange：延迟结束后指向交换机（死信收容交换机） x-dead-letter-routing-key：延迟结束后指向队列（死信收容队列），也可直接设置 queue name

3.5 Consumer 消费者

消息到底是 Broker 推送给消费者的？还是消费者主动获取的？消费者消费消息有两种模式。

一种是 Pull 模式，对应的方法是 basicGet。消息存放在服务端，只有消费者主动获取才能拿到消息。如果每隔一段时间获取一次消息，消息的实时性会降低。但是好处是可以根据自己的消费能力决定获取消息的频率。

另一种是 Push 模式，对应的方法是 basicConsume，只要生产者发消息到服务器，就马上推送给消费者，消息保存在客户端，实时性很高，如果消费不过来有可能会造成消息积压。Spring AMQP 是 push 方式，通过事件机制对队列进行监听，只要有消息到达队列，就会触发消费消息的方法。

RabbitMQ 中 pull 和 push 都有实现。而 kafka 和 RocketMQ 只有 pull。

https://www.rabbitmq.com/api-guide.html#consuming

由于队列有 FIFO 的特性，只有确定前一条消息被消费者接收之后，Broker 才会把这条消息从数据库删除，继续投递下一条消息。

一个消费者是可以监听多个队列的，一个队列也可以被多个消费者监听。

但是在生产环境中，我们一般是建议一个消费者只处理一个队列的消息。如果需要提升处理消息的能力，可以增加多个消费者。这个时候消息会在多个消费者之间轮询。

3.6 Exchange 交换机

现在我们来思考一个问题，如果要把一条消息发送给多个队列，被多个消费者消费，应该怎么做？生产者是不是必须要调用多次 basicPublish 的方法，依次发送给多个队列？就像消息推送的这种场景，有成千上万个队列的时候，对生产者来说压力太大了。

有没有更好的办法呢？其实，RabbitMQ 已经为我们考虑到了这一点，它设计了一个帮我们路由消息的组件，叫做 Exchange。

也就是说，不管有多少个队列需要接收消息，我都只需要发送到 Exchange 就 OK 了，由它帮我来分发。Exchange 是不会存储消息的，它只做一件事情，根据规则分发消息。

那么，Exchange 和这些需要接收消息的队列必须建立一个绑定关系，并且为每个

队列指定一个特殊的标识。

Exchange 和队列是多对多的绑定关系，也就说，一个交换机的消息一个路由给多个队列，一个队列也可以接收来自多个交换机的消息。

绑定关系建立好之后，生产者发送消息到 Exchange，也会携带一个特殊的标识。当这个标识跟绑定的标识匹配的时候，消息就会发给一个或者多个符合规则的队列。

那 Exchange 有哪些可设置的字段呢?

字段	说明
Name	交换机的名称
Type	**Direct**：把消息投递到那些 binding key 与 routing key 完全匹配的队列中 **fanout**:把所有发送到该 Exchange 的消息投递到所有与它绑定的队列中 **topic**:将消息路由到 binding key 与 routing key 模式匹配的队列中。 headers：通过匹配 AMQP 消息的 header 来路由 x-delayed-message:安装延时插件后，可以设置延时发送消息。
Durable	是否持久化（重启 rabbitmq 之后，交换机是否还存在）
AutoDelete	是否自动删除（当交换机把消息发送到与之绑定的消息队列后，是否删除交换机）
Internal	设置为“Yes”之后，表示当前 Exchange 是 RabbitMQ 内部使用，Queue 不会消费该类型交换机下的消息
Arguments	**alternate-exchange**：声明一个备用交换机，一般声明为 fanout 类型，这样交换机收到路由不到队列的消息就会发送到备用交换机绑定的队列中

这个特殊的标识是怎么运作的，后面再详细展开来讲。

3.7 Vhost 虚拟机

我们每个需要实现基于 RabbitMQ 的异步通信的系统，都需要在 Broker 上创建自己要用到的交换机、队列和它们的绑定关系。如果某个业务系统不想跟别人混用一个 Broker,怎么办？再采购一台硬件服务器单独安装一个 RabbitMQ 服务？这种方式成本太高了。在同一个硬件服务器上安装多个 RabbitMQ 的服务呢？比如再运行一个 5673 的端口?

没有必要这样做，因为 RabbitMQ 也考虑到了这一点，设计了虚拟主机 VHOST。

VHOST 除了可以提高硬件资源的利用率之外，还可以实现资源的隔离和权限的控制。它的作用类似于其他编程语言中的 namespace 和 package，不同的 VHOST 中

可以有同名的 Exchange 和 Queue，它们是完全透明的。

这个时候，我们可以为不同的业务系统创建专属于他们自己的 VHOST，然后再为他们创建专属的用户，给用户分配对应的 VHOST 的权限。比如给风控系统的用户分配风控系统的 VHOST 的权限，这个用户可以访问里面的交换机和队列。给超级管理员分配所有 VHOST 的权限。

我们安装 RabbitMQ 的时候会自带一个默认的 VHOST，名字是“/”。

4、消息分发机制

我们说到 RabbitMQ 引入 Exchange 是为了实现消息的灵活路由，到底有哪些路由方式?

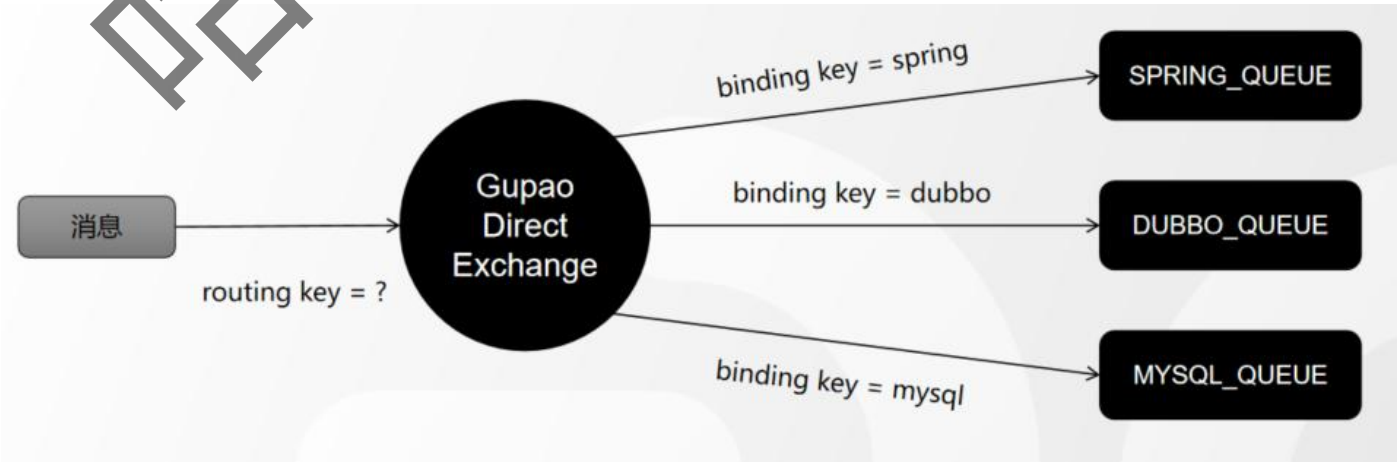
RabbitMQ 中一共有四种类型的交换机，Direct、Topic、Fanout、Headers。其中 Headers 不常用。交换机的类型可以在创建的时候指定，网页或者代码中。

4.1 Direct 直连

一个队列与直连类型的交换机绑定，需指定一个明确的绑定键（binding key）。

生产者发送消息时会携带一个路由键（routing key）。

当消息的路由键与某个队列的绑定键完全匹配时，这条消息才会从交换机路由到这个队列上。多个队列也可以使用相同的绑定键。



例如：

```
channel.basicPublish(“MY_DIRECT_EXCHANGE”,”spring”,”msg 1”);
```

只有第一个队列能收到消息。

直连类型的交换机，适用于一些业务用途明确的消息。比如 HR 系统跟销售系统之间通信，传输的是销售系统专用的消息，就可以建一个直连类型的交换机，使用明确的绑定键。

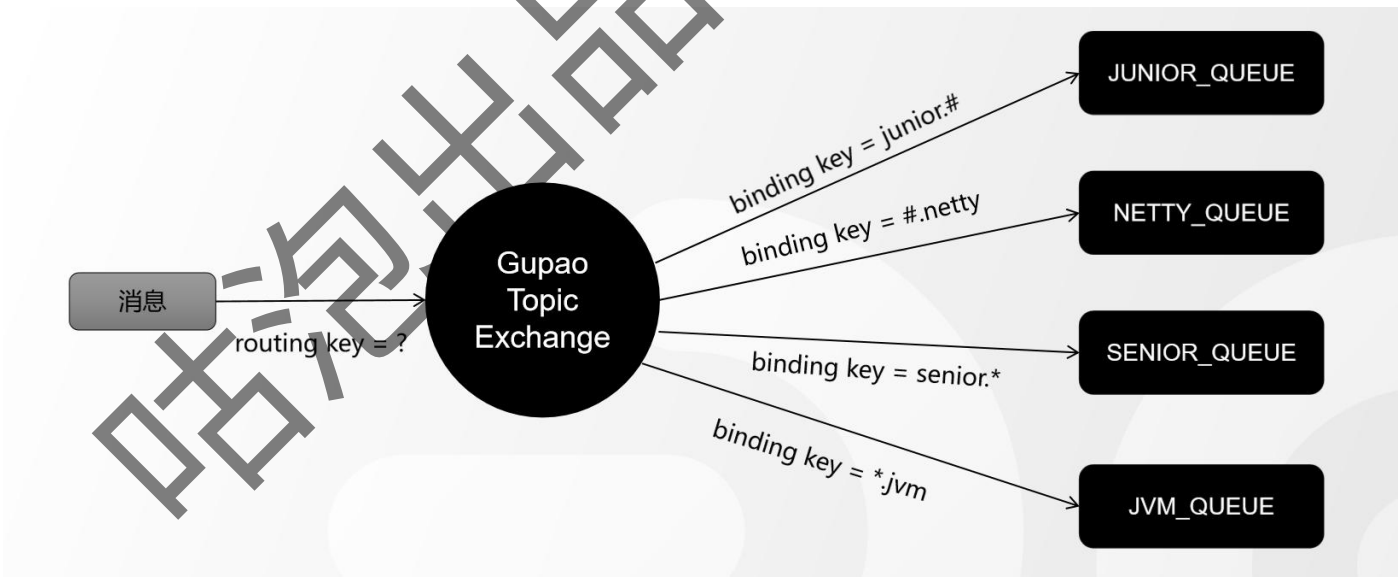
4.2 Topic 主题

一个队列与主题类型的交换机绑定时，可以在绑定键中使用通配符。支持两个通配符：

代表匹配 0 个或者多个单词

* 代表匹配不多不少一个单词

单词（word）指的是用英文的点“.”隔开的字符。例如 a.bc.def 是 3 个单词。



以上图为例，有 4 个队列绑定到我创建的主题类型的交换机，使用不同的绑定键。

解读：第一个队列支持路由键以 junior 开头的消息路由，后面可以有单词，也可以没有。

第二个队列支持路由键以 netty 结尾，前面可以有也可以没有单词的消息路由。

四单个队列支持路由键以 senior 开头，并且后面是一个单词的消息路由。

第四个队列支持路由键以 jvm 结尾，并且前面是一个单词的消息路由。

现在我们尝试发送消息，看看那些队列能收到：

```
channel.basicPublish("MY_TOPIC_EXCHANGE","junior.abc.jvm","msg 2");
```

只有第一个队列能收到消息。

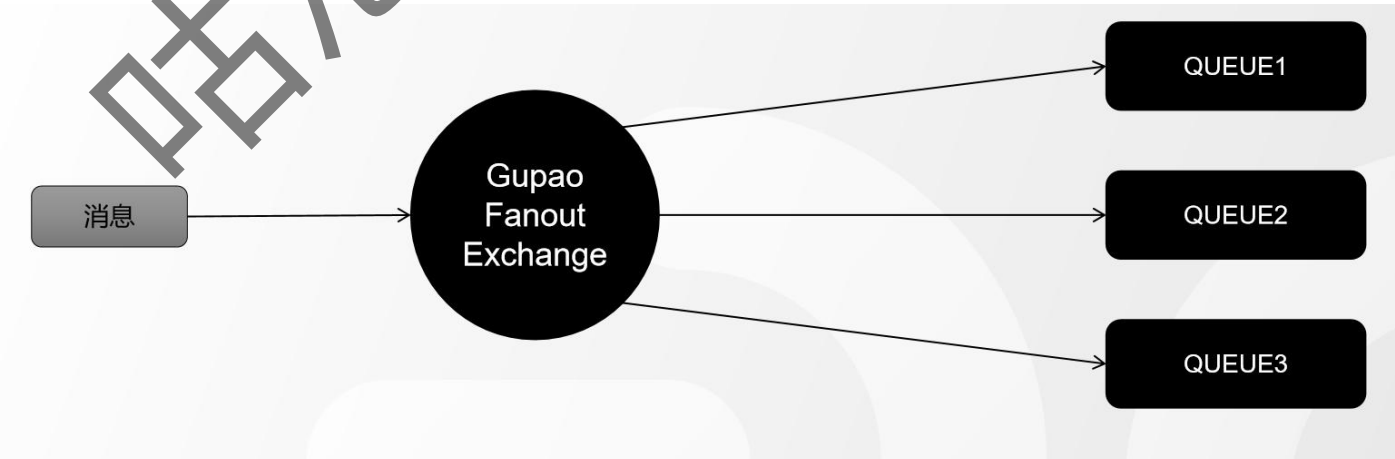
```
channel.basicPublish("MY_TOPIC_EXCHANGE","senior.netty", "msg 3");
```

第二个队列和第三个队列能收到消息。

主题类型的交换机适用于一些根据业务主题或者消息等级过滤消息的场景，比如说一条消息可能既跟资金有关，又跟风控有关，那就可以让这个消息指定一个多级的路由键。第一个单词代表跟资金相关，第二个单词代表跟风控相关。下游的业务系统的队列就可以使用不同的绑定键去接收消息了。

4.3 Fanout 广播

广播类型的交换机与队列绑定时，不需要指定绑定键。因此生产者发送消息到广播类型的交换机上，也不需要携带路由键。消息达到交换机时，所有与之绑定了的队列，都会收到相同的消息的副本。



例如：

```
channel.basicPublish("MY_FANOUT_EXCHANGE", "", "msg 4");
```

三个队列都会收到 msg 4。

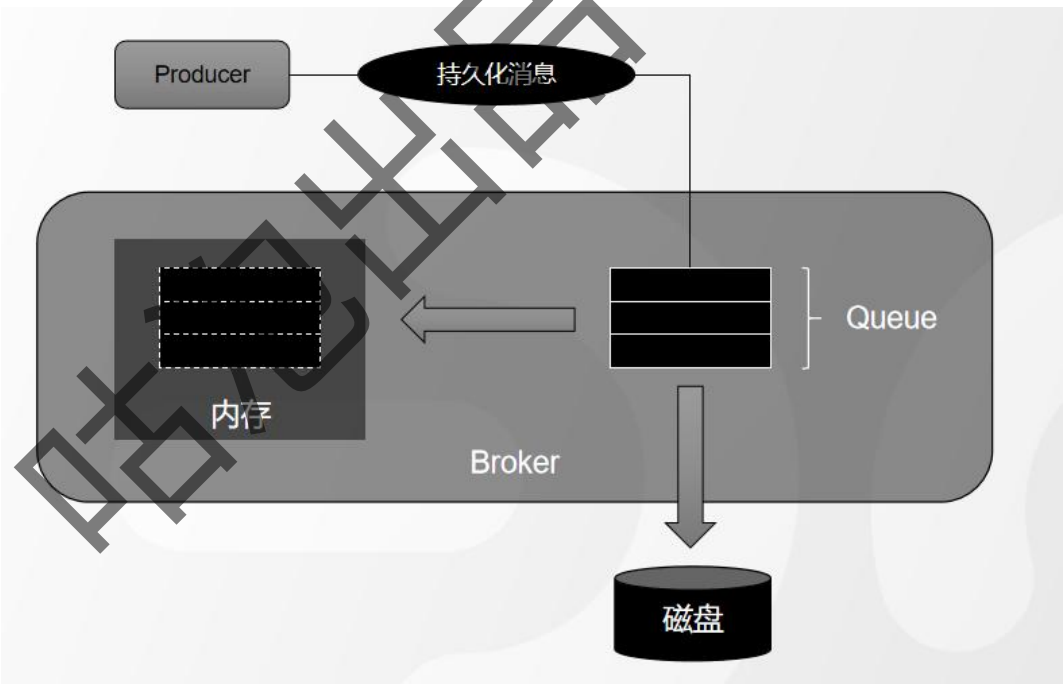
广播类型的消息适用于一些通用的业务消息。比如产品系统产品数据变化的消息，是所有的系统都会用到的，就可以创建一个广播类型的交换机，大家自己建队列就绑定就 OK 了。

现在我们已经明白了 RabbitMQ 的概念模型，下面来看下到底怎么用。首先我们需要有一个 RabbitMQ 的服务。

5、RabbitMQ 持久化与内存管理

5.1 持久化机制

RabbitMQ 的持久化分为消息持久化、队列持久化、交换器持久化。无论是持久化消息还是非持久化消息都可以被写入磁盘。



当 RabbitMQ 收到消息时，如果是持久化消息，则会储存在内存中，同时也会写入磁盘；如果是非持久化消息，则只会存在内存中。当内存使用达到 RabbitMQ 的临界值时，内存中的数据会被交换到磁盘，持久化消息由于本就存在于磁盘中，不会被

重复写入。

消息的持久化是在发消息时，通过 deliveryMode 设置，队列、交换器也可以通过参数持久化，非持久化的消息、队列、交换器在服务重启后会消失，即使已经被写入磁盘。

5.2 内存控制

RabbitMQ 中通过内存阈值参数控制内存的使用量，当内存使用超过配置的阈值时，RabbitMQ 会阻塞客户端的连接并停止接收从客户端发来的消息，以免服务崩溃，同时，会发出内存告警，此时客户端于与服务端的心跳检测也会失效。

当出现内存告警时，可以通过管理命令临时调整。

```
rabbitmqctl set_vm_memory_high_watermark <fraction>
```

fraction 为内存阈值，默认是 0.4，表示 RabbitMQ 使用的内存超过系统内存的 40%时，会产生内存告警，通过此命令修改的阈值在重启后会失效。可以通过修改配置文件的方式，使之永久生效，但是需要重启服务。

```
# rabbitmq.conf

vm_memory_high_watermark.relative=0.4

#vm_memory_high_watermark.absolute=1GB
```

RabbitMQ 提供 relative 与 absolute 两种配置方式

relative：相对值，也就是前面的 fraction 参数，建议 0.4～0.66，不能太大

absolute：绝对值，固定大小，单位为 KB、MB、GB

```
rabbitmqctl set_vm_memory_high_watermark absolute <value>
```

5.4 内存换页

在 RabbitMQ 达到内存阈值并阻塞生产者之前，会尝试将内存中的消息换页到磁盘，以释放内存空间。内存换页由换页参数控制，默认为 0.5，表示当内存使用量达到内存阈值的 50%时会进行换页，也就是 0.4*0.5=0.2。

```
vm_memory_high_watermark_paging_ratio=0.5
```

当换页阈值大于 1 时，相当于禁用了换页功能

5.5 磁盘控制

RabbitMQ 通过磁盘阈值参数控制磁盘的使用量，当磁盘剩余空间小于磁盘阈值时，RabbitMQ 同样会阻塞生产者，避免磁盘空间耗尽。

磁盘阈值默认 50M，由于是定时检测磁盘空间，不能完全消除因磁盘耗尽而导致崩溃的可能性，比如在两次检测之间，磁盘空间从大于 50M 变为 0M。

一种相对谨慎的做法是将磁盘阈值大小设置与内存相等

```
rabbitmqctl set_disk_free_limit <limit>

rabbitmqctl set_disk_free_limit mem_relative <fraction>

# limit 为绝对值，KB、MB、GB

# fraction 为相对值，建议 1.0～2.0 之间

# rabbitmq.conf

disk_free_limit.relative=1.5

# disk_free_limit.absolute=50MB
```

6、RabbiMQ 插件管理

RabbitMQ 插件机制的设计，主要是用于面对特殊场景的需求，可以实现任意扩展。

6.1 插件列表管理命令

```
rabbitmq-plugins list
```

此命令，列出当前可以安装使用的插件。插件前面[ ] 为空说明，没有安装。有 e* 说明插件是安装了的。

6.2 插件安装使用，

我们以 rabbitmq_management 为例。

```
rabbitmq-plugins enable rabbitmq_management
```

此命令，就是安装启用管理。

6.3 插件卸载

同上，插件卸载，只需要将我们的命令，enable 改为，disbale 即可。

```
rabbitmq-plugins disable rabbitmq_management
```

7、RabbitMQ 配置备查

RabbitMQ 常用的三种自定义服务器的通用方法：

配置文件 rabbitmq.conf

环境变量文件 rabbitmq-env.conf

补充配置文件 advanced.config

rabbitmq.conf 和 rabbitmq-env.conf 的位置

在二进制安装中路径是在 ： 安装目录下的/etc/rabbitmq/

rpm 安装： /etc/rabbitmq/

如果 rabbitmq.conf 和 rabbitmq-env.conf 的两个文件不存在，那么我们可以创建该文件，然后我们可以通过环境变量
指定该文件的位置。

补充 ：

rabbitmqctl rabbitmqctl 是管理虚拟主机和用户权限的工具

rabbitmq-plugins 是管理插件的工具

7.1 rabbitmq.conf

在 rabbitmq 3.7.0 之前，rabbitmq.conf 使用了 Erlang 语法配置格式，新的版本使用了 sysctl 格式.

sysctl 语法：

单个信息都在一行里面

配置信息以 key value 的形式保存。

‘#’ 开头表示注释。

配置属性和描述（官网链接：https://www.rabbitmq.com/configure.html）

属性	描述	默认值
listeners	要监听 AMQP 0-9-1 and AMQP 1.0 的端口	listeners.tcp.default = 5672
num_acceptors.tcp	接受 tcp 连接的 erlang 进程数	num_acceptors.tcp = 10
handshake_timeout	AMQP 0-9-1 超时时间，也就是最大的连接时间，单位毫秒	handshake_timeout = 10000

属性	描述	默认值
listeners.ssl	启用 TLS 的协议	默认值为 none
num_acceptors.ssl	接受基于 TLS 协议的连接的 erlang 进程数	num_acceptors.ssl = 10
ssl_options	TLS 配置	ssl_options =none
ssl_handshake_timeout	TLS 连接超时时间 单位为毫秒	ssl_handshake_timeout = 5000
vm_memory_high_watermark	触发流量控制的内存阈值，可以为相对值(0.5),或者绝对值 vm_memory_high_watermark.relative = 0.6 ,vm_memory_high_watermark.absolute = 2GB	默认 vm_memory_high_watermark.relative = 0.4
vm_memory_calculation_strategy	内存使用报告策略，assigned：使用 Erlang 内存分配器统计信息 rss：使用操作系统 RSS 内存报告。这使用特定于操作系统的方法，并可能启动短期子进程。legacy：使用遗留内存报告（运行时认为将使用多少内存）。这种策略相当不准确。erlang 与 legacy 一样 是为了向后兼容	vm_memory_calculation_strategy = allocated
vm_memory_high_watermark_paging_ratio	当内存的使用达到了 50%后,队列开始将消息分页到磁盘	vm_memory_high_watermark_paging_ratio = 0.5
total_memory_available_override_value	该参数用于指定系统的可用内存总量，一般不使用，适用于在容器等一些获取内存实际值不精确的环境	默认未设置
disk_free_limit	Rabbitmq 存储数据的可用空间限制，当低于该值的时候，将触发流量限制，设置可参考 vm_memory_high_watermark 参数	disk_free_limit.absolute = 50MB
log.file.level	控制记录日志的等级，有 info,error,warning,debug	log.file.level = info
channel_max	最大通道数，但不包含协议中使用的特殊通道号 0，设置为 0 表示无限制，不建议使用该值，容易出现 channel 泄漏	channel_max = 2047
channel_operation_timeout	通道操作超时，单位为毫秒	channel_operation_timeout = 15000
heartbeat	表示连接参数协商期间服务器建议的心跳超时的值。如果两端都设置为 0，则禁用心跳,不建议禁用	heartbeat = 60
default_vhost	rabbitmq 安装后启动创建的虚拟主机	default_vhost = /
default_user	默认创建的用户名	default_user = guest
default_pass	默认用户的密码	default_pass = guest
default_user_tags	默认用户的标签	default_user_tags.administrator = true
default_permissions	在创建默认用户是分配给默认用户的权限	default_permissions.configure = .* default_permissions.read = .* default_permissions.write = .*
loopback_users	允许通过回环地址连接到 rabbitmq 的用户列表,如果要允	loopback_users.guest = true(默认为

属性	描述	默认值
	许guest用户远程连接(不安全)请将该值设置为none,如果要将一个用户设置为仅 localhost 连接的话，配置 loopback_users.username =true(username 要替换成用户名)	只能本地连接)
cluster_formation.classic_config.nodes	设置集群节点 cluster_formation.classic_config.nodes.1 = rabbit@hostname1	
cluster_formation.classic_config.nodes.2 = rabbit@hostname2	默认为空，未设置	
collect_statistics	统计收集模式，none 不发出统计信息事件，coarse 每个队列连接都发送统计一次,fine 每发一条消息的统计数据	collect_statistics = none
collect_statistics_interval	统计信息收集间隔，以毫秒为单位	collect_statistics_interval = 5000
delegate_count	用于集群内通信的委托进程数。在多核的服务器上我们可以增加此值	delegate_count = 16
tcp_listen_options	默认的套接字选项	tcp_listen_options.backlog = 128 .....
hipe_compile	设置为 true 以使用 HiPE 预编译 RabbitMQ 的部分，HiPE 是 Erlang 的即时编译器，启用 HiPE 可以提高吞吐量两位数，但启动时会延迟几分钟。Erlang 运行时必须包含 HiPE 支持。如果不是，启用此选项将不起作用。HiPE 在某些平台上根本不可用，尤其是 Windows。	hipe_compile = false
cluster_keepalive_interval	节点应该多长时间向其他节点发送 keepalive 消息(以毫秒为单位),keepalive 的消息丢失不会被视为关闭	cluster_keepalive_interval = 10000
queue_index_embed_msgs_below	消息的字节大小,低于该大小，消息将直接嵌入队列索引中 bytes	queue_index_embed_msgs_below = 4096
mnesia_table_loading_retry_timeout	等待集群中 Mnesia 表可用的超时时间，单位毫秒	mnesia_table_loading_retry_timeout = 30000
mnesia_table_loading_retry_limit	集群启动时等待 Mnesia 表的重试次数，不适用于 Mnesia 升级或节点删除。	mnesia_table_loading_retry_limit = 10
mirroring_sync_batch_size	要在队列镜像之间同步的消息的批处理大小	mirroring_sync_batch_size = 4096
queue_master_locator	队列主节点的策略，有三大策略 min-masters, client-local, random	queue_master_locator = client-local
proxy_protocol	如果设置为 true ,则连接需要通过反向代理连接，不能直连接	proxy_protocol = false

属性	描述	默认值
management.listener.port	rabbitmq web 管理界面使用的端口	management.listener.port = 15672

查看 rabbitmq 的有效配置

```
rabbitmqctl environment
```

7.2 advanced.config

某些配置设置不可用或难以使用 sysctl 格式进行配置。因此，可以使用 Erlang 术语格式的其他配置文件 advanced.config

它将与 rabbitmq.conf 文件中提供的配置合并。

配置属性和描述

属性	描述	默认值
msg_store_index_module	设置队列索引使用的模块	{rabbit, [{msg_store_index_module, rabbit_msg_store_ets_index}]}
backing_queue_module	队列内容的实现模块。	{rabbit,[{backing_queue_module,rabbit_variable_queue}]}
msg_store_file_size_limit	消息储存的文件大小,现有的节点更改是危险的，可能导致数据丢失	默认值 16777216
trace_vhosts	内部的 tracer 使用，不建议更改	{rabbit, [{trace_vhosts, []}]}
msg_store_credit_disc_bound	设置消息储存库给队列进程的积分,默认一个队列进程被赋予 4000 个消息积分	{rabbit, [{msg_store_credit_disc_bound, {4000, 800}}]}
queue_index_max_journal_entries	队列的索引日志超过该阈值将刷新到磁盘	{rabbit, [{queue_index_max_journal_entries, 32768}]}
lazy_queue_explicit_gc_run_operation_threshold	在内存压力下为延迟队列设置的值，该值可以触发垃圾回收和减少内存使用,降低该值,会降低性能,提高该值，会导致更高的内存消耗	{rabbit,[{lazy_queue_explicit_gc_run_operation_threshold, 1000}]}
queue_explicit_gc_run_operation_threshold	在内存压力下，正常队列设置的值，该值可以触发垃圾回收和减少内存使用,降低该值,会降低性能,提高该值，会导致更高的内存消耗	{rabbit, [{queue_explicit_gc_run_operation_threshold, 1000}]}

7.3 rabbitmq-env.conf

通过 rabbitmq-env.conf 来定义环境变量
RABBITMQ_NODENAME 指定节点名称

属性	描述	默认值
RABBITMQ_NODE_IP_ADDRESS	绑定的网络接口	默认为空字符串表示绑定本机所有的网络接口
RABBITMQ_NODE_PORT	端口	默认为 5672
RABBITMQ_DISTRIBUTION_BUFFER_SIZE	节点之间通信连接的数据缓冲区大小	默认为 128000,该值建议不要使用低于 64MB
RABBITMQ_IO_THREAD_POOL_SIZE	运行时用于 io 的线程数	建议不要低于 32，linux 默认为 128 ，windows 默认为 64
RABBITMQ_NODENAME	rabbitmq 节点名称，集群中要注意节点名称唯一	linux 默认节点名为 rabbit@$hostname
RABBITMQ_CONFIG_FILE	rabbitmq 的配置文件路径，注意不要加文件的后缀(.conf)	默认 $RABBITMQ_HOME/etc/rabbitmq/rabbitmq(二进制安装) /etc/rabbitmq/rabbitmq(rpm 安装)
RABBITMQ_ADVANCED_CONFIG_FILE	advanced.config 文件路径	默认 $RABBITMQ_HOME/etc/rabbitmq/advanced(二进制安装) /etc/rabbitmq/advanced(rpm 安装)
RABBITMQ_CONF_ENV_FILE	环境变量配置文件路径	默认 $RABBITMQ_HOME/etc/rabbitmq/rabbitmq-env.conf(二进制安装) /etc/rabbitmq/rabbitmq-env.conf(rpm 安装)
RABBITMQ_SERVER_CODE_PATH	在使用 HiPE 模块时需要使用	默认为空
RABBITMQ_LOGS	指定日志文件位置	默认为 $RABBITMQ_HOME/etc/var/log/rabbitmq/

网络设置 http://www.rabbitmq.com/networking.html

RABBITMQ_DISTRIBUTION_BUFFER_SIZE 节点间通信缓冲区大小,默认值 128Mb,节点流量比较多的集群中，可以提升该值，建议该值不要低于 64MB。

tcp 缓存区大小

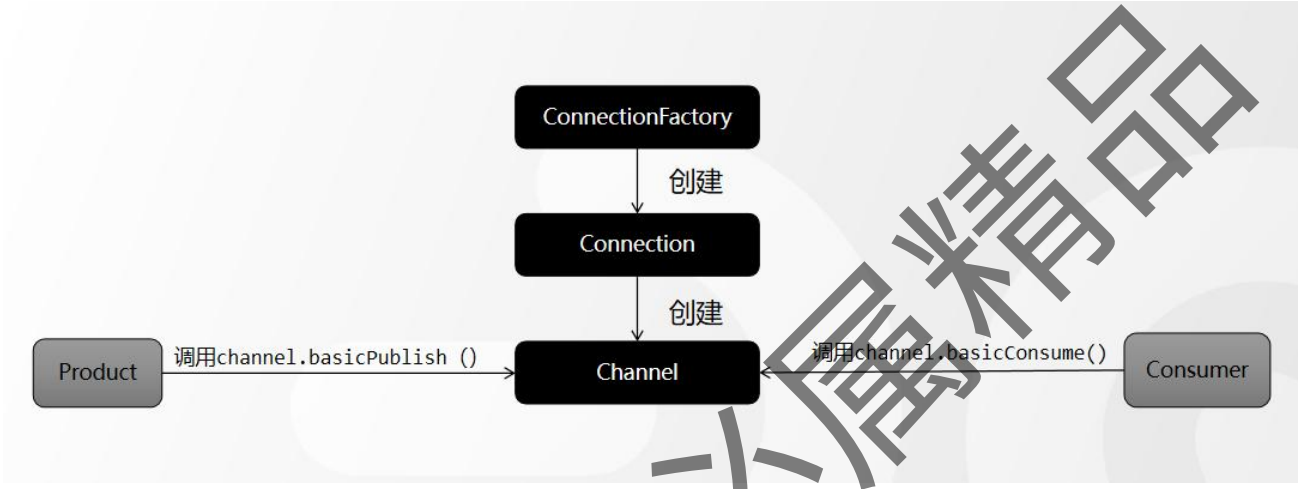
下示例将 AMQP 0-9-1 连接的 TCP 缓冲区设置为 192 KiB：

```
tcp_listen_options.backlog = 128tcp_listen_options.nodelay =
truetcp_listen_options.linger.on = truetcp_listen_options.linger.timeout =
0tcp_listen_options.sndbuf = 196608tcp_listen_options.recbuf = 196608
```

8、Java 基础 API 介绍

虽然大部分时候都是在 Spring 里面用 RabbitMQ，但我们还是从 Java API 开始，可以知道它的本质，而且在脱离 Spring 之后一样可以编程管理对象和消息。

https://www.rabbitmq.com/api-guide.html



8.1 引入依赖

创建 Maven 工程，pom.xml 引入依赖

```
<dependency>
   <groupId>com.rabbitmq</groupId>
   <artifactId>amqp-client</artifactId>
   <version>5.6.0</version>
</dependency>
```

8.2 消费者基础代码

```
package com.gupaoedu.vip.mq.rabbit.javaapi.simple;

import com.rabbitmq.client.*;

import java.io.IOException;

/**
 * 消息消费者
```

```
 */
public class MyConsumer {
    private final static String EXCHANGE_NAME = "SIMPLE_EXCHANGE";
    private final static String QUEUE_NAME = "SIMPLE_QUEUE";

    public static void main(String[] args) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        // 连接 IP
        factory.setHost("192.168.8.149");
        // 默认监听端口
        factory.setPort(5672);
        // 虚拟机
        factory.setVirtualHost("/");

        // 设置访问的用户
        factory.setUsername("admin");
        factory.setPassword("123456");
        // 建立连接
        Connection conn = factory.newConnection();
        // 创建消息通道
        Channel channel = conn.createChannel();

        // 声明交换机
        // String exchange, String type, boolean durable, boolean autoDelete, Map<String,
Object> arguments
        channel.exchangeDeclare(EXCHANGE_NAME,"direct",false, false, null);

        // 声明队列
        // String queue, boolean durable, boolean exclusive, boolean autoDelete,
Map<String, Object> arguments
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
        System.out.println(" Waiting for message....");

        // 绑定队列和交换机
        channel.queueBind(QUEUE_NAME,EXCHANGE_NAME,"gupao.best");

        // 创建消费者
        Consumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties,
                                       byte[] body) throws IOException {
                String msg = new String(body, "UTF-8");
                System.out.println("Received message : '" + msg + "'");
                System.out.println("consumerTag : " + consumerTag );
                System.out.println("deliveryTag : " + envelope.getDeliveryTag() );
            }
        };
```

```
        // 开始获取消息
        // String queue, boolean autoAck, Consumer callback
        channel.basicConsume(QUEUE_NAME, true, consumer);
    }
}
```

8.3 生产者基础代码

```
package com.gupaoedu.vip.mq.rabbit.javaapi.simple;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

/**
 * 消息生产者
 */
public class MyProducer {
    private final static String EXCHANGE_NAME = "SIMPLE_EXCHANGE";

    public static void main(String[] args) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        // 连接 IP
        factory.setHost("192.168.8.149");
        // 连接端口
        factory.setPort(5672);
        // 虚拟机
        factory.setVirtualHost("/");
        // 用户
        factory.setUsername("admin");
        factory.setPassword("123456");

        // 建立连接
        Connection conn = factory.newConnection();
        // 创建消息通道
        Channel channel = conn.createChannel();

        // 发送消息
        String msg = "Hello world, Rabbit MQ";

        // String exchange, String routingKey, BasicProperties props, byte[] body
        channel.basicPublish(EXCHANGE_NAME, "gupao.best", null, msg.getBytes());

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

在 ConnectionFactory 中可以查看所有默认属性。

连接自动恢复 automaticRecovery

nio = false; 默认是 Blocking IO

connectionFactory.setNioParams(new NioParams().setNbIoThreads(4));

9、Spring AMQP API

https://www.docs4dev.com/docs/zh/spring-amqp/2.1.2.RELEASE/reference/_reference.html

9.1 Spring AMQP 介绍

思考:

Java API 方式编程，有什么问题?

Spring 封装 RabbitMQ 的时候，它做了什么事情?

1、管理对象（队列、交换机、绑定）

2、封装方法（发送消息、接收消息）

Spring AMQP 是对 Spring 基于 AMQP 的消息收发解决方案，它是一个抽象层，不依赖于特定的 AMQP Broker 实现和客户端的抽象，所以可以很方便地替换。比如我们可以使用 spring-rabbit 来实现。

```
<dependency>
   <groupId>org.springframework.amqp</groupId>
   <artifactId>spring-rabbit</artifactId>
   <version>1.3.5.RELEASE</version>
</dependency>
```

包括 3 个 jar 包:

Amqp-client-3.3.4.jar —— java api 的包

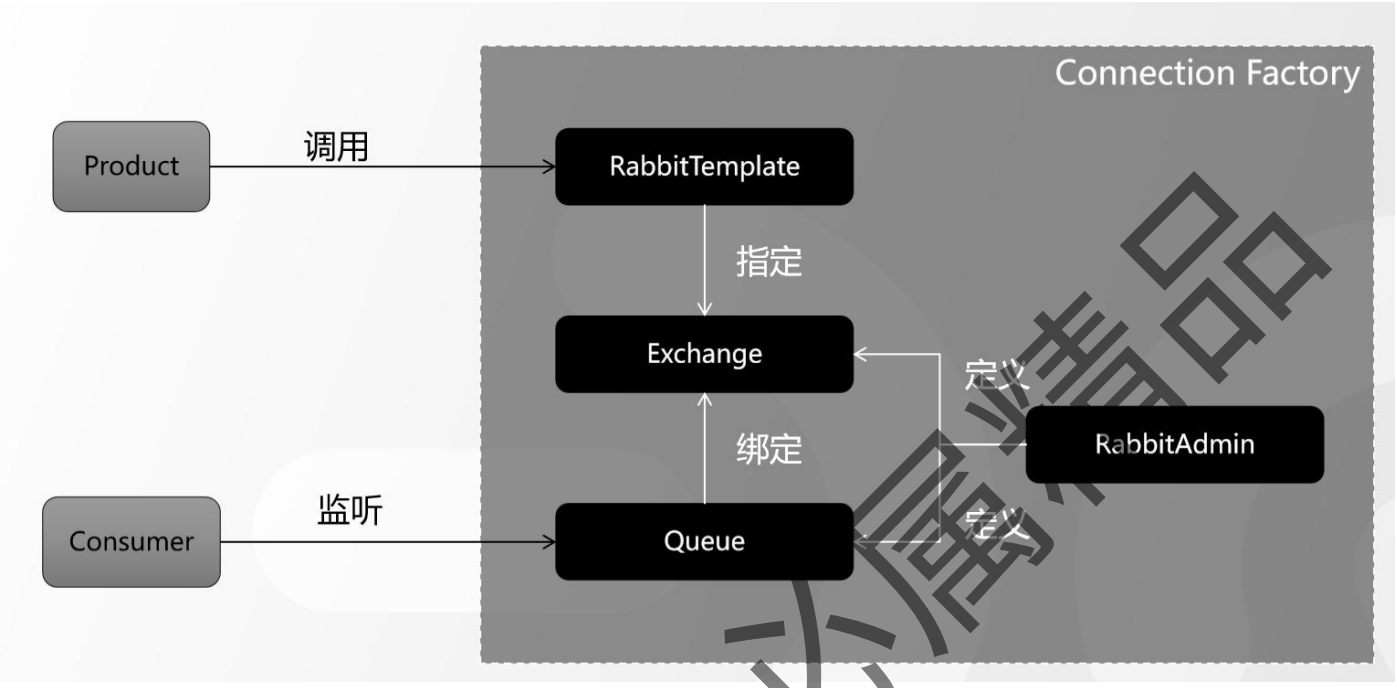
Spring-amqp.jar —— 对 AMQP 的封装

Spring.rabbit.jar —— rabbitq 对 AMQP 在 Spring 中的实现

9.2 Spring 集成 RabbitMQ 配置解读

```
  <rabbit:connection-factory id="connectionFactory" virtual-host="/" username="guest"
password="guest" host="127.0.0.1" port="5672" />


   <rabbit:admin id="connectAdmin" connection-factory="connectionFactory" />


   <rabbit:queue name="MY_FIRST_QUEUE" durable="true" auto-delete="false"
exclusive="false" declared-by="connectAdmin" />


   <rabbit:direct-exchange name="MY_DIRECT_EXCHANGE" durable="true" auto-delete="false"
declared-by="connectAdmin">
       <rabbit:bindings>
           <rabbit:binding queue="MY_FIRST_QUEUE" key="FirstKey">
           </rabbit:binding>
       </rabbit:bindings>
   </rabbit:direct-exchange>


<bean id="jsonMessageConverter"
  class="org.springframework.amqp.support.converter.Jackson2JsonMessageConverter" />
  <rabbit:template  id="amqpTemplate" exchange="${gupao.exchange}"
connection-factory="connectionFactory" message-converter="jsonMessageConverter" />


   <bean id="messageReceiver" class="com.gupaoedu.vip.mq.rabbit.springapi.
consumer.FirstConsumer"></bean>
```

```
<rabbit:listener-container connection-factory="connectionFactory">
    <rabbit:listener queues="MY_FIRST_QUEUE" ref="messageReceiver" />
</rabbit:listener-container>
```



9.3 Spring AMQP 核心对象

对象	描述
ConnectionFactory	Spring AMQP 的连接工厂接口，用于创建连接。CachingConnectionFactory 是 ConnectionFactory 的一个实现类。
RabbitAdmin	RabbitAdmin 是 AmqpAdmin 的实现，封装了对 RabbitMQ 的基础管理操作，比如对交换机、队列、绑定的声明和删除等。
Message	Message 是 Spring AMQP 对消息的封装。
RabbitTemplate	RabbitTemplate 是 AmqpTemplate 的一个实现（目前为止也是唯一的实现），用来简化消息的收发，支持消息的确认（Confirm）与返回（Return）。它封装了创建连接、创建消息信道、收发消息、消息格式转换（ConvertAndSend→Message）、关闭信道、关闭连接等等操作
MessageListener	MessageListener 是 Spring AMQP 异步消息投递的监听器接口，它只有一个方法 onMessage，用于处理消息队列推送来的消息，作用类似于 Java API 中的 Consumer
MessageListenerContainer	MessageListenerContainer 可以理解为 MessageListener 的容器，一个 Container 只有一个 Listener，但是可以生成多个线程使用相同的 MessageListener 同时消费消息。Container 可以管理 Listener 的生命周期，可以用于对于消费者进行配置。 例如：动态添加移除队列、对消费者进行设置，例如 ConsumerTag、Arguments、并发、消费者数量、消息确认模式等等。 在 SpringBoot2.0 中新增了一个 DirectMessageListenerContainer。
MessageListenerContainerFactory	可以在消费者上指定，当我们需要监听多个 RabbitMQ 的服务器的时候，指定不同的 MessageListenerContainerFactory
MessageConvertor	在调用 RabbitTemplate 的 convertAndSend() 方法发送消息时，会使用 MessageConvertor 进行消息的序列化，默认使用 SimpleMessageConverter。

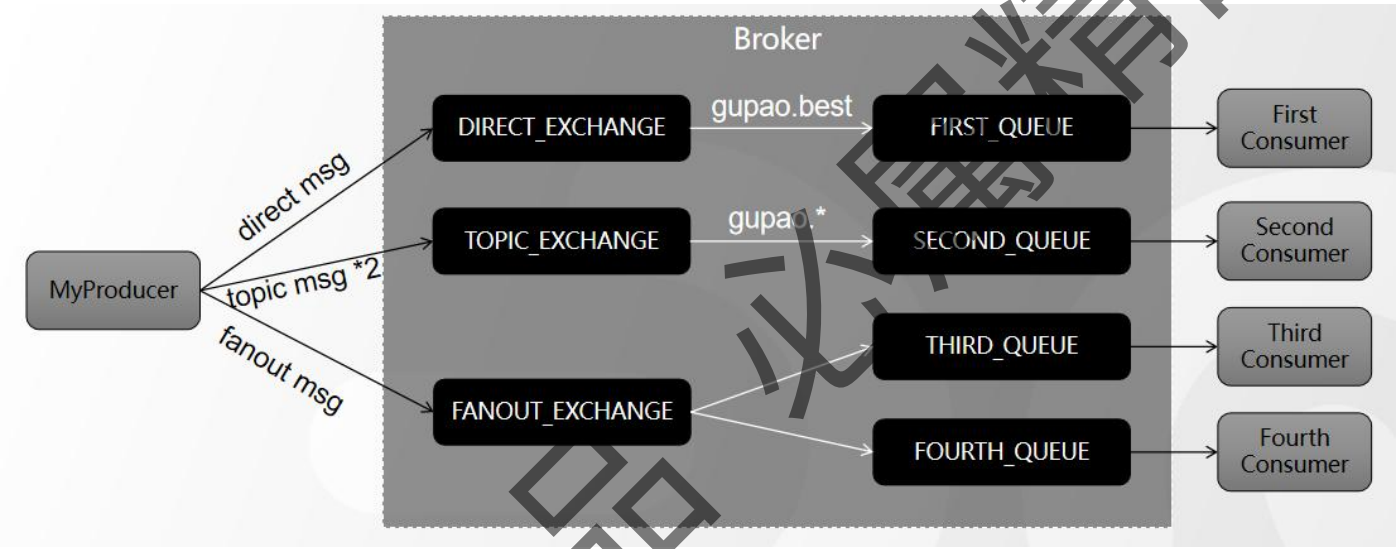
	在某些情况下，我们需要选择其他的高效的序列化工具。如果我们不想在每次发送消息时自己处理消息，就可以直接定义一个 MessageConvertor。

10、Spring Boot 集成 RabbitMQ

gupaoedu-vip-springboot-demo 工程中，为什么没有定义 Spring AMQP 的任何一个对象，也能实现消息的收发？Spring Boot 做了什么？

RabbitAutoConfiguration.java

参考工程：gupaoedu-vip-rabbitmq-springbootapi



3 个交换机与 4 个队列绑定。4 个消费者分别监听 4 个队列。

生产者发送 4 条消息，4 个队列收到 5 条消息。消费者打印出 5 条消息。

10.1 配置文件

RabbitConfig.java

定义交换机

```
@Bean("vipDirectExchange")
 public DirectExchange getDirectExchange(){
     return new DirectExchange(directExchange);
}
```

```
    @Bean("vipTopicExchange")
    public TopicExchange getTopicExchange(){
        return new TopicExchange(topicExchange);
    }

    @Bean("vipFanoutExchange")
    public FanoutExchange getFanoutExchange(){
        return new FanoutExchange(fanoutExchange);
    }
```

定义队列

```
    @Bean("vipFirstQueue")
    public Queue getFirstQueue(){
        return new Queue(firstQueue);
    }

    @Bean("vipSecondQueue")
    public Queue getSecondQueue(){
        return new Queue(secondQueue);
    }

    @Bean("vipThirdQueue")
    public  Queue getThirdQueue(){
        return  new Queue(thirdQueue);
    }

    @Bean("vipFourthQueue")
    public  Queue getFourthQueue(){
```

```
        return  new Queue(fourthQueue);
    }
```

定义绑定

```
    @Bean
    public Binding bindFirst(@Qualifier("vipFirstQueue") Queue queue,
@Qualifier("vipDirectExchange") DirectExchange exchange){
        return BindingBuilder.bind(queue).to(exchange).with("gupao.best");
    }


    @Bean
    public Binding bindSecond(@Qualifier("vipSecondQueue") Queue queue,
@Qualifier("vipTopicExchange") TopicExchange exchange){
        return BindingBuilder.bind(queue).to(exchange).with("*.gupao.*");
    }


    @Bean
    public Binding bindThird(@Qualifier("vipThirdQueue") Queue queue,
@Qualifier("vipFanoutExchange") FanoutExchange exchange){
        return BindingBuilder.bind(queue).to(exchange);
    }


    @Bean
    public Binding bindFourth(@Qualifier("vipFourthQueue") Queue queue,
@Qualifier("vipFanoutExchange") FanoutExchange exchange){
        return BindingBuilder.bind(queue).to(exchange);
    }
```

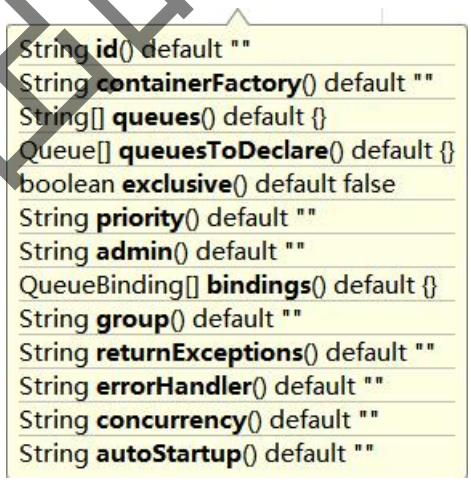
10.2 消费者

FirstConsumer.java

定义监听（后面三个消费者省略）；

在消费者类中可以有多个处理（不同类型的消息）的方法。

```
@Component
@PropertySource("classpath:gupaomq.properties")
@RabbitListener(queues = "${com.gupaoedu.firstqueue}")
public class FirstConsumer {


   @RabbitHandler
   public void process(@Payload Merchant merchant){
       System.out.println("First Queue received msg : " + merchant.getName());
  }
}
```



注解属性

10.3 生产者

RabbitSender.java

注入 RabbitTemplate 发送消息

```
@Autowired
AmqpTemplate gupaoTemplate;

public void send() throws JsonProcessingException {
    Merchant merchant =  new Merchant(1001,"a direct msg : 中原镖局","汉中省解放路 266 号");
    gupaoTemplate.convertAndSend(directExchange,directRoutingKey, merchant);

    gupaoTemplate.convertAndSend(topicExchange,topicRoutingKey1,"a topic msg : shanghai.gupao.teacher");
    gupaoTemplate.convertAndSend(topicExchange,topicRoutingKey2,"a topic msg : changsha.gupao.student");

    // 发送 JSON 字符串
    ObjectMapper mapper = new ObjectMapper();
    String json = mapper.writeValueAsString(merchant);
    System.out.println(json);
    gupaoTemplate.convertAndSend(fanoutExchange,"", json);
}
```


10.4 Spring Boot 参数解析

https://docs.spring.io/spring-boot/docs/2.1.6.RELEASE/reference/html/common-application-properties.html
https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

注：前缀 spring.rabbitmq.全部省略

全部配置总体上分成三大类：连接类、消息消费类、消息发送类

基于 Spring Boot 2.1.5

属性值	说明	默认值
address	客户端连接的地址，有多个的时候使用逗号分隔，该地	

	址可以是 IP 与 Port 的结合	
host	RabbitMQ 的主机地址	localhost
port	RabbitMQ 的端口号	
virtual-host	连接到 RabbitMQ 的虚拟主机	
username	登录到 RabbitMQ 的用户名	
password	登录到 RabbitMQ 的密码	
ssl.enabled	启用 SSL 支持	false
ssl.key-store	保存 SSL 证书的地址	
ssl.key-store-password	访问 SSL 证书的地址使用的密码	
ssl.trust-store	SSL 的可信地址	
ssl.trust-store-password	访问 SSL 的可信地址的密码	
ssl.algorithm	SSL 算法，默认使用 Rabbit 的客户端算法库	
cache.channel.checkout-timeout	当缓存已满时，获取 Channel 的等待时间，单位为毫秒	
cache.channel.size	缓存中保持的 Channel 数量	
cache.connection.mode	连接缓存的模式	CHANNEL
cache.connection.size	缓存的连接数	
connnection-timeout	连接超时参数单位为毫秒：设置为“0”代表无穷大	
dynamic	默认创建一个 AmqpAdmin 的 Bean	true
listener.simple.acknowledge-mode	容器的 acknowledge 模式	
listener.simple.auto-startup	启动时自动启动容器	true
listener.simple.concurrency	消费者的最小数量	
listener.simple.default-requeue-rejected	投递失败时是否重新排队	true
listener.simple.max-concurrency	消费者的最大数量	
listener.simple.missing-queues-fatal	容器上声明的队列不可用时是否失败	
listener.simple.prefetch	在单个请求中处理的消息个数，他应该大于等于事务数量	
listener.simple.retry.enabled	不论是不是重试的发布	false
listener.simple.retry.initial-interval	第一次与第二次投递尝试的时间间隔	1000ms
listener.simple.retry.max-attempts	尝试投递消息的最大数量	3
listener.simple.retry.max-interval	两次尝试的最大时间间隔	10000ms
listener.simple.retry.multiplier	上一次尝试时间间隔的乘数	1.0
listener.simple.retry.stateless	重试是有状态的还是无状态的	true
listener.simple.transaction-size	在一个事务中处理的消息数量。为了获得最佳效果，该值应设置为小于等于每个请求中处理的消息个数，即 listener.prefetch 的值	
publisher-confirms	开启 Publisher Confirm 机制	
publisher-returns	开启 Publisher Return 机制	
template.mandatory	启用强制信息	false
template.receive-timeout	receive()方法的超时时间	0
template.reply-timeout	sendAndReceive()方法的超时时间	5000
template.retry.enabled	设置为 true 的时候 RabbitTemplate 能够实现重试	false
template.retry.initial-interval	第一次与第二次发布消息的时间间隔	1000
template.retry.max-attempts	尝试发布消息的最大数量	3
template.retry.max-interval	尝试发布消息的最大时间间隔	10000

template.retry.multiplier	上一次尝试时间间隔的乘数	1.0