**RabbitMQ**

RabbitMQ 是轻量级的，易于在本地和云端部署。它支持多种消息传递协议。RabbitMQ 可以部署在分布式和联合配置中，以满足大规模、高可用性的要求。

* 支持多种消息传递协议、消息队列、传递确认、灵活的队列路由、多种交换类型。
* 使用BOSH、Chef、Docker 和 Puppet进行 部署。使用喜欢的编程语言开发跨语言消息传递，例如：Java、.NET、PHP、Python、JavaScript、Ruby、Go等。
* 部署为集群以获得高可用性和吞吐量；跨多个可用区和区域 联合。
* 可插拔认证、授权，支持TLS和LDAP。轻量级且易于部署在公共和私有云中。
* 支持持续集成、运营指标以及与其他企业系统集成 的各种工具和插件。用于扩展 RabbitMQ 功能 的灵活插件方法。
* HTTP-API、命令行工具和用于管理和监控RabbitMQ 的 UI。

[https://www.rabbitmq.com/](https://www.rabbitmq.com/#features)

**搭建RabbitMQ环境**

### Docker构建

# for RabbitMQ 3.9, the latest series

# timeline: https://www.rabbitmq.com/versions.html

docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.9-management

### Yum 安装软件包

**导入以下密钥**

## 主 RabbitMQ 签名密钥

rpm --import https://github.com/rabbitmq/signing-keys/releases/download/2.0/rabbitmq-release-signing-key.asc

##现代Erlang存储库

rpm --import https://packagecloud.io/rabbitmq/erlang/gpgkey

## RabbitMQ 服务器存储库

rpm --import https://packagecloud.io/rabbitmq/rabbitmq-server/gpgkey

**添加数据源**

vim /etc/yum.repos.d/rabbitmq.repo

##

## Zero dependency Erlang

##

[rabbitmq\_erlang]

name=rabbitmq\_erlang

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

repo\_gpgcheck=1

gpgcheck=1

enabled=1

# PackageCloud's repository key and RabbitMQ package signing key

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

https://github.com/rabbitmq/signing-keys/releases/download/2.0/rabbitmq-release-signing-key.asc

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

##

## RabbitMQ server

##

[rabbitmq\_server]

name=rabbitmq\_server

baseurl=https://packagecloud.io/rabbitmq/rabbitmq-server/el/7/$basearch

repo\_gpgcheck=1

gpgcheck=1

enabled=1

# PackageCloud's repository key and RabbitMQ package signing key

gpgkey=https://packagecloud.io/rabbitmq/rabbitmq-server/gpgkey

https://github.com/rabbitmq/signing-keys/releases/download/2.0/rabbitmq-release-signing-key.asc

sslverify=1

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

metadata\_expire=300

[rabbitmq\_server-source]

name=rabbitmq\_server-source

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

repo\_gpgcheck=1

gpgcheck=0

enabled=1

gpgkey=https://packagecloud.io/rabbitmq/rabbitmq-server/gpgkey

sslverify=1

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

metadata\_expire=300

**更新数据源**

yum update –y

**开始安装**

## install these dependencies from standard OS repositories

yum install socat logrotate -y

yum install erlang rabbitmq-server -y

**开启web插件**

rabbitmq-plugins enable rabbitmq\_management

启动服务

**#启动**

systemctl start rabbitmq-server

**#查看启动是否正常**

systemctl status rabbitmq-server

**添加一个用户**

rabbitmqctl add\_user 用户名 密码

**配置权限**

rabbitmqctl set\_permissions -p "/" 用户名 ".\*" ".\*" ".\*"

**查看用户权限**

rabbitmqctl list\_user\_permissions 用户名

**设置用户组**

rabbitmqctl set\_user\_tags 用户名 administrator

http://47.105.133.123:15672/

## 配置项

RabbitMQ 带有默认的内置设置。在某些环境（例如开发和质量保证）中，这些可能完全足够。对于所有其他情况，以及生产部署调整，有一种方法可以在代理和插件中配置许多东西。

### 配置方式

| **机制** | **描述** |
| --- | --- |
| 配置文件 | 包含服务器和插件设置   * [TCP 侦听器和其他网络相关设置](https://www.rabbitmq.com/networking.html) * [TLS](https://www.rabbitmq.com/ssl.html) * [资源限制（警报）](https://www.rabbitmq.com/alarms.html) * [身份验证和授权后端](https://www.rabbitmq.com/access-control.html)  * [消息存储设置](https://www.rabbitmq.com/persistence-conf.html) |
| 环境变量 | 定义节点名称、文件和目录位置、从 shell 获取的运行时标志，或在环境配置文件rabbitmq-env.conf（Linux、MacOS、BSD）和rabbitmq-env-conf.bat（Windows） 中设置 |
| rabbitmqctl | 当使用内部认证/授权后端时， rabbitmqctl是管理虚拟主机、用户和权限的工具。它还用于管理运行时参数和策略。 |
| rabbitmq-queues | rabbitmq-queues是管理特定于quorum queues的设置的工具。 |
| rabbitmq-plugins | rabbitmq-plugins是管理插件的工具。 |
| rabbitmq-diagnostics | rabbitmq-diagnostics允许检查节点状态，包括有效配置，以及许多其他指标和健康检查。 |
| [参数和策略](https://www.rabbitmq.com/parameters.html) | 定义可以在运行时更改的集群范围的设置以及便于为队列组（交换等）配置的设置，例如包括可选的队列参数。 |
| 运行时（Erlang VM）标志 | 控制系统的较低级别方面：内存分配设置、节点间通信缓冲区大小、运行时调度程序设置等。 |
| [操作系统内核限制](https://www.rabbitmq.com/configure.html#kernel-limits) | 控制内核强制执行的进程限制：最大打开文件句柄限制、最大进程和内核线程数、最大驻留集大小等。 |

大多数设置都是使用前两种方法配置的。

### 配置文件

配置文件：

* /etc/rabbitmq/advanced.config
* /etc/rabbitmq/rabbitmq.conf

使用rabbitmq-diagnostics status命令：

rabbitmq-diagnostics status

要检查特定节点的位置，包括远程运行的节点，请使用-n（--node 的缩写）开关：

rabbitmq-diagnostics status -n [节点名称]

默认配置文件位置是特定于发行版的。RabbitMQ 包或节点不会创建任何配置文件。创建文件时，用户和部署工具应使用以下位置：

| **平台** | **默认配置文件目录** | **示例配置文件路径** |
| --- | --- | --- |
| 通用二进制包 | $RABBITMQ\_HOME/etc/rabbitmq/ | $RABBITMQ\_HOME/etc/rabbitmq/rabbitmq.conf ,  $RABBITMQ\_HOME/etc/rabbitmq/advanced.config |
| Debian 和 Ubuntu | /etc/rabbitmq/ | /etc/rabbitmq/rabbitmq.conf ,  /etc/rabbitmq/advanced.config |
| 基于 RPM 的 Linux | /etc/rabbitmq/ | /etc/rabbitmq/rabbitmq.conf ,  /etc/rabbitmq/advanced.config |
| 视窗 | %APPDATA%\RabbitMQ\ | %APPDATA%\RabbitMQ\rabbitmq.conf ,  %APPDATA%\RabbitMQ\advanced.config |
| MacOS 自制包 | ${install\_prefix}/etc/rabbitmq/，Homebrew 自制前缀通常是/usr/local | ${install\_prefix}/etc/rabbitmq/rabbitmq.conf ,  ${install\_prefix}/etc/rabbitmq/advanced.config |

环境变量可用于覆盖配置文件的位置：

1. # 覆盖主配置文件位置
2. RABBITMQ\_CONFIG\_FILE =/path/to/a/custom/location/rabbitmq.conf
3. # 覆盖高级配置文件位置
4. RABBITMQ\_ADVANCED\_CONFIG\_FILE=/path/to/a/custom/location/advanced.config
5. # 覆盖环境变量文件位置
6. RABBITMQ\_CONF\_ENV\_FILE =/path/to/a/custom/location/rabbitmq-env.conf

在大多数发行版中，示例文件放置在与实际文件应该放置的位置相同的位置（见上文）。在 Debian 和 RPM 发行版中，政策禁止这样做；而是分别在/usr/share/doc/rabbitmq-server/ 或/usr/share/doc/rabbitmq-server-3.9.13/下找到文件。

### 配置文件格式

所有支持的 RabbitMQ 版本都对主配置文件使用类似 ini 的 sysctl 配置文件格式 。该文件通常命名为rabbitmq.conf。新的配置格式更简单，更易于人类阅读和机器生成。与 RabbitMQ 3.7.0 之前使用的经典配置格式相比，它也相对有限。例如，在配置LDAP 支持时，可能需要使用深度嵌套的数据结构来表达所需的配置。为了满足这种需求，现代 RabbitMQ 版本允许在单独的文件中同时使用两种格式：rabbitmq.conf使用新样式格式，建议用于大多数设置，advanced.config涵盖了比 ini-风格配置无法表达

| **配置文件** | **使用的格式** | **目的** |
| --- | --- | --- |
| rabbitmq.conf | 新样式格式（sysctl 或类似 ini） | 主配置文件。应该用于大多数设置。人类更容易阅读，机器（部署工具）更容易生成。不是每个设置都可以用这种格式表示。 |
| 高级配置 | 经典（Erlang 术语） | 无法以新样式配置格式表达的有限数量的设置，例如LDAP 查询。仅应在必要时使用。 |
| rabbitmq-env.conf（在 Windows 上为 rabbitmq-env.conf.bat） | 环境变量对 | 用于在一处设置与 RabbitMQ 相关的 环境变量。 |

### 主配置文件 rabbitmq.conf

配置文件rabbitmq.conf 允许配置 RabbitMQ 服务器和插件。从 RabbitMQ 3.7.0 开始，格式为sysctl 格式。语法可以用 3 行简单解释：

* 一种设置使用一条线
* 行是结构化的Key = Value
* 任何以#字符开头的行都是注释

一个简约的示例配置文件如下：

1. listeners.tcp.**default** = 5673

要覆盖主 RabbitMQ 配置文件位置，请使用RABBITMQ\_CONFIG\_FILE 环境变量。使用.conf作为新样式配置格式的文件扩展名，例如/etc/rabbitmq/rabbitmq.conf或 /data/configuration/rabbitmq/rabbitmq.conf

配置项解析：

|  |  |
| --- | --- |
| **key** | **文档** |
| listeners | 监听“普通”AMQP 0-9-1 和 AMQP 1.0 连接的端口或主机名/对（无TLS）。  默认：listeners.tcp.default = 5672 |
| num\_acceptors.tcp | 将接受 TCP 侦听器连接的 Erlang 进程数。  默认：num\_acceptors.tcp = 10 |
| handshake\_timeout | AMQP 0-9-1 握手的最长时间（在套接字连接和 TLS 握手之后），以毫秒为单位。  默认：握手超时= 10000 |
| listeners.ssl | 侦听启用 TLS 的 AMQP 0-9-1 和 AMQP 1.0 连接的端口或主机名/对。  默认值：无 |
| num\_acceptors.ssl | 将接受来自客户端的 TLS 连接的 Erlang 进程数。  默认：num\_acceptors.ssl = 10 |
| ssl\_options | TLS 配置。  默认：ssl\_options = 无 |
| ssl\_handshake\_timeout | TLS 握手超时，以毫秒为单位。  默认：ssl\_handshake\_timeout = 5000 |
| vm\_memory\_high\_watermark | 触发流控制的内存阈值。可以是绝对的或相对于操作系统可用的 RAM 量，例如：  vm\_memory\_high\_watermark.relative = 0.6  vm\_memory\_high\_watermark.absolute = 2 GB  默认：vm\_memory\_high\_watermark.relative = 0.4 |
| vm\_memory\_calculation\_strategy | 内存使用报告策略。可以是以下之一：   * 已分配：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，并且可能会启动短暂的子进程。 * legacy：使用遗留内存报告（运行时认为使用了多少内存）。这种策略是相当不准确的。 * erlang：与legacy相同，为向后兼容而保留   默认：vm\_memory\_calculation\_strategy = 已分配 |
| vm\_memory\_high\_watermark\_paging\_ratio | 队列开始将消息分页到磁盘以释放内存的高水位线限制的一部分。  默认：  vm\_memory\_high\_watermark\_paging\_ratio = 0.5 |
| total\_memory\_available\_override\_value | 可以覆盖可用内存总量，而不是使用特定于操作系统的方法从环境中推断出来。这仅应在节点可用的实际最大 RAM 量与节点推断的值不匹配时使用，例如由于容器化或节点无法意识到的类似限制。该值可以设置为整数字节数，或者以信息单位（例如“8GB”）。例如，当该值设置为 4 GB 时，节点会认为它运行在具有 4 GB RAM 的机器上。  默认值：未定义（未设置或使用）。 |
| disk\_free\_limit | RabbitMQ 存储数据的分区的磁盘可用空间限制。当可用磁盘空间低于此限制时，将触发流量控制。该值可以相对于 RAM 的总量设置，也可以设置为以字节为单位的绝对值，或者以信息单位（例如“50MB”或“5GB”）为单位：  disk\_free\_limit.relative = 3.0  disk\_free\_limit.absolute = 2 GB  默认情况下，可用磁盘空间必须超过 50MB。  默认：disk\_free\_limit.absolute = 50 MB |
| log.file.level | 控制日志记录的粒度。该值是日志事件类别和日志级别对的列表。  默认：log.file.level = info |
| channel\_max | 与客户端协商的最大允许通道数，不包括协议中使用的特殊通道号 0。设置为 0 意味着“无限制”，这是一个危险的值，因为应用程序有时会出现通道泄漏。使用更多通道会增加代理的内存占用。  默认：channel\_max = 2047 |
| channel\_operation\_timeout | 通道操作超时，以毫秒为单位（内部使用，由于消息传递协议的差异和限制，不直接暴露给客户端）。  默认：channel\_operation\_timeout = 15000 |
| max\_message\_size | 允许的最大消息有效负载大小（以字节为单位）。较大的消息将被拒绝，并出现适当的通道异常。  默认值：134217728  最大值：536870912 |
| heartbeat | 表示服务器在连接参数协商过程中建议的心跳超时值。如果两端都设置为 0，则禁用心跳（不建议这样做）。  Default:heartbeat = 60 |
| default\_vhost | 当 RabbitMQ 从头开始​​创建新数据时要创建的虚拟主机。交换 `amq.rabbitmq.log` 将存在于这个虚拟主机中。  默认：default\_vhost = / |
| default\_user | RabbitMQ 从头开始​​创建新数据时要创建的用户名。  默认：default\_user = guest |
| default\_pass | 默认用户的密码。  默认：default\_pass = guest |
| default\_user\_tags | 默认用户的标签。  默认：default\_user\_tags.administrator = true |
| default\_permissions | 创建默认用户时分配给默认用户的权限。  默认：  default\_permissions.configure = .\*  default\_permissions.read = .\*  default\_permissions.write = .\* |
| loopback\_users | 仅允许通过环回接口（即“localhost”）连接到代理的用户列表。要允许默认的 `guest` 用户远程连接（不适合生产使用的安全做法），请将其设置为 `none`：  loopback\_users = none  要将另一个用户限制为仅限本地主机的连接，请这样做（`monitoring` 是用户的名称）：  loopback\_users.monitoring = true  默认：  # 来宾使用众所周知的  # 凭据并且只能  # 从本地主机登录  # 默认情况下  loopback\_users.guest = true |
| cluster\_formation.classic\_config.nodes | 经典对等发现后端要联系的节点列表。例如，在第一次启动时使用节点 `rabbit@hostname1`和`rabbit@hostname2` 进行集群：  cluster\_formation.classic\_config.nodes.1 = rabbit@hostname1  cluster\_formation.classic\_config.nodes.2 = rabbit@hostname2  默认值：`none`（未设置） |
| collect\_statistics | 统计收集模式。主要与管理插件相关。选项是：   * `none`（不发出统计事件） * `coarse`（发送每个队列/每个通道/每个连接的统计信息） * `fine`（也发出每条消息的统计信息）   默认：collect\_statistics = none |
| collect\_statistics\_interval | 以毫秒为单位的统计信息收集间隔。主要与管理插件相关。  默认：collect\_statistics\_interval = 5000 |
| management\_db\_cache\_multiplier | 缓存会将最后一次查询的经过时间乘以该值，并将结果缓存这段时间。  默认：management\_db\_cache\_multiplier = 5 |
| auth\_mechanisms | 提供给客户端的 SASL 身份验证机制。  默认：  auth\_mechanisms.1 = PLAIN  auth\_mechanisms.2 = AMQPLAIN |
| auth\_backends | 要使用的身份验证和授权后端列表。除了 `rabbit\_auth\_backend\_internal` 之外的其他数据库可以通过plugins获得。  默认：auth\_backends.1 = internal |
| reverse\_dns\_lookups | 设置为 `true` 让 RabbitMQ 在客户端连接上执行反向 DNS 查找，并通过 `rabbitmqctl` 和管理插件显示该信息。  默认：reverse\_dns\_lookups =false |
| delegate\_count | 用于集群内通信的委托进程数。  默认：delegate\_count = 16 |
| tcp\_listen\_options | 默认套接字选项。  默认：  tcp\_listen\_options.backlog = 128  tcp\_listen\_options.nodelay = true  tcp\_listen\_options.linger.on = true  tcp\_listen\_options.linger.timeout = 0  tcp\_listen\_options.exit\_on\_close = false |
| hipe\_compile | 不使用。不再支持此选项。从 Erlang 22 开始不再支持 HiPE。  默认：hipe\_compile = false |
| cluster\_partition\_handling | 如何处理网络分区。可用模式有：   * ignore * autoheal * pause\_minority * pause\_if\_all\_down   pause\_if\_all\_down模式需要额外的参数：   * nodes * recover   默认：cluster\_partition\_handling = ignore |
| cluster\_keepalive\_interval | 节点应该多久向其他节点发送保活消息（以毫秒为单位）。请注意，这与net\_ticktime 不同；错过的 keepalive 消息不会导致节点被视为关闭。  默认：cluster\_keepalive\_interval = 10000 |
| queue\_index\_embed\_msgs\_below | 以字节为单位的消息大小，低于该消息将直接嵌入到队列索引中。  默认：queue\_index\_embed\_msgs\_below = 4096 |
| mnesia\_table\_loading\_retry\_timeout | 等待集群中的 Mnesia 表可用时使用的超时。  默认：mnesia\_table\_loading\_retry\_timeout = 30000 |
| mnesia\_table\_loading\_retry\_limit | 在集群启动中等待 Mnesia 表时重试。请注意，此设置不适用于 Mnesia 升级或节点删除。  默认：mnesia\_table\_loading\_retry\_limit = 10 |
| mirroring\_sync\_batch\_size | 用于将消息传输到非同步副本（队列镜像）的批量大小。  默认：mirroring\_sync\_batch\_size = 4096 |
| queue\_master\_locator | 队列领导者定位策略。可用的策略是：   * min-masters * client-local * random   默认：queue\_master\_locator = client-local |
| proxy\_protocol | 如果设置为true，RabbitMQ 将期望在打开 AMQP 连接时首先发送代理协议头。这意味着在 RabbitMQ 前面设置一个符合代理协议的反向代理（例如HAproxy 或AWS ELB ）。启用代理协议后，客户端无法直接连接到 RabbitMQ，因此所有连接都必须经过反向代理。  默认：proxy\_protocol = false |
| cluster\_name | 操作员控制的集群名称。此名称用于标识集群，并由 federation 和 Shovel 插件记录传输消息的来源或路径。可以设置为任意字符串以帮助识别集群（例如london）。AMQP 0-9-1 客户端可以在服务器属性映射中检查此名称。  默认值：默认情况下，名称源自集群中的第一个（种子）节点。 |

### 高级配置文件

使用 sysctl 格式无法或难以配置某些配置设置。因此，可以使用 Erlang 术语格式的附加配置文件（与rabbitmq.config相同）。该文件通常命名为advanced.config。它将与rabbitmq.conf中提供的配置合并。要覆盖高级配置文件位置，请使用RABBITMQ\_ADVANCED\_CONFIG\_FILE 环境变量。

配置项解析：

|  |  |
| --- | --- |
| **key** | **文档** |
| msg\_store\_index\_module | 队列索引的实现模块。  默认值：rabbit\_msg\_store\_ets\_index  { rabbit，[  {msg\_store\_index\_module，rabbit\_msg\_store\_ets\_index}  ]} |
| backing\_queue\_module | 队列内容的实现模块。  默认：  { rabbit，[  {backing\_queue\_module，rabbit\_variable\_queue}  ]} |
| msg\_store\_file\_size\_limit | 消息存储段文件大小。为具有现有（已初始化）数据库的节点更改此设置是危险的，可能会导致数据丢失！  默认值：16777216  {rabbit, [  %%使用现有（已初始化）数据为节点 %% 更改此设置是危险的，可能  %% 导致数据丢失！  {msg\_store\_file\_size\_limit, 16777216 }  ]} |
| trace\_vhosts | 由跟踪器内部使用。  默认：  { rabbit，[  {trace\_vhosts，[]}  ]} |
| msg\_store\_credit\_disc\_bound | 该值仅在消息持久化到消息存储时生效。如果消息嵌入在队列索引中，则修改此设置无效，因为在写入队列索引时不使用 credit\_flow。  默认：  { rabbit，[  {msg\_store\_credit\_disc\_bound, { 4000 , 800 }}  ]} |
| queue\_index\_max\_journal\_entries | 在多少个队列索引日志条目之后，它将被刷新到磁盘。  默认：  { rabbit，[  {queue\_index\_max\_journal\_entries，32768 }  ]} |
| lazy\_queue\_explicit\_gc\_run\_operation\_threshold | 仅适用于在内存压力下的惰性队列的可调值。这是触发垃圾收集器和其他内存减少活动的阈值。较低的值可能会降低性能，较高的值可以提高性能，但会导致更高的内存消耗。  默认：  { rabbit，[  {lazy\_queue\_explicit\_gc\_run\_operation\_threshold，1000}  ] } |
| queue\_explicit\_gc\_run\_operation\_threshold | 仅适用于内存压力下的正常队列的可调值。这是触发垃圾收集器和其他内存减少活动的阈值。较低的值可能会降低性能，较高的值可以提高性能，但会导致更高的内存消耗。  默认：  {rabbit，[  {queue\_explicit\_gc\_run\_operation\_threshold，1000}  ] } |

### 配置值加密

敏感的配置条目（例如密码、包含凭据的 URL）可以在 RabbitMQ 配置文件中加密。代理在开始时解密加密条目。加密值必须在 Erlang加密 元组内：{encrypted, ...}。以下是默认用户的加密密码配置文件示例：

1. [
2. {rabbit, [
3. {default\_user, << "guest" >>},
4. {default\_pass,
5. {encrypted,
6. << "cPAymwqmMnbPXXRVqVzpxJdrS8mHEKuo2V+3vt1u/fymexD9oztQ2G/oJ4PAaSb2c5N/hRJ2aqP/X0VAfx8xOQ==" >>
7. }
8. },
9. {config\_entry\_decoder, [
10. {密码，<< “我的密码” >>}
11. ]}
12. ]}
13. ]。

密码短语不必在配置文件中硬编码，它可以在单独的文件中：

1. [
2. {rabbit, [ %% ...
3. {config\_entry\_decoder, [
4. {passphrase, {file, "/path/to/passphrase/file" }}
5. ]}
6. ]}
7. ]。

### rabbitmq环境变量配置

在基于 UNIX 的系统（Linux、MacOS 和 BSD 风格）上，可以使用名为rabbitmq-env.conf的文件 来定义代理将使用的环境变量。其位置可使用RABBITMQ\_CONF\_ENV\_FILE环境变量进行配置。rabbitmq-env.conf使用标准环境变量名称，但没有RABBITMQ\_前缀。例如，RABBITMQ\_CONFIG\_FILE变量在下面显示为CONFIG\_FILE并且 RABBITMQ\_NODENAME变为NODENAME。

RabbitMQ 使用的所有环境变量都使用前缀RABBITMQ\_（除非在rabbitmq-env.conf或 rabbitmq-env-conf.bat 中定义）。在 shell 环境中设置的环境变量优先于在rabbitmq-env.conf或 rabbitmq-env-conf.bat中设置的那些，这反过来会覆盖 RabbitMQ 内置默认值。

可用于配置 RabbitMQ 的关键环境变量：

|  |  |
| --- | --- |
| **名称** | **描述** |
| RABBITMQ\_NODE\_IP\_ADDRESS | 如果您只想绑定到一个网络接口，请更改此设置。可以在配置文件中设置绑定到两个或多个接口。  默认值：空字符串，表示“绑定到所有网络接口”。 |
| RABBITMQ\_NODE\_PORT | 默认值：5672。 |
| RABBITMQ\_DIST\_PORT | 用于节点间和 CLI 工具通信的端口。如果节点配置文件设置kernel.inet\_dist\_listen\_min或 kernel.inet\_dist\_listen\_max键，则忽略。有关详细信息，请参阅[网络](https://www.rabbitmq.com/networking.html)，有关特定于 Windows 的详细信息， 请参阅[Windows Quirks](https://www.rabbitmq.com/windows-quirks.html)。  默认值：RABBITMQ\_NODE\_PORT + 20000 |
| ERL\_EPMD\_ADDRESS | [epmd](https://www.rabbitmq.com/networking.html" \l "epmd) 使用的接口，节点间和 CLI 工具通信中的组件。  默认值：所有可用接口，包括 IPv6 和 IPv4。 |
| ERL\_EPMD\_PORT | [epmd](https://www.rabbitmq.com/networking.html" \l "epmd) 使用的端口，节点间和 CLI 工具通信中的组件。  **默认值**：4369 |
| RABBITMQ\_DISTRIBUTION\_BUFFER\_SIZE | 用于节点间通信连接的[传出数据缓冲区大小限制，以千字节为单位。](https://erlang.org/doc/man/erl.html" \l "+zdbbl)不建议使用低于 64 MB 的值。  **默认值**：128000 |
| RABBITMQ\_NODENAME | 每个 Erlang-node-and-machine 组合的节点名称应该是唯一的。要运行多个节点，请参阅[集群指南](https://www.rabbitmq.com/clustering.html)。  **默认**：  Unix\*： rabbit@$HOSTNAME  Windows： rabbit @%COMPUTERNAME% |
| RABBITMQ\_CONFIG\_FILE | 主 RabbitMQ 配置文件路径，例如 /etc/rabbitmq/rabbitmq.conf或/data/configuration/rabbitmq.conf用于新样式的配置格式文件。如果使用经典配置格式，则扩展名必须为.config  **默认：**   * 通用 UNIX：$RABBITMQ\_HOME/etc/rabbitmq/rabbitmq.conf * Debian：/etc/rabbitmq/rabbitmq.conf * RPM：/etc/rabbitmq/rabbitmq.conf * MacOS(Homebrew) : ${install\_prefix}/etc/rabbitmq/rabbitmq.conf , Homebrew 前缀通常是/usr/local * 视窗：%APPDATA%\RabbitMQ\rabbitmq.conf |
| RABBITMQ\_ADVANCED\_CONFIG\_FILE | * “高级”（基于 Erlang 术语的）RabbitMQ 配置文件路径，带有.config文件扩展名。例如，/data/rabbitmq/advanced.config。 * **默认**： * 通用 UNIX：$RABBITMQ\_HOME/etc/rabbitmq/advanced.config * Debian：/etc/rabbitmq/advanced.config * RPM：/etc/rabbitmq/advanced.config * MacOS (Homebrew) : ${install\_prefix}/etc/rabbitmq/advanced.config , Homebrew 前缀通常是/usr/local * Windows：%APPDATA%\RabbitMQ\advanced.config |
| RABBITMQ\_CONF\_ENV\_FILE | * 包含环境变量定义的文件的位置（没有RABBITMQ\_ 前缀）。请注意，Windows 上的文件名与其他操作系统不同。 * **默认**： * 通用 UNIX 包：$RABBITMQ\_HOME/etc/rabbitmq/rabbitmq-env.conf * Ubuntu 和 Debian：/etc/rabbitmq/rabbitmq-env.conf * RPM：/etc/rabbitmq/rabbitmq-env.conf * MacOS (Homebrew) : ${install\_prefix}/etc/rabbitmq/rabbitmq-env.conf , Homebrew 前缀通常是/usr/local * Windows：%APPDATA%\RabbitMQ\rabbitmq-env-conf.bat |
| RABBITMQ\_MNESIA\_BASE | * 此基本目录包含 RabbitMQ 服务器的节点数据库、消息存储和集群状态文件的子目录，每个节点一个，除非明确设置RABBITMQ\_MNESIA\_DIR。重要的是，有效的 RabbitMQ 用户有足够的权限随时在此目录中读取、写入和创建文件和子目录。此变量通常不会被覆盖。通常RABBITMQ\_MNESIA\_DIR会被覆盖。 * **默认**： * 通用 UNIX 包：$RABBITMQ\_HOME/var/lib/rabbitmq/mnesia * Ubuntu 和 Debian软件包：/var/lib/rabbitmq/mnesia/ * RPM：/var/lib/rabbitmq/plugins * MacOS (Homebrew) : ${install\_prefix}/var/lib/rabbitmq/mnesia , Homebrew 前缀通常是/usr/local * 视窗：%APPDATA%\RabbitMQ |
| RABBITMQ\_MNESIA\_DIR | * 存储此 RabbitMQ 节点数据的目录。这包括模式数据库、消息存储、集群成员信息和其他持久节点状态。 * **默认**： * 通用 UNIX 包：$RABBITMQ\_MNESIA\_BASE/$RABBITMQ\_NODENAME * Ubuntu 和 Debian软件包：$RABBITMQ\_MNESIA\_BASE/$RABBITMQ\_NODENAME * 转：$RABBITMQ\_MNESIA\_BASE/$RABBITMQ\_NODENAME * MacOS (Homebrew) : ${install\_prefix}/var/lib/rabbitmq/mnesia/$RABBITMQ\_NODENAME , Homebrew 前缀通常是/usr/local * Windows：%APPDATA%\RabbitMQ\$RABBITMQ\_NODENAME |
| RABBITMQ\_PLUGINS\_DIR | * [插件](https://www.rabbitmq.com/plugins.html)存档文件 所在的目录列表并从中提取。这是类似PATH的变量，其中不同的路径由特定于操作系统的分隔符分隔（:用于 Unix，;用于 Windows）。插件可以[安装](https://www.rabbitmq.com/plugins.html)到此处列出的任何目录中。不得包含[路径限制部分](https://www.rabbitmq.com/configure.html#directory-and-path-restrictions)中提到的任何字符。 * **默认**： * 通用 UNIX 包：$RABBITMQ\_HOME/plugins * Ubuntu 和 Debian软件包：/var/lib/rabbitmq/plugins * RPM：/var/lib/rabbitmq/plugins * MacOS (Homebrew) : ${install\_prefix}/Cellar/rabbitmq/${version}/plugins , Homebrew 前缀通常是/usr/local * Windows : %RABBITMQ\_HOME%\plugins |
| RABBITMQ\_PLUGINS\_EXPAND\_DIR | * 节点展开（解包）[插件](https://www.rabbitmq.com/plugins.html)的目录并将其用作代码路径位置。不得包含[路径限制部分](https://www.rabbitmq.com/configure.html#directory-and-path-restrictions)中提到的任何字符。 * **默认**： * 通用 UNIX 包：$RABBITMQ\_MNESIA\_BASE/$RABBITMQ\_NODENAME-plugins-expand * Ubuntu 和 Debian软件包：$RABBITMQ\_MNESIA\_BASE/$RABBITMQ\_NODENAME-plugins-expand * RPM：$RABBITMQ\_MNESIA\_BASE/$RABBITMQ\_NODENAME-plugins-expand * MacOS（自制）： ${install\_prefix}/var/lib/rabbitmq/mnesia/$RABBITMQ\_NODENAME-plugins-expand * Windows：%APPDATA%\RabbitMQ\$RABBITMQ\_NODENAME-plugins-expand |
| RABBITMQ\_USE\_LONGNAME | * 当设置为true时，这将导致 RabbitMQ 使用完全限定的名称来标识节点。这在使用完全限定域名或使用 IP 地址作为主机名或节点名的一部分的环境中可能很有用。请注意，如果不重置节点，则无法将节点从短名称切换为长名称。 * **默认值**：假 |
| RABBITMQ\_SERVICENAME | * 已安装的 Windows 服务的名称。这将出现在 services.msc中。 * **默认值**：RabbitMQ。 |
| RABBITMQ\_CONSOLE\_LOG | * 将此变量设置为new或重新使用以将控制台输出从服务器重定向到默认RABBITMQ\_BASE目录 中 名为 %RABBITMQ\_SERVICENAME%的文件。 * 如果未设置，来自服务器的控制台输出将被丢弃（默认）。 * new：每次服务启动时都会创建一个新文件。 * 重用：每次服务启动时文件都会被覆盖。 * **默认值**：（无） |
| RABBITMQ\_SERVER\_CODE\_PATH | * 启动运行时指定的额外代码路径（目录）。将在节点启动时 传递给erl命令。 * **默认值**：（无） |
| RABBITMQ\_CTL\_ERL\_ARGS | * 调用 rabbitmqctl时使用 的erl命令的参数。这可以设置为指定用于 Erlang 分发的端口范围：-kernel inet\_dist\_listen\_min 35672 -kernel inet\_dist\_listen\_max 35680 * **默认值**：（无） |
| RABBITMQ\_SERVER\_ERL\_ARGS | * 调用 RabbitMQ 服务器时使用的erl命令 的标准参数。这应该仅出于调试目的而被覆盖。覆盖此变量 *将替换*默认值。 * **默认**： * Unix\*: +P 1048576 +t 5000000 +stbt db +zdbbl 128000 * 窗户：无 |
| RABBITMQ\_SERVER\_ADDITIONAL\_ERL\_ARGS | * 调用 RabbitMQ 服务器时使用的erl命令 的附加参数。此变量的值附加到默认参数列表 ( RABBITMQ\_SERVER\_ERL\_ARGS )。 * **默认**： * Unix\*：无 * 窗户：无 |
| RABBITMQ\_SERVER\_START\_ARGS | * 调用 RabbitMQ 服务器时使用的erl命令 的额外参数。这不会覆盖 RABBITMQ\_SERVER\_ERL\_ARGS。 * **默认值**：（无） |
| RABBITMQ\_DEFAULT\_USER | * 此环境变量**仅用于开发和 CI 环境**。这与rabbitmq.conf中的default\_user含义相同，但优先级更高。在无法提供配置文件的情况下，此选项可能更方便，并且环境变量是为[用户播种](https://www.rabbitmq.com/access-control.html#seeding)的唯一方法。 * **默认值**：（无） |
| RABBITMQ\_DEFAULT\_PASS | * 此环境变量**仅用于开发和 CI 环境**。这与rabbitmq.conf中的default\_pass含义相同，但优先级更高。在无法提供配置文件的情况下，此选项可能更方便，并且环境变量是为[用户播种](https://www.rabbitmq.com/access-control.html#seeding)的唯一方法。 * **默认值**：（无） |
| RABBITMQ\_DEFAULT\_VHOST | * 此环境变量**仅用于开发和 CI 环境**。这与rabbitmq.conf中的default\_vhost含义相同，但优先级更高。[在无法提供配置文件并且环境变量是播种用户](https://www.rabbitmq.com/access-control.html#seeding)和虚拟主机 的唯一方法的情况下，此选项可能更方便。 * **默认值**：（无） |

## 管理界面

RabbitMQ 管理插件提供基于 HTTP 的 API 用于管理和监视 RabbitMQ 节点和集群，以及基于浏览器的 UI 和命令行工具rabbitmqadmin。它定期收集和汇总有关系统许多方面的数据。这些指标在 UI 和监控系统中都暴露给操作员，用于长期存储、警报、可视化、图表分析等。

该插件可以配置为使用 HTTPS、OAuth 2、非标准端口、路径前缀、HTTP 服务器选项、自定义严格传输安全设置、 跨域资源共享等。一些设置直接影响度量收集系统和这个插件的 CPU 资源使用：

* 统计数据收集间隔
* 使用的消息速率模式（速率保真度）

该插件还提供了用于分析节点内存使用情况的工具，以及与监控、指标、用户、权限和拓扑管理相关的其他功能。以前它还提供定义导出和导入功能。这些现在是核心 RabbitMQ 功能，不需要或依赖此插件。在多节点集群中，管理插件最常在每个节点上启用。

### 管理 UI 和外部监控系统

管理 UI 及其HTTP API是 RabbitMQ 的内置监控选项。对于开发难或不可能引入外部监控的环境，这是一个方便的选择。但是，管理 UI 有许多限制：

* 监控系统与被监控系统交织在一起
* 一定的开销
* 它只存储最近的数据（想想几个小时，而不是几天或几个月）
* 它有一个基本的用户界面
* 它的设计强调易用性而不是最佳可用性。
* 管理 UI 访问通过RabbitMQ 权限标签系统 （或 JWT 令牌范围的约定）控制

Prometheus 和 Grafana 或ELK 堆栈等长期指标存储和可视化服务更适合生产系统。

* 将监控系统与被监控系统分离
* 降低开销
* 长期指标存储
* 访问其他相关指标，例如Erlang 运行时指标
* 更强大和可定制的用户界面
* 易于共享指标数据：指标状态和仪表板
* 指标访问权限并非特定于 RabbitMQ
* 节点特定指标的收集和聚合，对单个节点故障更具弹性

RabbitMQ从 3.8开始为Prometheus 和 Grafana提供一流的支持。推荐用于生产环境。

### 用法

管理插件包含在 RabbitMQ 发行版中。像任何其他插件一样，它必须先启用才能使用。这是使用rabbitmq-plugins完成的：

rabbitmq-plugins enable rabbitmq\_management

可以使用 Web 浏览器在http:// {node-hostname} :15672/访问管理 UI 。管理 UI 被实现为依赖于HTTP API的单页应用程序。其中一些功能包括：

* 声明、列出和删除交换、队列、绑定、用户、虚拟主机和用户权限。
* 监控队列长度、消息速率（全局和每个队列、交换或通道）、队列的资源使用情况、节点 GC 活动、客户端连接的数据速率等。
* 监控节点资源使用：套接字和文件描述符、 内存使用分解、可用磁盘空间和节点间通信链路上的带宽使用。
* 管理用户（提供当前用户的管理权限）。
* 管理策略和运行时参数（提供当前用户的足够权限）。
* 导出模式（虚拟主机、用户、权限、队列、交换、绑定、参数、策略）并将其导入节点 start。这可用于恢复目的 或设置新节点和集群的自动化。
* 强制关闭客户端连接，清除队列。
* 发送和接收消息（在开发环境和故障排除中很有用）。

### 访问和权限

管理 UI 需要身份验证和授权，就像 RabbitMQ 需要连接客户端一样。除了成功的身份验证之外，管理 UI 访问还由用户标签控制。标签是使用rabbitmqctl管理的。默认情况下，新创建的用户没有设置任何标签。

|  |  |
| --- | --- |
| **标签** | **权限** |
| (None) | 无法访问管理插件 |
| management | 用户可以通过消息传递协议执行的任何操作，外加：   * 列出他们可以通过 AMQP 登录的虚拟主机 * 查看“他们的”虚拟主机中的所有队列、交换和绑定 * 查看和关闭自己的频道和连接 * 查看涵盖所有虚拟主机的“全局”统计信息，包括其中其他用户的活动 |
| policymaker | 一切“管理”可以加上：   * 查看、创建和删除可以通过 AMQP 登录的虚拟主机的策略和参数 |
| monitoring | 一切“管理”可以加上：   * 列出所有虚拟主机，包括他们无法使用消息传递协议访问的主机 * 查看其他用户的连接和频道 * 查看节点级数据，例如内存使用和集群 * 查看所有虚拟主机的真正全局统计信息 |
| administrator | 一切“政策制定者”和“监控”都可以加上：   * 创建和删除虚拟主机 * 查看、创建和删除用户 * 查看、创建和删除权限 * 关闭其他用户的连接 |

如果由于缺少具有足够权限的用户或忘记/不正确的权限而无法访问管理 UI，则必须使用CLI 工具来管理用户及其凭据。

* rabbitmqctl add\_user 应该用于创建用户
* rabbitmqctl set\_permissions授予用户所需的权限
* rabbitmqctl set\_user\_tags应该用于授予用户管理 UI 访问权限

### OAuth 2 进行身份验证

RabbitMQ 可以配置为使用JWT 编码的 OAuth 2.0 访问令牌 来验证客户端应用程序和管理 UI 用户。这样做时，管理 UI 不会自动重定向用户以针对 OAuth 2 服务器进行身份验证，这必须单独配置。目前，仅支持UAA授权服务器。要将用户重定向到 UAA 服务器进行身份验证，请使用以下配置：

1. management.enable\_uaa = **true**
2. management.uaa\_client\_id = rabbit\_user\_client
3. management.uaa\_location = https://my-uaa-server-host: 8443 /uaa

使用management.enable\_uaa = true时，仍然可以 针对 HTTP API使用HTTP 基本身份验证进行身份验证。这意味着以下两个访问方法都可以。要切换到仅使用 OAuth 2 进行身份验证以进行管理 UI 访问，请将 management.disable\_basic\_auth配置键设置为true。

### HTTP API

该 API 旨在用于监控和警报目的。它提供对有关节点、连接、通道、队列、消费者等状态的详细信息的访问。可以将 HTTP API 和管理 UI 配置为使用不同的端口或网络接口，启用 HTTPS 等。例如启用 HTTPS 并保留对只能使用 HTTP（没有 TLS）的客户端的支持。可以将管理插件配置为使用 HTTPS。

使用management.tcp.port键配置端口：

1. management.tcp.port = 15672

可以使用management.tcp.ip键配置 API 端点将使用的接口，类似于消息传递协议侦听器：

1. management.tcp.ip = 0.0。0.0

可以使用management.tcp.ip键配置 API 端点将使用的接口，类似于消息传递协议侦听器：

1. # rabbitmq-diagnostics -s listeners
2. Interface: [::], port: 15672, protocol: http, purpose: HTTP API
3. Interface: [::], port: 25672, protocol: clustering, purpose: inter-node and CLI tool communication
4. Interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0

#### HTTPS

可以将管理插件配置为使用 HTTPS。

1. management.ssl.port        = 15671
2. management.ssl.cacertfile = /path/to/ca\_certificate.pem
3. management.ssl.certfile    = /path/to/server\_certificate.pem
4. management.ssl.keyfile     = /path/to/server\_key.pem
5. # # 只有在私钥受密码保护时才能使用此密钥
6. # management.ssl.password = bunnies

可以在不同的端口上同时使用 HTTP 和 HTTPS：

1. management.tcp.port        = 15672
2. management.ssl.port        = 15671
3. management.ssl.cacertfile = /path/to/ca\_certificate.pem
4. management.ssl.certfile    = /path/to/server\_certificate.pem
5. management.ssl.keyfile     = /path /to/server\_key.pem

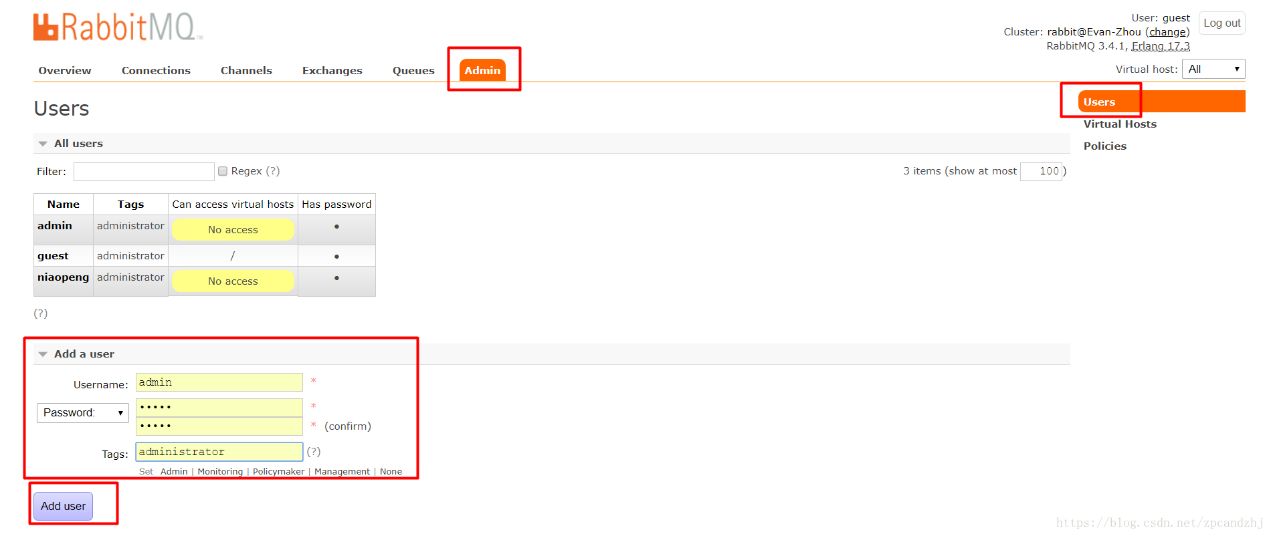
#### 高级 HTTP 选项

管理插件使用的嵌入式 Web 服务器Cowboy提供了许多可用于自定义服务器行为的选项。大多数选项是在 RabbitMQ 3.7.9 中引入的

|  |  |
| --- | --- |
| **配置项** | **介绍** |
| 响应压缩 | 默认情况下启用响应压缩。要显式启用它，请使用management.tcp.compress：  # 对于 RabbitMQ 3.7.9 及更高版本  management.tcp.compress = true |
| 客户端不活动超时 | 一些 HTTP API 端点快速响应，其他可能需要向客户端返回或流式传输相当大的数据集（例如数千个连接）或执行与输入成正比的操作（例如导入大型定义文件）。在这些情况下，处理请求所花费的时间可能会超过 Web 服务器和 HTTP 客户端中的某些超时。   * management.tcp.inactivity\_timeout控制 HTTP(S) 客户端的 TCP 连接不活动超时。到达时，HTTP 服务器将关闭连接。 * management.tcp.request\_timeout控制客户端必须发送 HTTP 请求的时间窗口。 * management.tcp.idle\_timeout控制客户端必须在 HTTP 请求的上下文中发送更多数据（如果有）的时间窗口。   如果在 HTTP 客户端和管理 HTTP 服务器之间使用负载平衡器或代理，则inactivity\_timeout和idle\_timeout值应至少与负载平衡器使用的超时和不活动值一样大，并且通常更大。 |
| HTTP 请求记录 | 要创建对 HTTP API 的请求的简单访问日志，请将management.http\_log\_dir键的值设置为可以创建日志的目录的路径：  management.http\_log\_dir = /path/to/folder |
| 统计区间 | 默认情况下，服务器将每 5 秒（5000毫秒）发出一次统计事件。管理插件中显示的消息速率值是在此期间计算的。增加此值将减少在具有大量统计信息发送实体（如连接、通道、队列）的环境中统计信息收集的 CPU 资源消耗。为此，请将collect\_statistics\_interval配置键的值设置为所需的间隔（以毫秒为单位）并重新启动节点：  collect\_statistics\_interval = 15000 |
| 消息率 | 管理插件默认显示全局消息速率，以及每个队列、通道、交换和虚拟主机。这些被称为基本消息速率。它还可以显示通道到交换、交换到队列和队列到通道的所有组合的消息速率。这些被称为详细消息率。默认情况下禁用详细消息速率，因为当存在大量通道、队列和交换的组合时，它们可能占用大量内存。或者，可以完全禁用消息速率。这可以帮助减少插件的 CPU 资源消耗。消息速率模式由 management.rates\_mode配置键控制：  # 支持的值：basic、basic、none  management.rates\_mode = basic |
| 样本（数据点）保留 | 管理插件将保留一些数据的样本，例如消息速率和队列长度。根据数据的保留时间，UI 图表上的某些时间范围选项可能不完整或不可用。策略有以下三种：  global：为概述和虚拟主机页面保留数据多长时间  basic : 为单个连接、通道、交换和队列保留数据多长时间  detailed：在成对的连接、通道、交换和队列之间保留多长时间的消息速率数据（如“消息速率细分”下所示）  所有三个策略都是强制性的，并且必须包含至少一个保留设置（期限）。 |
| 禁用统计信息和指标收集 | 可以禁用 UI 和HTTP API中的统计信息，以便它们仅用于操作。如果正在使用Prometheus 和 Grafana等外部监控解决方案，这可能是一个有用的功能。如果通过以下任一方式禁用统计信息，则所有图表和详细统计信息都将隐藏在 UI 中。  management\_agent.disable\_metrics\_collector = true |
| 内容安全策略 (CSP) | 可以配置HTTP API 响应使用的CSP 标头值。默认值为script-src 'self' 'unsafe-eval' 'unsafe-inline'; object-src 'self'：  management.csp.policy = script-src 'self' 'unsafe-eval' 'unsafe-inline'; object-src 'self'  该值可以是任何有效的 CSP 标头字符串：  management.csp.policy = default-src <https://rabbitmq.eng.example.local>  也允许使用通配符：  management.csp.policy = default-src 'self' \*.eng.example.local  CSP 策略frame-ancestors指令可用于防止管理 UI 的框架嵌入，从而减轻某些类型的跨框架脚本攻击：  # 禁止 UI 的 iframe 嵌入  management.csp.policy = frame-ancestors 'none' |
| 严格的运输安全 (HSTS) | 可以配置HTTP API 响应使用的Strict Transport Security 标头值：  management.hsts.policy = max-age= 31536000 ; includeSubDomains |
| 跨域资源共享 (CORS) | 默认情况下，管理 UI 应用程序将使用跨域资源共享机制（也称为 CORS）拒绝访问托管在与其自身不同的源上的网站。可以将来源列入白名单：  management.cors.allow\_origins.1 = https://origin1.org  management.cors.allow\_origins.2 = https://origin2.org  可以允许任何来源使用通配符使用 API。对于可能向公众公开 UI 应用程序的部署，强烈建议不要这样做。  management.cors.allow\_origins.1 = \*  CORS 飞行前请求由浏览器缓存。管理插件默认定义了 30 分钟的超时时间。该值可以更改。它在几秒钟内配置：  management.cors.allow\_origins.1 = https://origin1.org  management.cors.allow\_origins.2 = https://origin2.org  management.cors.max\_age = 3600 |
| 登录会话超时 | 用户登录后，默认情况下，她的 Web UI 登录会话将在 8 小时后过期。可以使用 login\_session\_timeout设置配置不同的超时。  该值应该是一个整数：它以分钟为单位控制登录会话的长度。当时间到时，用户将被注销。  management.login\_session\_timeout = 60 |
| 路径前缀 | 某些环境要求对管理插件的所有 HTTP 请求使用自定义前缀。management.path\_prefix设置允许为管理插件中的 所有 HTTP 请求处理程序设置任意前缀。将management.path\_prefix设置为/my-prefix 指定所有 API 请求使用 URI host:port/my-prefix/api/[...]管理 UI 登录页面将具有 URI host:port/my-prefix/ - 请注意，在这种情况下需要尾部斜杠。  management.path\_prefix = / my-prefix |

### 界面功能演示

#### 添加admin用户



1、超级管理员(administrator)

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

2、监控者(monitoring)

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

3、策略制定者(policymaker)

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

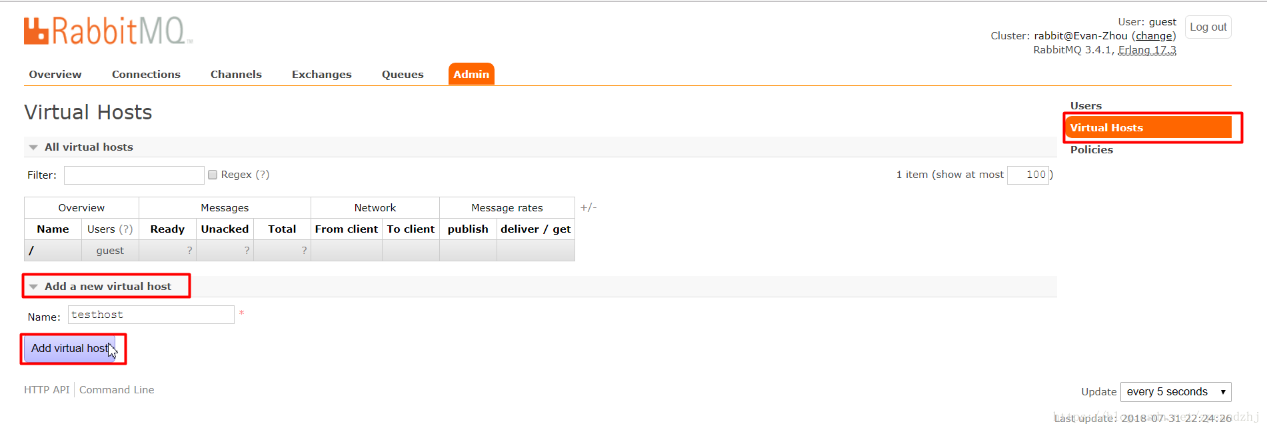
4、普通管理者(management)

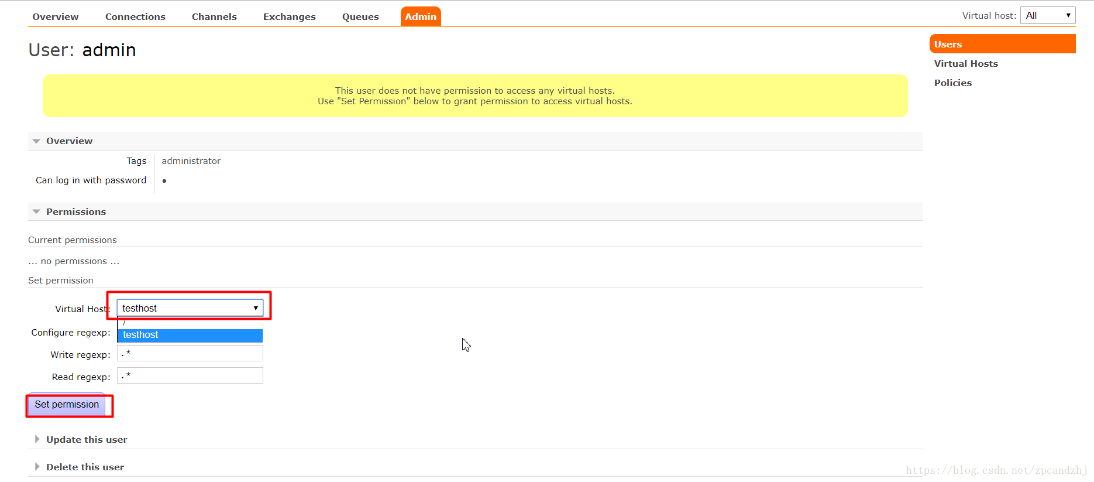
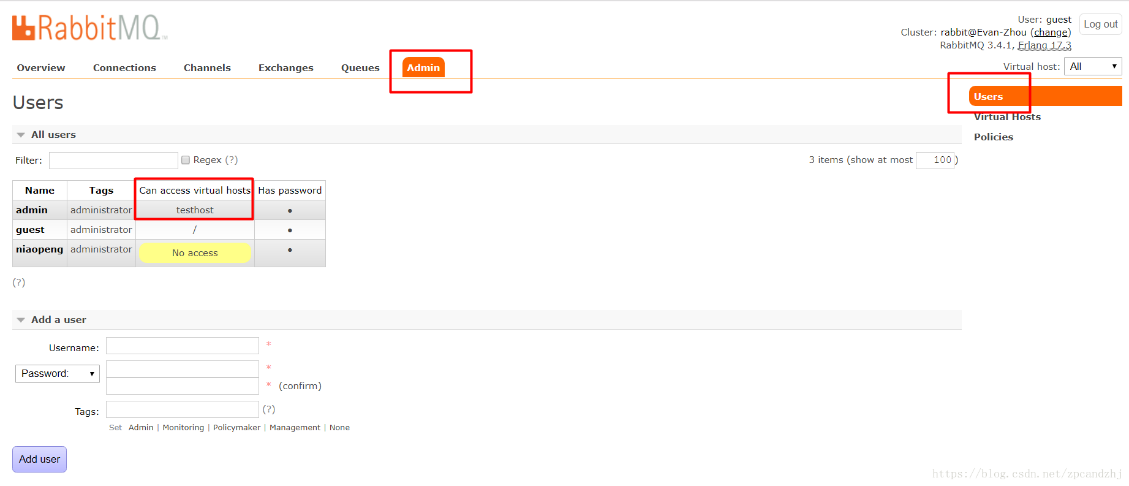
仅可登陆管理控制台，无法看到节点信息，也无法对策略进行管理。

5、其他

无法登陆管理控制台，通常就是普通的生产者和消费者。

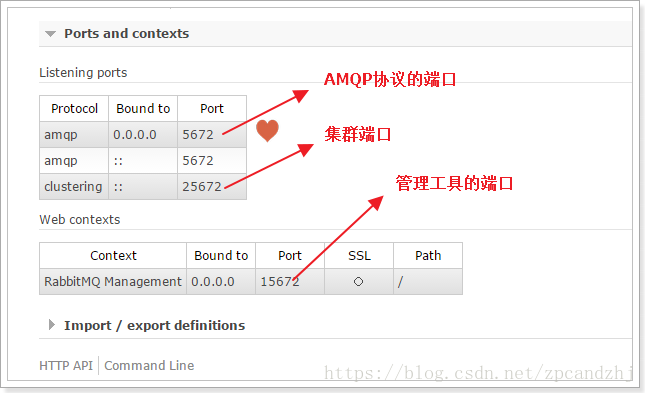
#### 创建Virtual Hosts



选中Admin用户，设置权限：  
  
看到权限已加：  


#### 管理界面中的功能





## 监控

监控系统通常与警报系统集成。当监控系统检测到异常时，通常会将某种警报传递给警报系统，该系统会通知相关方，例如技术运营团队。监控定义为通过运行状况检查和指标随时间推移捕获系统行为的过程。这有助于检测异常情况：当系统不可用、遇到异常负载、耗尽某些资源或以其他方式不在其正常（预期）参数范围内运行时。监控涉及长期收集和存储指标，这不仅对异常检测很重要，而且对根本原因分析、趋势检测和容量规划也很重要。

许多监控系统会定期轮询其监控的服务。完成的频率因工具而异，但通常可以由操作员配置。非常频繁的轮询会对受监控的系统产生负面影响。建议的指标收集间隔为 15 秒。要以更接近实时的间隔收集，请使用 5 秒 - 但不能更低。对于速率指标，请使用跨越 4 个指标收集间隔的时间范围，以便它可以容忍竞争条件并具有弹性以应对抓取失败。对于生产系统，建议收集间隔为 30 甚至 60 秒。 Prometheus导出器 API 设计为每 15 秒抓取一次，包括生产系统。

迈向有用的监控系统的第一步始于基础设施和内核指标。其中有很多，但有些比其他更重要。在运行 RabbitMQ 节点或应用程序的所有主机上收集以下指标：

* CPU 统计信息（用户、系统、io等待和空闲百分比）
* 内存使用（已使用、缓冲、缓存和空闲百分比）
* 虚拟内存统计信息（脏页刷新、写回卷）
* 磁盘 I/O（操作和每单位时间传输的数据量，服务操作时间）
* 挂载上用于节点数据目录的可用磁盘空间
* beam.smp使用的文件描述符与最大系统限制
* 按状态（ ESTABLISHED、CLOSE\_WAIT、TIME\_WAIT）的 TCP 连接
* 网络吞吐量（接收的字节数，发送的字节数）和最大网络吞吐量
* 网络延迟（集群中所有 RabbitMQ 节点之间以及客户端之间）

### [集群监控](https://www.rabbitmq.com/monitoring.html" \l "clusters)

在监控集群时，了解 HTTP API 提供的保证非常重要。在集群环境中，每个节点都可以为度量端点请求提供服务。可以从任何可以联系其对等节点的节点获取集群范围的指标。在产生响应之前，该节点将根据需要从其对等节点收集和组合数据。每个节点还可以向端点提供请求，这些端点 为其自身以及其他集群节点提供特定于节点的指标。与基础设施和操作系统指标一样，必须为每个节点收集特定于节点的指标。监控工具可以针对任何节点执行 HTTP API 请求。

GET /api/overview是返回集群范围指标的HTTP API端点。

| 名称 | JSON 字段名称 |
| --- | --- |
| 集群名称 | 集群名称 |
| 集群范围的消息速率 | message\_stats |
| 连接总数 | object\_totals.connections |
| 通道总数 | object\_totals.channels |
| 队列总数 | object\_totals.queues |
| 消费者总数 | object\_totals.consumers |
| 消息总数（就绪加未确认） | queue\_totals.messages |
| 准备发送的消息数 | queue\_totals.messages\_ready |
| [未确认](https://www.rabbitmq.com/confirms.html)消息的数量 | queue\_totals.messages\_unacknowledged |
| 最近发布的消息 | message\_stats.publish |
| 消息发布率 | message\_stats.publish\_details.rate |
| 最近发送给消费者的消息 | message\_stats.deliver\_get |
| 消息传递率 | message\_stats.deliver\_get\_details.rate |
| 其他消息统计 | message\_stats.\* |

### 节点指标

有两个 HTTP API端点提供对特定于节点的指标的访问：

* GET /api/nodes/{node}返回单个节点的统计信息
* GET /api/nodes返回所有集群成员的统计信息

后一个端点返回一个对象数组。支持（或可以支持）作为输入的监控工具应该更喜欢该端点，因为它减少了请求的数量。如果不是这种情况，请使用前一个端点依次检索每个集群成员的统计信息。这意味着监控系统知道集群成员的列表。

大多数指标代表时间点绝对值。有些代表最近一段时间的活动（例如，GC 运行和回收的字节）。与之前的值和历史平均值/百分比值相比，后一种指标最有用。

| 名称 | JSON 字段名称 |
| --- | --- |
| 使用的内存总量 | mem\_used |
| 内存使用高水位线 | mem\_limit |
| 内存警报是否有效？ | mem\_alarm |
| 可用磁盘空间低水位线 | disk\_free\_limit |
| 磁盘警报是否生效？ | disk\_free\_alarm |
| 可用的文件描述符 | fd\_total |
| 使用的文件描述符 | fd\_used |
| 文件描述符打开尝试 | io\_file\_handle\_open\_attempt\_count |
| 可用的插座 | sockets\_total |
| 使用的插座 | sockets\_used |
| 消息存储磁盘读取 | message\_stats.disk\_reads |
| 消息存储磁盘写入 | message\_stats.disk\_writes |
| 节点间通信链路 | cluster\_links |
| GC 运行 | gc\_num |
| GC 回收的字节数 | gc\_bytes\_reclaimed |
| Erlang 进程限制 | proc\_total |
| 使用的 Erlang 进程 | proc\_used |
| 运行时运行队列 | run\_queue |

### 单个队列指标

各个队列指标可通过HTTP API 通过GET /api/queues/{vhost}/{qname}端点获得。

| 名称 | JSON 字段名称 |
| --- | --- |
| 记忆 | memory |
| 消息总数（就绪加未确认） | messages |
| 准备发送的消息数 | messages\_ready |
| 未确认消息的数量 | messages\_unacknowledged |
| 最近发布的消息 | message\_stats.publish |
| 消息发布率 | message\_stats.publish\_details.rate |
| 最近发送的消息 | message\_stats.deliver\_get |
| 消息传递率 | message\_stats.deliver\_get.rate |
| 其他消息统计 | message\_stats.\* |

### 应用程序级指标

使用消息传递的系统几乎总是分布式的。在这样的系统中，哪个组件行为不端通常不是很明显。系统的每一个部分，包括应用程序，都应该受到监控和调查。例如，很容易判断一个节点的磁盘空间不足，但并不总是很容易判断为什么。一些客户端库和框架提供了注册指标收集器或开箱即用地收集指标的方法。 RabbitMQ Java 客户端和Spring AMQP就是两个例子。与其他人一样，开发人员必须在他们的应用程序代码中跟踪指标：

* 连接打开率
* 渠道开通率
* 连接失败（恢复）率
* 发表率
* 交货率
* 正送达确认率
* 负送达确认率
* 平均/第95百分位传递处理延迟

### 健康检查

健康检查是测试 RabbitMQ 服务的某个方面是否按预期运行的命令。健康检查由机器定期执行或由操作员交互执行。运行状况检查可用于评估节点的状态和活跃度，也可用作部署自动化和编排工具（包括升级期间）的就绪探测 。可以执行一系列健康检查，从最基本且极少产生误报的开始，到越来越全面、侵入性和固执己见的误报概率更高的检查。换句话说，健康检查越全面，结果就越不确定。健康检查可以验证单个节点（节点健康检查）或整个集群（集群健康检查）的状态。

虽然有健康检查，但数字越大并不意味着检查“更好”。健康检查可以有选择地使用和组合使用。除非另有说明，否则检查应遵循与度量收集相同的监控频率建议。RabbitMQ 的早期版本使用了侵入式健康检查，该检查已被弃用，应避免使用。使用本节中介绍的一项检查（或它们的组合）。

rabbitmq-diagnostics ping执行此检查：

rabbitmq-diagnostics -q ping

# => 如果退出代码为 0，则 Ping 成功

更全面的检查是执行rabbitmq-diagnostics status status，这包括第 1 阶段检查以及检索一些对其他检查有用的基本系统信息。

包括以前的检查并验证 RabbitMQ 应用程序是否正在运行，并且没有资源警报。

# 列出集群中有效的警报（如果有的话）

rabbitmq-diagnostics -q alarms

rabbitmq-diagnostics check\_running是一项检查，确保运行时正在运行并且其上的 RabbitMQ 应用程序没有停止或暂停。rabbitmq-diagnostics check\_local\_alarms检查节点上是否存在有效的本地警报。如果有，它将以非零状态退出。

rabbitmq-diagnostics -q check\_running && rabbitmq-diagnostics -q check\_local\_alarms

# 如果两个检查都成功，退出代码将为 0

包括以上中的所有检查以及对所有启用的侦听器的检查（使用临时 TCP 连接）。要检查节点上启用的所有侦听器，请使用rabbitmq-diagnostics listeners：

rabbitmq-diagnostics -q listeners

# => 接口：[::]，端口：25672，协议：集群，目的：节点间和 CLI 工具通信

# => 接口：[::]，端口：5672，协议：amqp ，用途：AMQP 0-9-1 和 AMQP 1.0

# => 接口：[::]，端口：5671，协议：amqp/ssl，用途：AMQP 0-9-1 和 AMQP 1.0 over TLS

# => 接口： [::]，端口：15672，协议：http，目的：HTTP API

# => 接口：[::]，端口：15671，协议：https，目的：基于 TLS 的 HTTP API (HTTPS)

rabbitmq-diagnostics check\_port\_connectivity是一个执行上述基本 TCP 连接检查的命令：

rabbitmq-diagnostics -q check\_port\_connectivity

#如果检查成功，退出码为0

包括以上的所有检查以及是否存在失败的虚拟主机的检查。rabbitmq-diagnostics check\_virtual\_hosts是一个检查是否有任何虚拟主机依赖项可能失败的命令。这是为所有虚拟主机完成的。

rabbitmq-diagnostics -q check\_virtual\_hosts

#如果检查成功，退出码为0

### 监控工具

收集 RabbitMQ 指标的第三方工具列表。这些工具的功能各不相同，但通常可以收集基础设施级别和 RabbitMQ 指标。此列表绝不是完整的。

| 监控工具 | 地址 |
| --- | --- |
| AppDynamics | [AppDynamics](https://www.appdynamics.com/community/exchange/extension/rabbitmq-monitoring-extension/), [GitHub](https://github.com/Appdynamics/rabbitmq-monitoring-extension) |
| AWS CloudWatch | [GitHub](https://github.com/noxdafox/rabbitmq-cloudwatch-exporter) |
| collectd | [GitHub](https://github.com/signalfx/integrations/tree/master/collectd-rabbitmq) |
| DataDog | [DataDog RabbitMQ integration](https://docs.datadoghq.com/integrations/rabbitmq/), [GitHub](https://github.com/DataDog/integrations-core/tree/master/rabbitmq) |
| Dynatrace | [Dynatrace RabbitMQ monitoring](https://www.dynatrace.com/technologies/rabbitmq-monitoring/) |
| Ganglia | [GitHub](https://github.com/ganglia/gmond_python_modules/tree/master/rabbit) |
| Graphite | [Tools that work with Graphite](http://graphite.readthedocs.io/en/latest/tools.html) |
| Munin | [Munin docs](http://munin-monitoring.org/), [GitHub](https://github.com/ask/rabbitmq-munin) |
| Nagios | [GitHub](https://github.com/nagios-plugins-rabbitmq/nagios-plugins-rabbitmq) |
| Nastel AutoPilot | [Nastel RabbitMQ Solutions](https://www.nastel.com/rabbitmq/) |
| New Relic | [NewRelic Plugins](https://newrelic.com/plugins/vmware-29/95), [GitHub](https://github.com/pivotalsoftware/newrelic_pivotal_agent) |
| Prometheus | [Prometheus guide](https://www.rabbitmq.com/prometheus.html), [GitHub](https://github.com/rabbitmq/rabbitmq-prometheus) |
| Sematext | [Sematext RabbitMQ monitoring integration](https://sematext.com/docs/integration/rabbitmq/), [Sematext RabbitMQ logs integration](https://sematext.com/docs/integration/rabbitmq-logs/) |
| Zabbix | [Zabbix by HTTP](https://git.zabbix.com/projects/ZBX/repos/zabbix/browse/templates/app/rabbitmq_http), [Zabbix by Agent](https://git.zabbix.com/projects/ZBX/repos/zabbix/browse/templates/app/rabbitmq_agent), [Blog article](http://blog.thomasvandoren.com/monitoring-rabbitmq-queues-with-zabbix.html) |
| Zenoss | [RabbitMQ ZenPack](https://www.zenoss.com/product/zenpacks/rabbitmq), [Instructional Video](http://www.youtube.com/watch?v=CAak2ayFcV0) |

## TLS 支持

RabbitMQ 内置了对 TLS 的支持。这包括客户端连接和流行的插件（如果适用），例如联合链接。也可以使用 TLS加密集群中的节点间连接。

* 使用 TLS 进行客户端连接的两种方式：直接或通过 TLS 终止代理
* TLS 支持的Erlang/OTP 要求
* 在 RabbitMQ 中启用 TLS
* 如何使用 tls-gen或手动为开发和 QA 环境生成自签名证书
* Java和.NET客户端中的 TLS 配置
* 客户端连接或相互（“mTLS”）的对等（证书链）验证
* 与 RabbitMQ 相关的公钥使用扩展
* 如何控制启用的TLS 版本和密码套件
* TLSv1.3支持
* 可用于评估 TLS 设置的工具
* 对 TLS 的已知攻击及其缓解措施
* 如何使用私钥密码

可以为 RabbitMQ 支持的所有协议启用 TLS，而不仅仅是本指南重点介绍的 AMQP 0-9-1。HTTP API、节点间和 CLI 工具流量也可以配置为使用 TLS (HTTPS)。对于客户端连接，有两种常见的方法：

* 配置 RabbitMQ 以处理 TLS 连接
* 使用代理或负载均衡器（例如HAproxy）执行客户端连接的TLS 终止，并使用普通 TCP 连接到 RabbitMQ 节点。

### TLS 支持的 Erlang/OTP 要求

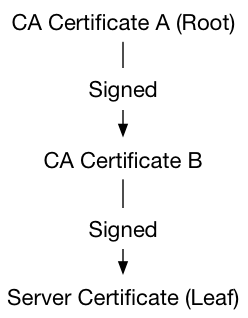
为了支持 TLS 连接，RabbitMQ 需要在 Erlang/OTP 安装中提供 TLS 和加密相关模块。Erlang asn1、crypto、 public\_key和ssl库（应用程序）必须已安装且正常运行。在 Debian 和 Ubuntu 上，这分别由erlang-asn1、 erlang-crypto、erlang-public-key和 erlang-ssl包提供。RabbitMQ的零依赖 Erlang RPM包括上述模块。如果预计使用椭圆曲线加密 (ECC) 密码套件， 强烈建议使用最近受支持的 Erlang 版本。早期版本在 ECC 支持方面存在已知限制。如果您遇到上述限制或任何其他不兼容性，请使用 TLS 终止选项

### TLS 基础

TLS 有两个主要目的：加密连接流量并提供一种对对等方进行身份验证（验证）以减轻中间人攻击的方法。两者都是使用一组称为公钥基础设施(PKI) 的角色、策略和程序来完成的。

PKI 基于可以加密（数学）验证的数字身份的概念。这些身份称为 证书或更准确地说，证书/密钥对。每个启用 TLS 的服务器通常都有自己的证书/密钥对，用于计算特定于连接的密钥，该密钥将用于加密在连接上发送的流量。此外，如果被询问，它可以将其证书（公钥）提供给连接对等方。客户可能有也可能没有自己的证书。在消息传递和 RabbitMQ 等工具的上下文中，客户端也很常见使用证书/密钥对，以便服务器可以验证其身份。

证书/密钥对由 OpenSSL 等工具生成，并由称为证书颁发机构(CA) 的实体签名。CA 颁发用户（应用程序或其他 CA）使用的证书。当证书由 CA 签名时，它们会形成信任链。此类链可以包含多个 CA，但最终会签署应用程序使用的证书/密钥对（叶证书或最终用户证书）。CA 证书链通常一起分布在一个文件中。这样的文件称为CA 包。这是具有一个根 CA 和一个叶（服务器或客户端）证书的最基本链的示例：



启用 TLS 的 RabbitMQ 节点必须在文件（CA 包）、证书（公钥）文件和私钥文件中拥有一组它认为受信任的证书颁发机构证书。这些文件将从本地文件系统中读取。RabbitMQ 节点进程的有效用户必须能够读取它们。

### tls-gen便捷生成证书/密匙对

在 MacOS 或 Linux 上生成所有这些东西的一种更简单的方法是使用tls-gen：它需要Python 3.5+，make和openssl在PATH。tls-gen支持用于密钥生成的 RSA 和椭圆曲线加密算法。下面是一个生成 CA 并使用它生成两个证书/密钥对的示例，一个用于服务器，另一个用于客户端。

1. git clone https://github.com/michaelklishin/tls-gen tls-gen
2. cd tls-gen/basic
3. # private key password
4. make PASSWORD=bunnies
5. make verify
6. make info
7. ls -l ./result

### RabbitMQ 中启用 TLS 支持

要在 RabbitMQ 中启用 TLS 支持，必须将节点配置为知道证书颁发机构捆绑包（具有多个 CA 证书的文件）的位置、服务器的证书文件和服务器的密钥。还应启用 TLS 侦听器以了解要侦听启用 TLS 的客户端连接的端口。

| 配置密钥 | 描述 |
| --- | --- |
| listeners.ssl | 用于侦听 TLS 连接的端口列表。RabbitMQ 可以侦听单个接口或多个接口。 |
| ssl\_options.cacert 文件 | 证书颁发机构 (CA) 捆绑文件路径 |
| ssl\_options.certfile | 服务器证书文件路径 |
| ssl\_options.keyfile | 服务器私钥文件路径 |
| ssl\_options.verify | 是否应该启用对等验证？ |
| ssl\_options.fail\_if\_no\_peer\_cert | 设置为true时，如果客户端未能提供证书，TLS 连接将被拒绝 |

这些选项在配置文件中提供。下面是一个配置文件的示例，它将在此主机名上的所有接口上的端口 5671 上启动一个 TLS 侦听器：

1. listeners.ssl.default=5671
2. ssl\_options.cacertfile=/path/to/ca\_certificate.pem
3. ssl\_options.certfile=/path/to/server\_certificate.pem
4. ssl\_options.keyfile=/path/to/server\_key.pem
5. ssl\_options.verify=verify\_peer
6. ssl\_options.fail\_if\_no\_peer\_cert=true

此配置还将执行对等证书链验证 ，因此没有任何证书的客户端将被拒绝。

可以完全禁用常规（非 TLS）侦听器。只有启用了 TLS 的客户端才能连接到这样的节点，并且只有当它们使用正确的端口时：

1. # 禁用非 TLS 侦听器，只有启用 TLS 的客户端才能连接
2. listeners.tcp = none
3. listeners.ssl.default = 5671
4. ssl\_options.cacertfile = /path/to/ca\_certificate.pem
5. ssl\_options.certfile    = /path/to/ server\_certificate.pem
6. ssl\_options.keyfile     = /path/to/server\_key.pem
7. ssl\_options.verify      = verify\_peer
8. ssl\_options.fail\_if\_no\_peer\_cert = true

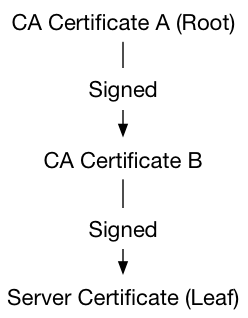
私钥可以选择受密码保护。要提供密码，请使用密码选项：

1. listeners.ssl.1 = 5671
2. ssl\_options.cacertfile = /path/to/ca\_certificate.pem
3. ssl\_options.certfile    = /path/to/server\_certificate.pem
4. ssl\_options.keyfile     = /path/to/server\_key.pem
5. ssl\_options.password    = t0p $3kRe7

### TLS 对等验证

当建立TLS 连接时，客户端和服务器执行需要几个步骤的连接协商。第一步是当对等方可选地交换他们的证书时。交换证书后，对等方可以选择尝试在其 CA 证书和提供的证书之间建立信任链。这用于验证对等方是它声称的身份（假设私钥没有被盗）。该过程称为对等验证或对等验证，并遵循称为认证路径验证算法的算法。

每个对等点都提供一个证书链，该证书链以“叶”（客户端或服务器）证书开头，并以至少一个证书颁发机构 (CA) 证书继续。该 CA 颁发（签名）叶子 CA。如果有多个 CA 证书，它们通常会形成一个签名链，这意味着每个 CA 证书都由下一个 CA 证书签名。例如，如果证书 B 由 A 签名，C 由 B 签名，则链为A、B、C。“最顶层”（第一个或唯一的）CA 通常被称为链的根 CA。根 CA 可以由知名证书颁发机构（商业供应商）或任何其他方（自签名）颁发。



在对等验证期间，TLS 连接客户端（或服务器）遍历对等提供的证书链，如果找到受信任的证书，则认为对等是受信任的。

每个启用 TLS 的工具和 TLS 实现，包括 Erlang/OTP 和 RabbitMQ，都有一种将一组证书标记为可信的方法。对此有三种常见的方法：

* 所有受信任的 CA 证书都必须添加到称为CA 证书包的单个文件中
* 目录中的所有 CA 证书都被认为是可信的
* 使用专用工具来管理受信任的 CA 证书

不同的 TLS 实现和工具使用不同的选项。在 RabbitMQ 的上下文中，这意味着受信任的证书管理方法对于不同的客户端库、工具和 RabbitMQ 服务器本身可能会有所不同。

RabbitMQ 依赖于 Erlang 的 TLS 实现。它假定 所有受信任的 CA 证书都添加到服务器证书包中。在执行对等验证时，RabbitMQ 只会认为根证书（列表中的第一个证书）是可信的。任何中间证书都将被忽略。如果希望中间证书也被认为是受信任的，则必须将它们添加到受信任的证书列表：证书包。虽然可以将最终（“叶”）证书（例如服务器和客户端使用的证书）放置到受信任的证书目录中，但更常见的做法是将 CA 证书添加到受信任的证书列表中。将多个证书相互附加并在单个证书颁发机构捆绑文件中使用的最常见方法是简单地连接它们：

1. cat rootca/ca\_certificate.pem otherca/ca\_certificate.pem > all\_cacerts.pem

### [证书链和验证深度](https://www.rabbitmq.com/ssl.html" \l "peer-verification-depth)

当使用由中间 CA 签名的客户端证书时，可能需要配置 RabbitMQ 服务器以使用更高的验证深度。深度是可以在有效证书路径中跟随对等证书的非自发行中间证书的最大数量。因此，如果深度为 0，则对等（例如客户端）证书必须由受信任的 CA 直接签名，如果为 1，则路径可以是“对等，CA，受信任的 CA”，如果是 2，则为“对等，CA，CA，受信任的 CA” ， 等等。默认深度为 1。RabbitMQ 服务器配置证书验证深度：

1. listeners.ssl.default = 5671
2. ssl\_options.cacertfile = /path/to/ca\_certificate.pem
3. ssl\_options.certfile = /path/to/server\_certificate.pem
4. ssl\_options.keyfile = /path/to/server\_key.pem
5. ssl\_options.verify = verify\_peer
6. ssl\_options.depth = 2
7. ssl\_options.fail\_if\_no\_peer\_cert = false

### [Java 客户端中使用 TLS](https://www.rabbitmq.com/ssl.html" \l "java-client)

在 RabbitMQ Java 客户端中启用 TLS 有两个主要部分：使用一些 Java 安全框架管道设置密钥存储和实现所需的对等验证策略。Java 安全框架中包含三个主要组件：Key Manager、 Trust Manager和Key Store。

对等点（在本例中为客户端连接）使用密钥管理器来管理其证书。在 TLS 连接/会话协商期间，密钥管理器将控制向远程对等方发送哪些证书。对等点使用信任管理器来管理远程证书。在 TLS 连接/会话协商期间，信任管理器将控制远程对等方信任哪些证书。信任管理器可用于实现任何证书链验证逻辑。密钥存储是证书存储概念的 Java 封装。所有证书必须存储为特定于 Java 的二进制格式 (JKS) 或 PKCS#12 格式。这些格式使用 KeyStore类进行管理。

#### 与 TLS 连接

示例将显示一个简单的客户端通过 TLS 连接到 RabbitMQ 服务器，而无需验证服务器证书，也无需向服务器提供任何客户端证书。

1. ConnectionFactory factory = new ConnectionFactory();
2. factory.setHost("localhost");
3. factory.setPort(5671);
4. factory.useSslProtocol();
6. // 告诉库为您设置默认的密钥和信任管理器，不做任何形式的远程服务器信任验证
7. Connection conn = factory.newConnection();
8. Channel channel = conn.createChannel();
10. // 非持久、独占、自动删除队列
11. channel.queueDeclare("rabbitmq-java-test", false, true, true, null);
12. channel.basicPublish("", "rabbitmq-java-test", null, "Hello, World".getBytes());
13. GetResponse chResponse = channel.basicGet("rabbitmq-java-test", false);
14. if (chResponse == null) {
15. System.out.println("No message retrieved");
16. } else {
17. byte[] body = chResponse.getBody();
18. System.out.println("Received: " + new String(body));
19. }
20. channel.close();
21. conn.close();

#### 连接 TLS 设置

启用 TLS 的连接，我们需要在 ConnectionFactory 的参数字段中设置一些新字段。

| 属性 | 描述 |
| --- | --- |
| Ssl.CertPath | 如果您的服务器需要客户端验证，这是 PKCS#12 格式的客户端证书的路径。这是可选的。 |
| Ssl.CertPassphrase | 如果您使用的是 PKCS#12 格式的客户端证书，那么它可能会有一个密码，您在此字段中指定该密码。 |
| Ssl.Enabled | 这是一个布尔字段，用于打开或关闭 TLS 支持。默认情况下它是关闭的。 |
| Ssl.ServerName | .NET 期望这与服务器发送过来的证书上的主题备用名称 (SAN) 或公用名称 (CN) 匹配。 |

### 公钥使用选项

公钥（证书）有许多描述密钥预期使用场景的字段。这些字段限制了各种工具如何允许使用密钥。例如，公钥可用于验证证书签名（充当证书颁发机构密钥）。这些字段还会影响RabbitMQ 节点和客户端在连接协商期间（更具体地说，TLS 握手）将使用哪些密码套件，因此解释这些影响是很重要的。本指南将通过一些有意的过度简化来涵盖它们。从广义上讲，这些领域属于以下三类之一：

* 关键用法
* 基本约束
* 扩展密钥使用

一些字段是布尔值，其他字段是不同类型的，例如可以设置或取消设置的一组选项（位）。数据服务在很大程度上不受所使用的约束和关键使用选项的影响。但是，有些对于本指南中描述的用例是必不可少的：

* 服务器身份验证（向客户端提供服务器节点的身份）
* 客户端身份验证（向服务器提供客户端身份）
* 验证数字签名
* 密钥加密

前两个选项用于对等验证。它们必须在公钥生成时分别为服务器和客户端证书设置。证书可以同时设置两个选项。tls-gen将确保正确设置这些约束和扩展。手动生成证书时，这是生成密钥对的操作员或密钥对提供者的责任。

### 密码套件

两个关键扩展对于两种主要类型的密码套件至关重要：

* 基于 ECC（椭圆曲线加密）的套件的数字签名
* 基于RSA的套件的密钥加密

强烈建议为 RabbitMQ 节点和客户端库都将使用的证书设置上述两个选项（位）。如果未设置这些位，则 TLS 实现将忽略一整类密码套件，从而可能导致在连接时混淆“未找到合适的密码套件”警报（错误消息）。

要查看为公钥设置了哪些约束和扩展，请使用openssl x509命令：

openssl x509 -in /home/result/ca\_certificate.pem -text -noout

它的输出将包括一个嵌套的扩展和约束列表，看起来类似于：

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

X509v3 Key Usage:

Digital Signature, Key Encipherment

X509v3 Extended Key Usage:

TLS Web Client Authentication

上述扩展集表示这是一个公钥，可用于验证客户端（向 RabbitMQ 节点提供客户端身份），不能用作证书颁发机构证书，可用于密钥加密和数字签名。

可以配置 RabbitMQ 将使用哪些密码套件。请注意，并非所有套件都可在所有系统上使用。例如，要使用椭圆曲线密码，必须使用最近支持的 Erlang 版本。要列出运行节点的 Erlang 运行时支持的密码套件，请使用

rabbitmq-diagnostics cipher\_suites --format openssl

这将生成 OpenSSL 格式的密码套件列表。请注意，如果您使用--format erlang：

rabbitmq-diagnostics cipher\_suites --format erlang -q

然后rabbitmq-diagnostics cipher\_suites将以仅在经典配置格式中接受的格式列出密码套件。两种配置格式都接受 OpenSSL 格式。请注意，密码套件未在新样式配置格式中引用，但在经典格式中需要双引号。

1. listeners.ssl.1 = 5671
2. ssl\_options.cacertfile = /path/to/ca\_certificate.pem
3. ssl\_options.certfile = /path/to/server\_certificate.pem
4. ssl\_options.keyfile = /path/to/server\_key.pem
5. ssl\_options.versions.1 = tlsv1.2
6. # these MUST be disabled if TLSv1.3 is used
7. ssl\_options.honor\_cipher\_order = true
8. ssl\_options.honor\_ecc\_order     = true

在 TLS 连接协商期间，服务器和客户端协商将使用什么密码套件。可以强制服务器的 TLS 实现指定其偏好（密码套件顺序），以避免恶意客户端故意协商弱密码套件以准备对其进行攻击。为此，请将 Honor\_cipher\_order 和Honor\_ecc\_order配置为true：

### TLS 漏洞及其缓解措施

#### The ROBOT Attack

ROBOT 是一个存在 19 年之久的漏洞的回归，它允许使用 TLS 服务器的私钥执行 RSA 解密和签名操作。1998 年，Daniel Bleichenbacher 发现 SSL 服务器针对 PKCS #1 v1.5 填充中的错误提供的错误消息允许自适应选择密文攻击；当与 RSA 加密一起使用时，这种攻击完全破坏了 TLS 的机密性。通过使用一些细微的变化，这个漏洞仍然可以用于攻击当今 Internet 中的许多 HTTPS 主机。

对于易受攻击且仅支持 RSA 加密密钥交换的主机，这非常糟糕。这意味着攻击者可以被动地记录流量，然后再对其进行解密。对于通常使用前向保密但仍支持易受攻击的 RSA 加密密钥交换的主机，风险取决于攻击者执行攻击的速度。我们相信冒充服务器或中间人攻击是可能的，但更具挑战性。

ROBOT 攻击影响依赖 RSA 密码套件并在 19.3.6.4 和 20.1.7 之前的 Erlang/OTP 版本上运行的 RabbitMQ 安装。为了缓解，将 Erlang/OTP 升级到补丁版本并考虑限制支持的密码套件列表。

#### POODLE

POODLE是一种已知的 SSL/TLS 攻击，最初破坏了 SSLv3。从 3.4.0 版本开始，RabbitMQ 服务器拒绝接受 SSLv3 连接。2014 年 12 月，发布了影响 TLSv1.0 的 POODLE 攻击的修改版本。因此建议运行 Erlang 18.0 或更高版本，这 消除了 POODLE 的 TLS 1.0 实现漏洞，或者禁用 TLSv1.0 支持。

#### BEAST

BEAST攻击是影响 TLSv1.0 的已知漏洞。要缓解它，请禁用 TLSv1.0 支持

## 流（Streams）

Streams 是RabbitMQ 3.9中的一种新的持久和复制数据结构，它对具有非破坏性消费者语义的仅附加日志进行建模。它们可以像队列一样通过 RabbitMQ 客户端库使用，也可以通过 专用的二进制协议 插件和关联的客户端使用。推荐使用后一个选项，因为它可以访问所有特定于流的功能并提供最佳吞吐量（性能）。

### 应用场景

流的开发最初涵盖了 4 个现有队列类型无法提供或具有缺点的消息传递用例：

#### 大量消费者

当想要将相同的消息传递给多个订阅者时，用户当前必须为每个消费者绑定一个专用队列。如果消费者的数量很大，这可能会变得低效，尤其是在需要持久性和/或复制时。Streams 将允许任意数量的消费者以非破坏性的方式使用来自同一个队列的相同消息，从而无需绑定多个队列。流消费者也将能够从副本中读取，从而允许读取负载分布在整个集群中。

#### 重消费

由于所有当前的 RabbitMQ 队列类型都具有破坏性的消费行为，即当消费者使用完消息后，消息会从队列中删除，因此无法重新读取已消费的消息。Streams 将允许消费者在日志中的任何位置附加并从那里读取。

#### 吞吐性能

没有任何持久队列类型能够提供可以与任何现有的基于日志的消息传递系统竞争的吞吐量。流的设计以性能为主要目标。

#### 大量的日志

大多数 RabbitMQ 队列旨在收敛到空状态并进行优化，当给定队列上有数百万条消息时性能会更差。流旨在以有效的方式以最小的内存开销存储大量数据。

### 用法

可以指定可选队列和使用者参数的 AMQP 0.9.1 客户端库 将能够将流用作常规 AMQP 0.9.1 队列。就像队列一样，必须先声明流。

要声明流，请将x-queue-type队列参数设置为stream（默认为classic）。该参数必须由客户在声明时提供；它不能使用策略设置或更改。这是因为策略定义或适用策略可以动态更改，但队列类型不能。必须在申报时指定。

1. ConnectionFactory factory = **new** ConnectionFactory();
2. Connection connection = factory.newConnection();
3. Channel channel = connection.createChannel();
4. channel.queueDeclare(
5. "my-stream",
6. **true**,         // 持久的
7. **false**, **false**, // 不排他，不自动删除
8. Collections.singletonMap("x-queue-type", "stream")
9. );

声明一个将x-queue-type参数设置为stream的队列将在每个配置的 RabbitMQ 节点上创建一个具有副本的流。流是仲裁系统，因此强烈建议使用不均匀的集群大小。流仍然是 AMQP 0.9.1 队列，因此它可以在创建后绑定到任何交换，就像任何其他 RabbitMQ 队列一样。

Streams 支持 3 个额外的队列参数 ，这些参数最好使用策略进行配置：

* x-max-length-bytes：以字节为单位设置流的最大大小。
* x-max-age：设置流的最大年龄。
* x-stream-max-segment-size-bytes：单位：字节。一个流在磁盘上被分成固定大小的段文件。此设置控制这些的大小。

1. Map<String, Object> arguments = **new** HashMap<>();
2. arguments.put("x-queue-type", "stream");
3. arguments.put("x-max-length-bytes", 20\_000\_000\_000); // 最大流大小：20 GB
4. arguments.put("x-stream-max-segment-size-bytes", 100\_000\_000); // 段文件大小：100 MB
5. channel.queueDeclare(
6. "my-stream",
7. **true**,         // durable
8. **false**, **false**, // not exclusive, not auto-delete
9. arguments
10. );

### 客户端操作

由于流永远不会删除任何消息，因此任何消费者都可以从日志中的任何点开始读取/消费。这由x-stream-offset消费者参数控制。如果未指定，消费者将从消费者启动后写入日志的下一个偏移量开始读取。支持以下值：

* first - 从日志中的第一条可用消息开始
* last- 这从最后写入的消息“块”开始读取（块是流中使用的存储和传输单元，简单地说，它是由几条到几千条消息组成的一批消息，具体取决于入口）
* next - 与不指定任何偏移量相同
* 偏移量 - 一个数值，指定要附加到日志的确切偏移量。如果此偏移量不存在，它将分别钳制到日志的开头或结尾。
* 时间戳 - 一个时间戳值，指定附加到日志的时间点。它将钳制到最接近的偏移量，如果时间戳超出流的范围，它将分别钳制日志的开始或结束。对于 AMQP 0.9.1，使用的时间戳是 POSIX 时间，精度为一秒，即自 1970 年 1 月 1 日 00:00:00 UTC 以来的秒数。
* 间隔 - 一个字符串值，指定相对于当前时间附加日志的时间间隔。使用与x-max-age相同的规范

1. channel.basicQos(100); // 必须指定 QoS
2. channel.basicConsume(
3. "my-stream",
4. **false**,
5. Collections.singletonMap("x-stream-offset", "first"), // "first" 偏移规范
6. (consumerTag, message) -> {
7. // 消息处理
8. // ...
9. channel.basicAck(message.getEnvelope().getDeliveryTag(), **false**); // 需要确认
10. },
11. consumerTag -> { });

### 其他

以下操作可以以与经典队列和仲裁队列类似的方式使用，但有些操作具有一些特定于队列的行为。

* 声明
* 队列删除
* 确认
* 消费订阅
* 为消费者设置QoS 预取
* 取消消费者

流并不是传统意义上的真正队列，因此与 AMQP 0.9.1 的队列语义不太一致。其他队列类型支持的许多功能不受支持，并且永远不会由于队列类型的性质而受到支持。只要使用消费者确认，可以使用常规镜像队列的 AMQP 0.9.1 客户端库将能够使用流。由于它们的非破坏性读取语义，许多特性永远不会被流支持。特征比较：

| **特征** | **经典的** | **溪流** |
| --- | --- | --- |
| 非持久队列 | 是的 | 不 |
| [排他性](https://www.rabbitmq.com/queues.html) | 是的 | 不 |
| 每条消息持久化 | 每条消息 | 总是 |
| 会员变更 | 自动的 | 手动的 |
| [TTL](https://www.rabbitmq.com/ttl.html) | 是的 | 否 |
| [队列长度限制](https://www.rabbitmq.com/maxlength.html) | 是的 | 否 |
| [懒惰的行为](https://www.rabbitmq.com/lazy-queues.html) | 是的 | 固有 |
| [消息优先级](https://www.rabbitmq.com/priority.html) | 是的 | 不 |
| [消费者优先](https://www.rabbitmq.com/consumer-priority.html) | 是的 | 不 |
| [死信交换](https://www.rabbitmq.com/dlx.html) | 是的 | 不 |
| 遵守[政策](https://www.rabbitmq.com/parameters.html#policies) | 是的 |  |
| [对记忆警报](https://www.rabbitmq.com/alarms.html)作出反应 | 是的 | 否 |
| 中毒消息处理 | 不 | 不 |
| 全局[QoS 预取](https://www.rabbitmq.com/streams.html#global-qos) | 是的 | 不 |

## 功能标志

在混合版本集群中（例如，一些版本是 3.7.x，一些是 3.8.x），一些节点将支持一组不同的功能，在某些场景下表现不同，否则行为不完全相同：它们毕竟是不同的版本.功能标志是一种控制哪些功能在所有集群节点上被视为启用或可用的机制。如果启用了功能标志，则其关联的功能（或行为）也是如此。如果不是，那么集群中的所有节点都将禁用该功能（行为）。特性标志子系统允许具有不同版本的 RabbitMQ 节点确定它们是否兼容，然后一起通信，而不管它们的版本如何。这个子系统是在 RabbitMQ 3.8.0 中引入的，允许在不关闭整个集群的情况下滚动升级集群成员。

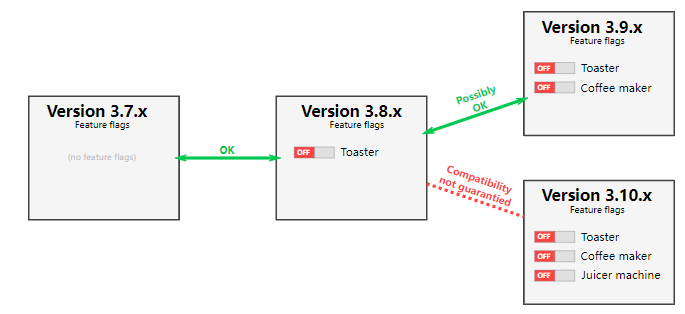
* 仅当集群中的所有节点都支持时，才能启用功能标志。
* 只有在以下情况下，节点才能加入或重新加入集群：
  + 它支持集群中启用的所有功能标志，并且
  + 如果集群支持在该节点上启用的所有功能标志。
* 只要没有启用 3.8.x 功能标志，RabbitMQ 3.7.x 和 3.8.x 节点就兼容。

两个主要命令

* 列出功能标志：rabbitmqctl list\_feature\_flags
* 要启用功能标志（或所有当前禁用的标志）：rabbitmqctl enable\_feature\_flag <全部 |名称>

### [RabbitMQ 版本](https://www.rabbitmq.com/feature-flags.html" \l "version-compatibility)

如前所述，功能标志子系统的主要目标是允许升级，无论 RabbitMQ 的版本如何，如果可能的话。因此，从 RabbitMQ 3.8.0 开始，可以升级到下一个补丁，次要或主要版本，除非在发行说明中另有说明。确实，有些更改不能作为功能标志来实现。



也可以从 RabbitMQ 3.7.x 升级到 3.8.x。实际上，RabbitMQ 3.7.x 没有功能标志子系统，并且 RabbitMQ 3.8.x 认为 3.7.x 节点有一个空的功能标志列表。因此，只要 3.8.x 节点禁用其所有功能标志，它就与 3.7.x 节点兼容。

但是，请注意，仅支持从一个次要升级到下一个次要或主要。要从例如 3.6.16 升级到 3.8.7，必须先升级到 3.7.28。同样，如果在使用的次要版本和下一个主要版本之间存在一个或多个次要版本分支。这可能有效（即主要版本之间可能没有不兼容的更改），但这种情况在设计上不受支持，原因如下：

* 跳过次要版本不在 CI 中测试。
* 非顺序版本可能支持也可能不支持同一组功能标志。可以删除对旧功能标志的支持。几个次要分支存在标志，它们已被删除，并且它们的关联功能/行为现在默认隐式启用，防止与旧节点进行集群。功能标志保留在一些（比如两个）次要版本中，以允许过渡期。
* 功能标志的弃用/删除策略尚未定义

### 操作

当一个节点第一次启动时，默认启用所有支持的功能标志。当节点升级到较新版本的 RabbitMQ 时，如果它是单个隔离节​​点，则默认启用新功能标志，如果它属于集群，则默认保持禁用状态。要列出功能标志，请使用

rabbitmqctl list\_feature\_flags

# => 列出功能标志 ...

# => 名称状态

# => empty\_basic\_get\_metric 已启用

# => 启用隐式默认绑定

# => quorum\_queue 已启用

为了提高表格的可读性，请切换到pretty\_table格式化程序：

rabbitmqctl -q --formatter pretty\_table list\_feature\_flags \

* name：功能标志的名称。
* state：如果启用或禁用功能标志，则启用或禁用，如果集群中的一个或多个节点不知道此功能标志（因此无法启用），则不支持。
* provided\_by：提供功能标志的 RabbitMQ 组件或插件。
* desc：功能标志的描述。
* doc\_url：网页的 URL 以了解有关功能标志的更多信息。
* stability：指示功能标志是稳定的还是 实验性的。

要启用功能标志，使用：

rabbitmqctl enable\_feature\_flag <名称>

要启用所有功能标志，请使用：

rabbitmqctl enable\_feature\_flag <全部>

list\_feature\_flags命令可再次用于验证功能标志的状态。

下面列出的功能标志是由 RabbitMQ 核心或与 RabbitMQ 捆绑的第 1 层插件之一提供的。

| **功能标志名称** | **介绍** |
| --- | --- |
| [empty\_basic\_get\_metric](https://www.rabbitmq.com/feature-flags.html" \l "ff-empty_basic_get_metric) | 计数 AMQP 0-9-1 basic.get在统计中的空队列上发出。 |
| [implicit\_default\_bindings](https://www.rabbitmq.com/feature-flags.html#ff-implicit_default_bindings) | 清除显式默认交换绑定，因为它们是隐式管理的。 |
| [quorum\_queue](https://www.rabbitmq.com/feature-flags.html#ff-quorum_queue) | 启用仲裁队列类型。 |
| [stream\_queue](https://www.rabbitmq.com/feature-flags.html#ff-stream_queue) | 启用流。 |
| [drop\_unroutable\_metric](https://www.rabbitmq.com/feature-flags.html#ff-drop_unroutable_metric) | 删除了无法路由的消息指标。 |
| [maintenance\_mode\_status](https://www.rabbitmq.com/feature-flags.html#ff-maintenance_mode_status) | 启用维护模式，因此可以耗尽节点进行维护或升级操作。 |
| [user\_limits](https://www.rabbitmq.com/feature-flags.html#ff-user_limits) | 启用与给定用户关联的连接和队列限制。 |
| [virtual\_host\_metadata](https://www.rabbitmq.com/feature-flags.html#ff-virtual_host_metadata) | 启用虚拟主机元数据。 |

当使用rabbitmqctl启用功能标志时，内部会发生以下情况：

* RabbitMQ 验证功能标志是否已启用。如果是，它会停止。
* 它验证是否支持功能标志。如果没有，它会停止。
* 它将功能标志状态标记为state\_change。这是一个内部过渡状态，用于通知消费者此功能标志。大多数情况下，这意味着依赖于此特定功能标志的组件将被阻止，直到状态更改为 enabled或disabled。
* 它启用了这个依赖的所有功能标志。因此，对于它们中的每一个，我们都会经历相同的过程。
* 它执行迁移功能，如果有的话。该函数负责准备或转换各种资源，例如更改数据库的模式。
* 如果上述所有步骤都成功，则功能标志状态变为 enabled。否则，它会恢复为disabled。

## 分布式RabbitMQ

AMQP 0-9-1、AMQP 1.0 和 RabbitMQ 通过插件（例如 STOMP）支持的其他消息传递协议（当然）本质上是分布式的：应用程序几乎总是连接到远程主机上的 RabbitMQ。通常有必要或希望使 RabbitMQ 代理本身分布式。有三种方法可以做到这一点：使用集群、使用联合和使用 Shovel 插件。本页概述了每种方法。请注意，所有三种方法都不是相互排斥的，并且可以组合使用：集群可以使用 Federation 或Shovel连接在一起，或两者兼而有之。

### Clustering（聚类）

集群将多台机器连接在一起形成一个集群。节点间通信对客户端透明地执行。集群的设计假设网络连接相当可靠并提供类似LAN的延迟。集群中的所有节点都必须运行兼容的 RabbitMQ 和Erlang版本。节点使用通常由部署自动化工具安装的预共享密钥相互验证。

虚拟主机、交换器、用户和权限会 自动复制到集群中的所有节点。

队列可能位于单个节点上，或复制其内容以获得更高的可用性。 Quorum queues是一种现代复制队列类型，专注于数据安全。 Streams是另一种允许重复使用（读取）的复制消息传递数据类型。连接到集群中任何节点的客户端可以使用集群中的所有非独占队列，即使它们不在该节点上。集群节点可以帮助提高队列内容的可用性、数据安全性并维持更多的并发客户端连接。

### Federation（联合）

联合允许一个代理上的交换或队列接收发布到另一个交换或队列的消息（代理可能是单个机器或集群）。通信是通过 AMQP（带有可选的 SSL）进行的，因此两个交换或队列要联合起来，它们必须被授予适当的用户和权限。联合交易所通过单向点对点链路连接。默认情况下，消息只会通过联合链路转发一次，但可以增加此值以允许更复杂的路由拓扑。

有些消息可能无法通过链接转发；如果消息在到达联合交换后不会被路由到队列，那么它首先不会被转发。联合队列类似地通过单向点对点链路连接。消息将在联合队列之间移动任意次数以跟随消费者。通常，您会使用联合来链接 Internet 上的代理，以进行发布/订阅消息传递和工作队列。

### Shovel（铲子）

使用Shovel 插件连接代理在概念上类似于将它们与联合连接。但是，该插件在较低级别上工作。虽然联合旨在提供交换和队列的自以为是的分布，但铲子只是从一个代理上的队列中消费消息，并将它们转发到另一个代理上的交换。

通常，当您需要比联合提供的更多控制权时，您会使用铲子通过 Internet 链接代理。动态铲子对于在单个代理上以特殊方式移动消息也很有用。

|  |  |
| --- | --- |
| **Federation and/or Shovel** | **Clustering** |
| Brokers在逻辑上是独立的，可能有不同的所有者。 | 一个集群形成一个单一的逻辑代理。 |
| 代理可以运行不同（并且在某些方面不兼容）版本的 RabbitMQ 和 Erlang。 | 节点必须运行兼容版本的RabbitMQ 和 Erlang。 |
| Brokers可以通过不可靠的 WAN 链接进行连接。通过 AMQP 0-9-1 进行通信（可选地由 **TLS**保护），需要设置适当的用户和权限。 | 经纪人必须通过相当可靠的局域网链接进行连接。节点将使用共享密钥相互验证，并可选择**使用启用 TLS 的链接**。 |
| 代理可以在您安排的任何拓扑中连接。链接可以是单向或双向的。 | 所有节点都在两个方向上连接到所有其他节点。 |
| **强调CAP 定理** 的可用性和分区容限 (AP) 。 | **强调CAP 定理** 的一致性和分区容差 (CP) 。 |
| Brokers中的一些交易所可能是联合的，而一些可能是本地的。 | 聚类是全有或全无的。 |
| 连接到任何代理的客户端只能使用该代理中的非独占队列。 | 连接到任何节点的客户端可以在所有节点上使用非独占队列。 |

## Clustering（聚类）

一个 RabbitMQ 集群是一个或多个节点的逻辑分组，每个节点共享用户、虚拟主机、队列、交换器、绑定、运行时参数和其他分布式状态。

### 集群构建

可以通过多种方式形成 RabbitMQ 集群：

* 通过在配置文件中列出集群节点以声明方式
* 以声明方式使用基于 DNS 的发现
* 以声明方式使用AWS (EC2) 实例发现（通过插件）
* 以声明方式使用Kubernetes 发现（通过插件）
* 以声明方式使用基于 Consul 的发现（通过插件）
* 以声明方式使用基于 etcd 的发现（通过插件）
* 手动使用rabbitmqctl

#### 节点名称（标识符）

RabbitMQ 节点由节点名称标识。节点名称由两部分组成，前缀（通常是rabbit）和主机名。例如，rabbit@node1.messaging.svc.local 是一个节点名，前缀为rabbit，主机名为 node1.messaging.svc.local。集群中的节点名称必须是唯一的。如果在给定主机上运行多个节点（这通常是开发和QA环境中的情况），它们必须使用不同的前缀，例如rabbit1@hostname和rabbit2@hostname。在集群中，节点使用节点名称相互识别和联系。这意味着每个节点名称的主机名部分都必须解析。

CLI 工具还使用节点名称识别和寻址节点。当一个节点启动时，它会检查它是否被分配了一个节点名。这是通过RABBITMQ\_NODENAME 环境变量完成的。如果没有显式配置值，则节点解析其主机名并在其前面添加rabbit以计算其节点名。如果系统使用完全限定域名 (FQDN) 作为主机名，则必须将 RabbitMQ 节点和 CLI 工具配置为使用所谓的长节点名。对于服务器节点，这是通过将RABBITMQ\_USE\_LONGNAME环境变量设置为true来完成的。对于CLI工具，必须设置 RABBITMQ\_USE\_LONGNAME 或必须指定--longnames选项。

#### 主机名解析

RabbitMQ 节点使用短域名或全限定域名 (FQDN) 相互寻址。因此，所有集群成员的主机名必须可以从所有集群节点以及可能使用rabbitmqctl等命令行工具的机器上解析。主机名解析可以使用任何标准操作系统提供的方法：

* DNS 记录
* 本地主机文件（例如/etc/hosts）

在更严格的环境中，DNS 记录或主机文件修改受到限制、不可能或不受欢迎，Erlang VM 可以配置为使用替代主机名解析方法，例如替代 DNS 服务器、本地文件、非标准主机文件位置，或多种方法。这些方法可以与标准操作系统主机名解析方法协同工作。

#### 端口访问

RabbitMQ 节点绑定到端口（打开服务器 TCP 套接字）以接受客户端和 CLI 工具连接。SELinux 等其他进程和工具可能会阻止 RabbitMQ 绑定到端口。发生这种情况时，节点将无法启动。

CLI 工具、客户端库和 RabbitMQ 节点也打开连接（客户端 TCP 套接字）。防火墙可以阻止节点和 CLI 工具相互通信。以下端口与集群中的节点间通信最相关：

* 4369：epmd，RabbitMQ 节点和 CLI 工具使用的帮助程序发现守护程序
* 6000 到 6500（通常是 6000、6001、6002 等到 6005）：由RabbitMQ 流复制使用
* 25672：用于节点间和 CLI 工具通信（Erlang 分发服务器端口），从动态范围分配（默认限制为单个端口，计算为 AMQP 端口 + 20000）。除非这些端口上的外部连接确实是必要的（例如集群使用联合或在子网外的机器上使用 CLI 工具），否则这些端口不应公开。
* 35672-35682：由 CLI 工具（Erlang 分发客户端端口）用于与节点通信，并从动态范围分配（计算为服务器分发端口 + 10000 到服务器分发端口 + 10010）。

#### 集群中的节点

RabbitMQ 代理操作所需的所有数据/状态都在所有节点之间复制。一个例外是消息队列，默认情况下它们驻留在一个节点上，尽管它们对所有节点都是可见和可访问的。要跨集群中的节点复制队列，请使用支持复制的队列类型。Quorum Queues指南中介绍了该主题。

一些分布式系统具有领导者和追随者节点。对于 RabbitMQ，这通常不是真的。RabbitMQ 集群中的所有节点都是平等的：RabbitMQ 核心中没有特殊节点。当考虑到仲裁队列和插件时，这个主题变得更加细微，但对于大多数意图和目的，所有集群节点都应该被认为是平等的。

许多CLI 工具操作可以针对任何节点执行。HTTP API客户端可以针对任何集群节点。个别插件可以指定（选择）某些节点在一段时间内“特殊”。例如，联合链接位于特定的集群节点上。如果该节点发生故障，链接将在另一个节点上重新启动。

#### 节点验证

RabbitMQ 节点和 CLI 工具（例如rabbitmqctl）使用 cookie 来确定是否允许它们相互通信。为了使两个节点能够通信，它们必须具有相同的共享密钥，称为 Erlang cookie。cookie 只是一串最多 255 个字符的字母数字字符。它通常存储在本地文件中。该文件必须只能由所有者访问（例如，具有600或类似的 UNIX 权限）。每个集群节点都必须有相同的 cookie。

如果该文件不存在，Erlang VM 将在 RabbitMQ 服务器启动时尝试使用随机生成的值创建一个。使用此类生成的 cookie 文件仅适用于开发环境。由于每个节点都会独立生成自己的值，因此这种策略在集群环境中并不真正可行。Erlang cookie 的生成应该在集群部署阶段完成，最好使用自动化和编排工具

Cookie 文件位置，Linux、MacOS、\*BSD：在UNIX系统上，cookie通常位于/var/lib/rabbitmq/.erlang.cookie（由服务器使用）和$HOME/.erlang.cookie（由 CLI 工具使用）。请注意，由于$HOME的值因用户而异，因此有必要为将使用 CLI 工具的每个用户放置 cookie 文件的副本。这适用于非特权用户和root。

Docker社区RabbitMQ映像使用RABBITMQ\_ERLANG\_COOKIE环境变量值来填充 cookie文件。使用此镜像的配置管理和容器编排工具必须确保集群中的每个 RabbitMQ 节点容器使用相同的值。在Kubernetes的上下文中，该值必须在有状态集的 pod 模板规范中指定。例如，这可以在RabbitMQ on Kubernetes示例存储库中看到。

对于从 20.2 开始的 Erlang 版本，cookie 文件的位置是：

* %HOMEDRIVE%%HOMEPATH%\.erlang.cookie（对于用户%USERNAME%通常是C:\Users\%USERNAME%\.erlang.cookie），如果同时设置了HOMEDRIVE和HOMEPATH环境变量
* %USERPROFILE%\.erlang.cookie（通常是C:\Users\%USERNAME%\.erlang.cookie）如果HOMEDRIVE和HOMEPATH都没有设置
* 对于RabbitMQ Windows 服务-%USERPROFILE%\.erlang.cookie（通常是C:\WINDOWS\system32\config\systemprofile）

如果使用Windows服务，则应将cookie从 C:\Windows\system32\config\systemprofile\.erlang.cookie 复制到用户运行rabbitmqctl.bat等命令的预期位置。

**使用 CLI 和运行时命令行参数覆盖**：作为替代方案，可以将选项“ -setcookie <value> ”添加到RABBITMQ\_SERVER\_ADDITIONAL\_ERL\_ARGS 环境变量值 以覆盖 RabbitMQ 节点使用的 cookie 值：

RABBITMQ\_SERVER\_ADDITIONAL\_ERL\_ARGS= "-setcookie cookie 值"

CLI 工具可以使用命令行标志获取 cookie 值：

rabbitmq-diagnostics status --erlang-cookie "cookie-value"

两者都是最不安全的选项，通常不推荐。

当 cookie 配置错误（例如，不相同）时，RabbitMQ 节点将记录错误，例如“来自不允许节点的连接尝试”、“”、“无法自动集群”。

#### 节点计数和仲裁

因为有几个特性（例如quorum queues，MQTT 中的客户端跟踪）需要集群成员之间达成共识，所以强烈建议使用奇数个集群节点：1、3、5、7 等等。强烈建议不要使用两个节点集群，因为集群节点不可能在连接丢失的情况下识别多数并形成共识。例如，当两个节点失去连接时，MQTT 客户端连接将不被接受，仲裁队列将失去可用性，等等。从共识的角度来看，四节点或六节点集群将具有与三节点和五节点集群相同的可用性特征。

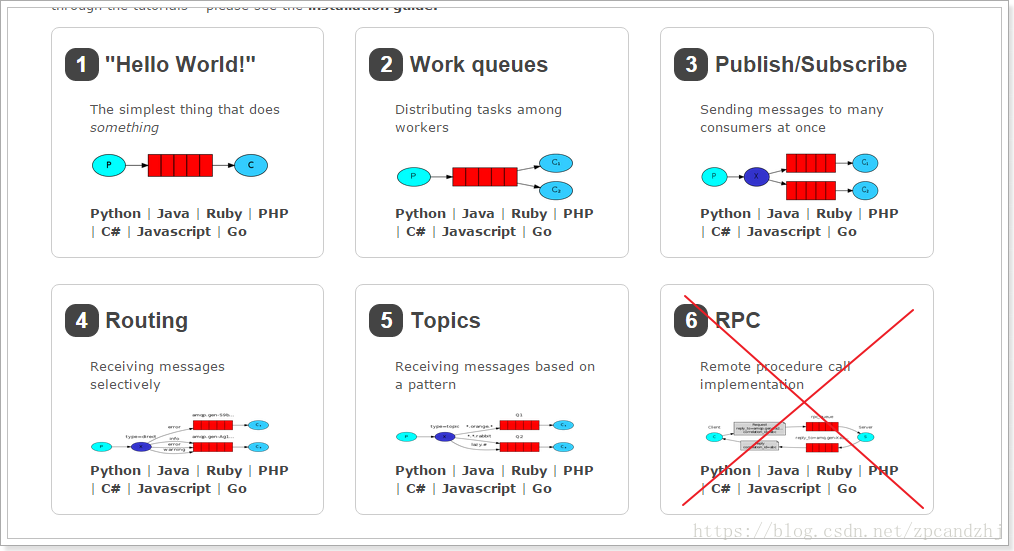
假设所有集群成员都可用，客户端可以连接到任何节点并执行任何操作。节点会将操作路由到 仲裁队列领导者或队列领导者副本 ，对客户端透明。对于所有支持的消息传递协议，客户端一次只能连接到一个节点。在节点故障的情况下，客户端应该能够重新连接到不同的节点，恢复其拓扑并继续操作。出于这个原因，大多数客户端库都接受端点列表（主机名或 IP 地址）作为连接选项。如果客户端支持，主机列表将在初始连接和连接恢复期间使用。

客户端连接、通道和队列将分布在集群节点上。操作员需要能够跨所有集群节点检查和监控此类资源：

RabbitMQ CLI工具（例如rabbitmq-diagnostics和rabbitmqctl）提供了检查资源和集群范围状态的命令。一些命令关注单个节点的状态（例如rabbitmq-diagnostics environment和rabbitmq-diagnostics status），其他命令检查集群范围的状态。后者的一些示例包括rabbitmqctl list\_connections、rabbitmqctl list\_mqtt\_connections、rabbitmqctl list\_stomp\_connections、rabbitmqctl list\_users、rabbitmqctl list\_vhosts等。

这种“集群范围”的命令通常会首先联系一个节点，发现集群成员并联系他们以检索和组合它们各自的状态。例如，rabbitmqctl list\_connections将联系所有节点，检索它们的AMQP 0-9-1 和 AMQP 1.0 连接，并将它们全部显示给用户。用户不必手动联系所有节点。假设集群状态不变（例如，没有关闭或打开连接），对两个不同节点一个接一个执行的两个 CLI 命令将产生相同或语义相同的结果。然而，“节点本地”命令不会产生相同的结果，因为两个节点很少有相同的状态：至少它们的节点名称会不同！

**5.五种队列**

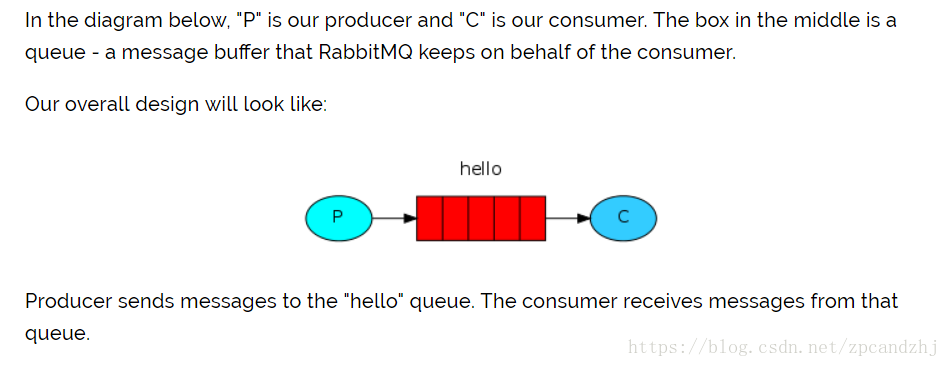


**5.1.导入my-rabbitmq项目**

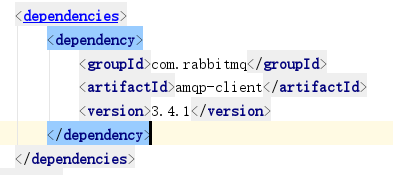
**5.2.简单队列**

**5.2.1.图示**

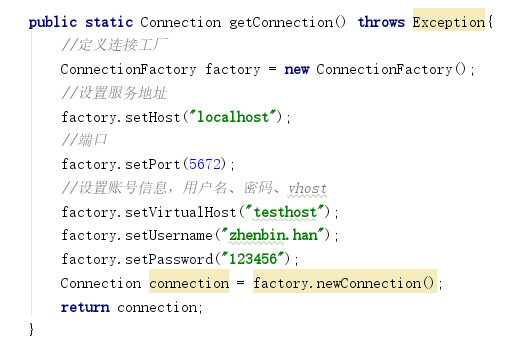
在下面的图表中，“P”是我们的生产者，“C”是我们的消费者。中间的框是一个队列—RabbitMQ代表消费者保存的消息缓冲区。我们的整体设计如下:



生产者向“hello”队列发送消息。使用者从该队列接收消息。



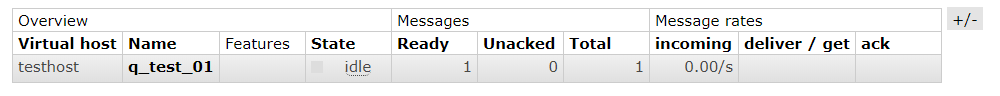
获取MQ的连接工具类

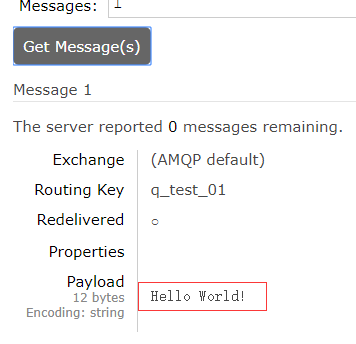


生产者发送消息到队列

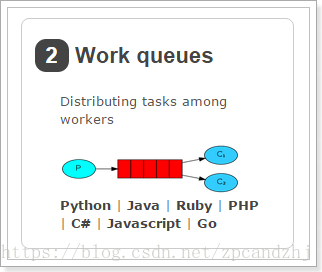
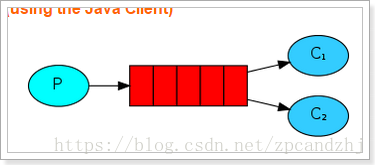


管理工具中查看消息



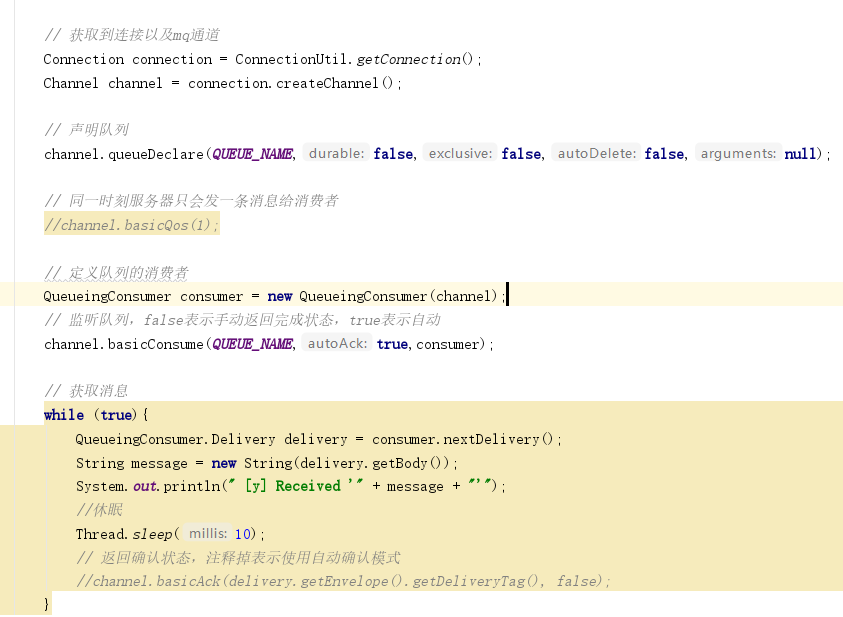


**5.3.Work模式**

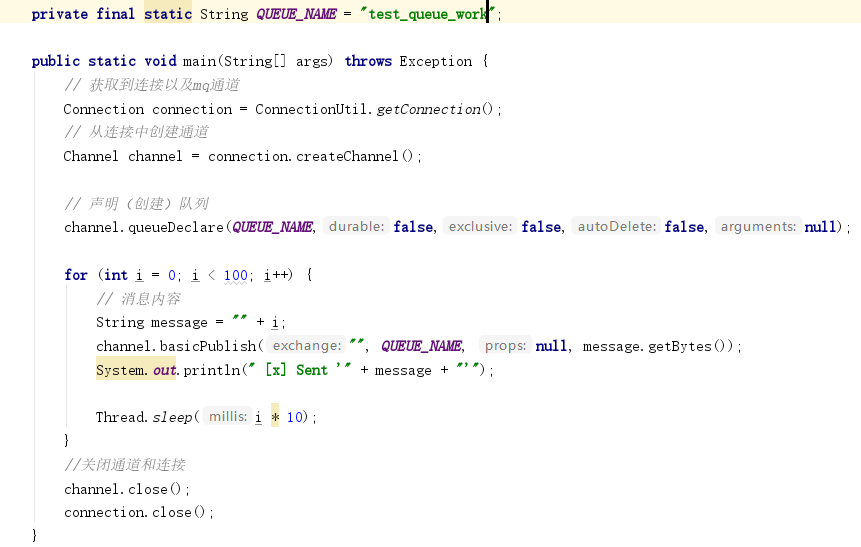
 

一个生产者、2个消费者。一个消息只能被一个消费者获取。

消费者两个



生产者一个



2个概念

* 轮询分发 ：使用任务队列的优点之一就是可以轻易的并行工作。如果我们积压了好多工作，我们可以通过增加工作者（消费者）来解决这一问题，使得系统的伸缩性更加容易。在默认情况下，RabbitMQ将逐个发送消息到在序列中的下一个消费者(而不考虑每个任务的时长等等，且是提前一次性分配，并非一个一个分配)。平均每个消费者获得相同数量的消息。这种方式分发消息机制称为Round-Robin（轮询）。
* 公平分发 ：虽然上面的分配法方式也还行，但是有个问题就是：比如：现在有2个消费者，所有的奇数的消息都是繁忙的，而偶数则是轻松的。按照轮询的方式，奇数的任务交给了第一个消费者，所以一直在忙个不停。偶数的任务交给另一个消费者，则立即完成任务，然后闲得不行。而RabbitMQ则是不了解这些的。这是因为当消息进入队列，RabbitMQ就会分派消息。它不看消费者为应答的数目，只是盲目的将消息发给轮询指定的消费者。

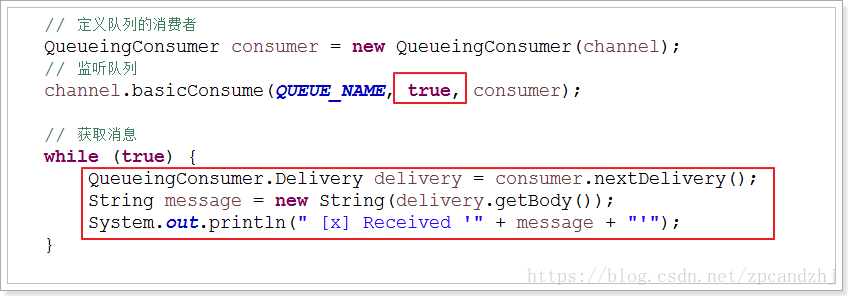
为了解决这个问题，我们使用basicQos( prefetchCount = 1)方法，来限制RabbitMQ只发不超过1条的消息给同一个消费者。当消息处理完毕后，有了反馈，才会进行第二次发送。还有一点需要注意，使用公平分发，必须关闭自动应答，改为手动应答。

**5.5.消息的确认模式**

消费者从队列中获取消息，服务端如何知道消息已经被消费呢？

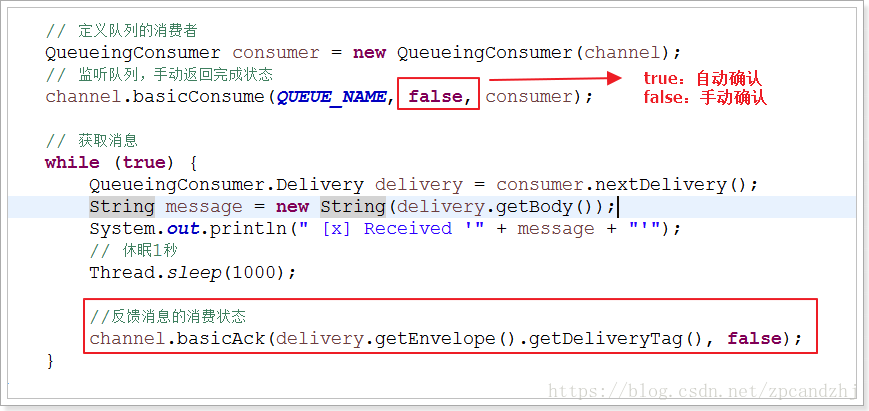
模式1：自动确认

只要消息从队列中获取，无论消费者获取到消息后是否成功消息，都认为是消息已经成功消费。

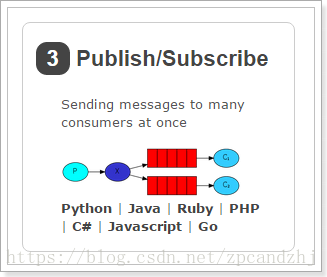


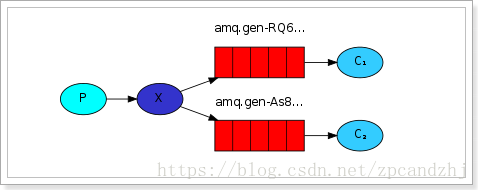
模式2：手动确认

消费者从队列中获取消息后，服务器会将该消息标记为不可用状态，等待消费者的反馈，如果消费者一直没有反馈，那么该消息将一直处于不可用状态。



**5.6.订阅模式**





解读：

1、1个生产者，多个消费者

2、每一个消费者都有自己的一个队列

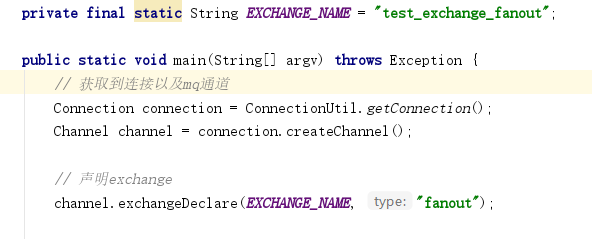
3、生产者没有将消息直接发送到队列，而是发送到了交换机

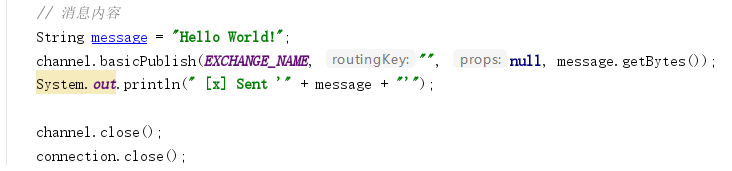
4、每个队列都要绑定到交换机

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

注意：一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费

消息的生产者，向交换机中发送消息。

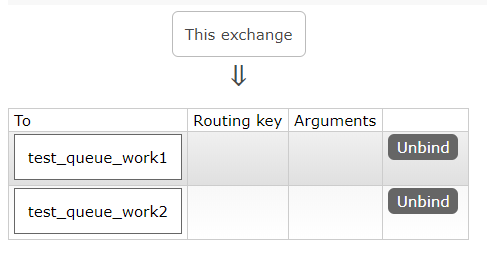




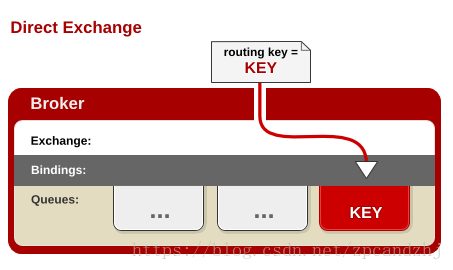
消费者

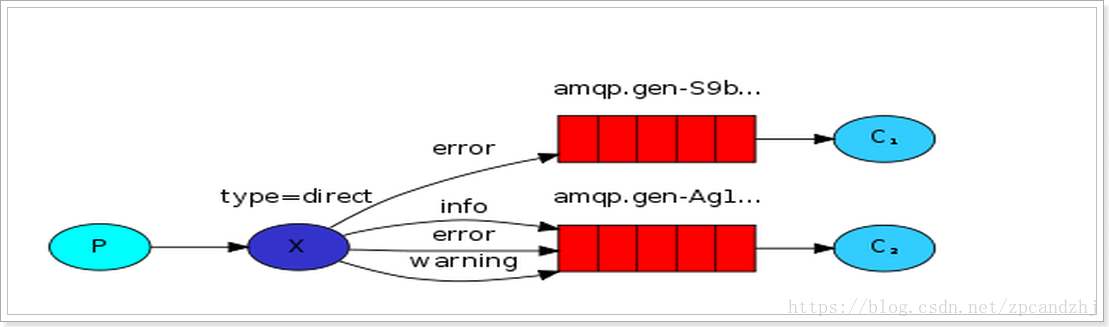


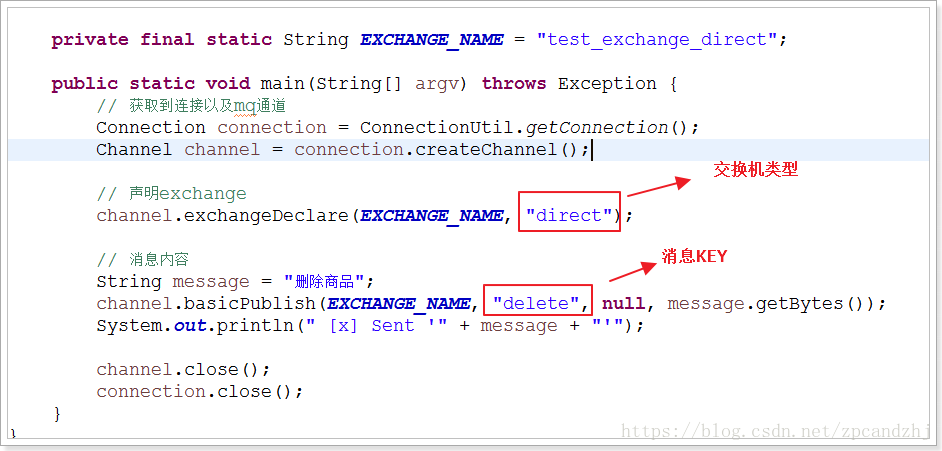
查看队列和交换机的绑定关系：

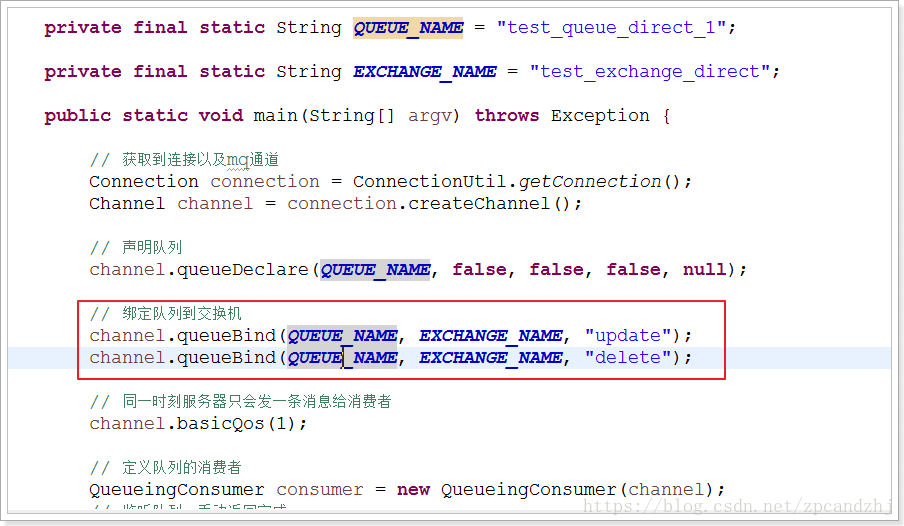


**5.7.路由模式**



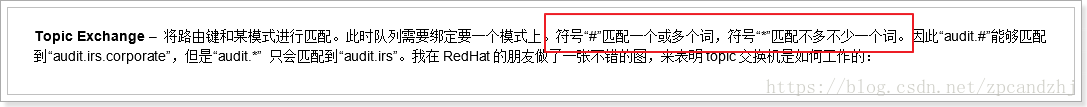


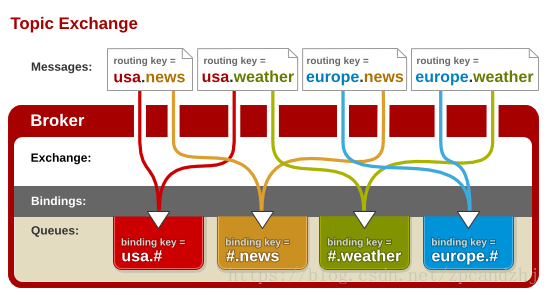


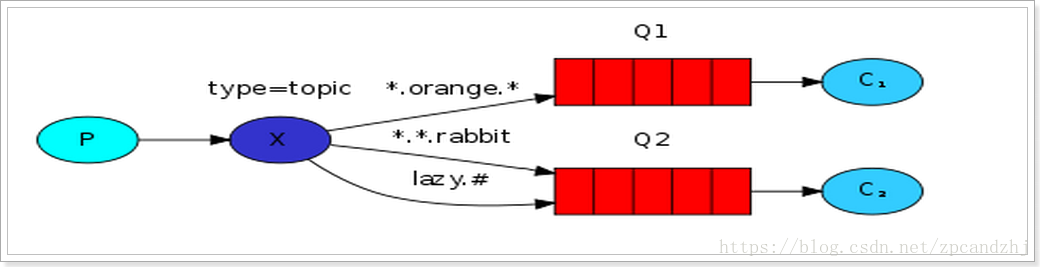




**5.8.主题模式（通配符模式）**







同一个消息被多个消费者获取。一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费到消息。



