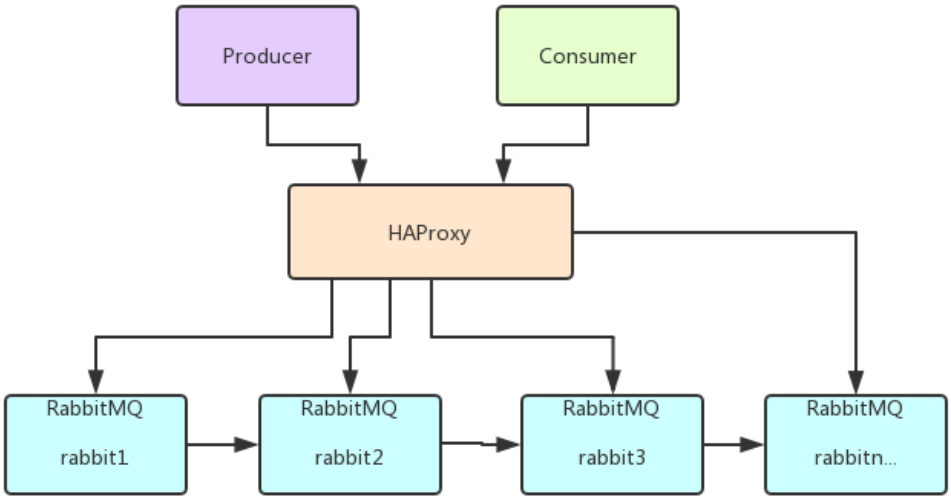
RabbitMQ高可用集群搭建

摘要：实际生产应用中都会采用消息队列的集群方案，如果选择RabbitMQ那么有必要了解下它的集群方案原理

一般来说，如果只是为了学习RabbitMQ或者验证业务工程的正确性那么在本地环境或者测试环境上使用其单实例部署就可以了，但是出于MQ中间件本身的可靠性、并发性、吞吐量和消息堆积能力等问题的考虑，在生产环境上一般都会考虑使用RabbitMQ的集群方案。

1 、集群方案的原理

RabbitMQ这款消息队列中间件产品本身是基于Erlang编写，Erlang语言天生具备分布式特性（通过同步Erlang集群各节点的magic cookie来实现）。因此，RabbitMQ天然支持Clustering。这使得RabbitMQ本身不需要像ActiveMQ、Kafka那样通过ZooKeeper分别来实现HA方案和保存集群的元数据。集群是保证可靠性的一种方式，同时可以通过水平扩展以达到增加消息吞吐量能力的目的。



2 、一般模式集群

2.1 准备工作

准备三台虚拟机

192.168.223.128

192.168.223.129

192.168.223.130

首先确保RabbitMQ运行没有问题

```
[root@super ~]# rabbitmqctl status
Status of node rabbit@super ...
[{pid,10232},
 {running_applications,
     [{rabbitmq_management,"RabbitMQ Management Console","3.6.5"},
      {rabbitmq_web_dispatch,"RabbitMQ Web Dispatcher","3.6.5"},
```

```
      {webmachine,"webmachine","1.10.3"},
      {mochiweb,"MochiMedia Web Server","2.13.1"},
      {rabbitmq_management_agent,"RabbitMQ Management Agent","3.6.5"},
      {rabbit,"RabbitMQ","3.6.5"},
      {os_mon,"CPO  CXC 138 46","2.4"},
      {syntax_tools,"Syntax tools","1.7"},
      {inets,"INETS  CXC 138 49","6.2"},
      {amqp_client,"RabbitMQ AMQP Client","3.6.5"},
      {rabbit_common,[],"3.6.5"},
      {ssl,"Erlang/OTP SSL application","7.3"},
      {public_key,"Public key infrastructure","1.1.1"},
      {asn1,"The Erlang ASN1 compiler version 4.0.2","4.0.2"},
      {ranch,"Socket acceptor pool for TCP protocols.","1.2.1"},
      {mnesia,"MNESIA  CXC 138 12","4.13.3"},
      {compiler,"ERTS  CXC 138 10","6.0.3"},
      {crypto,"CRYPTO","3.6.3"},
      {xmerl,"XML parser","1.3.10"},
      {sasl,"SASL  CXC 138 11","2.7"},
      {stdlib,"ERTS  CXC 138 10","2.8"},
      {kernel,"ERTS  CXC 138 10","4.2"}]},
 {os,{unix,linux}},
 {erlang_version,
     "Erlang/OTP 18 [erts-7.3] [source] [64-bit] [async-threads:64] [hipe]
[kernel-poll:true]\n"},
 {memory,
     [{total,56066752},
      {connection_readers,0},
      {connection_writers,0},
      {connection_channels,0},
      {connection_other,2680},
      {queue_procs,268248},
      {queue_slave_procs,0},
      {plugins,1131936},
      {other_proc,18144280},
      {mnesia,125304},
      {mgmt_db,921312},
      {msg_index,69440},
      {other_ets,1413664},
      {binary,755736},
      {code,27824046},
      {atom,1000601},
      {other_system,4409505}]},
 {alarms,[]},
 {listeners,[{clustering,25672,"::"},{amqp,5672,"::"}]},
 {vm_memory_high_watermark,0.4},
 {vm_memory_limit,411294105},
 {disk_free_limit,50000000},
 {disk_free,13270233088},
 {file_descriptors,
     [{total_limit,924},{total_used,6},{sockets_limit,829},{sockets_used,0}]},
 {processes,[{limit,1048576},{used,262}]},
 {run_queue,0},
 {uptime,43651},
 {kernel,{net_ticktime,60}}]
```

RabbitMQ的集群是依赖erlang集群，而erlang集群是通过这个cookie进行通信认证的，因此我们搭集群的第一步就是干cookie。怎么办?

必须使集群中也就是这3台机器的.erlang.cookie文件中cookie值一致，且权限为owner只读。

机器192.168.223.128中的Cookie:

```
[root@G ~]# cat .erlang.cookie
ATOAYQTPNAFZIVULUQRX
```

机器192.168.223.129中的Cookie:

```
[root@F ~]# cat .erlang.cookie
ATOAYQTPNAFZIVULUQRX[root@F ~]#
```

机器192.168.223.130中的Cookie:

```
[root@F ~]# cat .erlang.cookie
ATOAYQTPNAFZIVULUQRX[root@F ~]#
```

修改文件权限如下：

```
cd /var/lib/rabbitmq
chmod 600 .erlang.cookie
```

停止rabbitmq服务

```
systemctl stop rabbitmq-server
```

2.2 配置DNS域名解析

为了方便，不需要写一长串的ip地址，配置域名解析，vim /etc/hosts

```
192.168.223.128 ydt1
192.168.223.129 ydt2
192.168.223.130 ydt3
```

保证相互之间能够ping通

```
[root@ydt rabbitmq]# ping ydt2
PING ydt2 (192.168.223.129) 56(84) bytes of data.
64 bytes from ydt2 (192.168.223.129): icmp_seq=1 ttl=64 time=0.371 ms
64 bytes from ydt2 (192.168.223.129): icmp_seq=2 ttl=64 time=0.347 ms
64 bytes from ydt2 (192.168.223.129): icmp_seq=3 ttl=64 time=0.360 ms
64 bytes from ydt2 (192.168.223.129): icmp_seq=4 ttl=64 time=0.254 ms
64 bytes from ydt2 (192.168.223.129): icmp_seq=5 ttl=64 time=0.322 ms
```

2.3 配置集群启动

先把三台机器的防火墙先关了，如果你不想关，可以关闭各自的4369集群通信端口

```
#关闭防火墙
systemctl stop firewalld
-------------------------------------------------------------------------------
#开启端口,其他端口照做
firewall-cmd --zone=public --add-port=4369/tcp --permanent
#重启防火墙
firewall-cmd --reload
#查看端口号是否开启
firewall-cmd --query-port=4369/tcp
#测试是否可以访问虚拟机端口
telnet 192.168.223.128 4369
```

集群模式启动三个节点，并设置节点名（如果集群关系已经建立，后面不用操作了）：

```
#当然，你也可以使用后台启动：RABBITMQ_NODENAME=rabbit1 rabbitmq-server -detached
RABBITMQ_NODENAME=rabbit1 rabbitmq-server start
RABBITMQ_NODENAME=rabbit2 rabbitmq-server start
RABBITMQ_NODENAME=rabbit3 rabbitmq-server start
```

关闭命令

```
rabbitmqctl -n rabbit1 stop
rabbitmqctl -n rabbit2 stop
rabbitmqctl -n rabbit3 stop
```

PS：以下操作如果集群关系已经建立的情况下就不需要再执行了

rabbit1操作作为主节点：

```
rabbitmqctl -n rabbit1 stop_app
rabbitmqctl -n rabbit1 reset
rabbitmqctl -n rabbit1 start_app
```

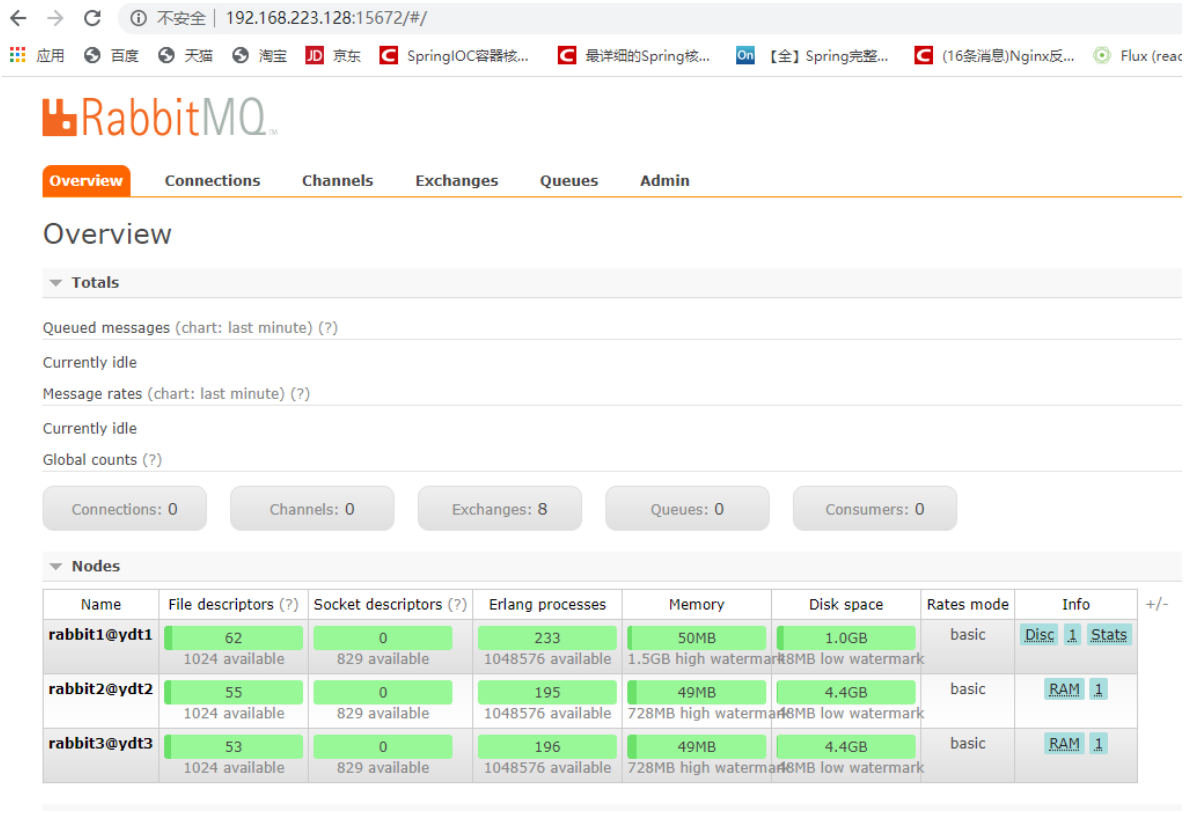
rabbit2/rabbit3操作为从节点：

```
#rabbit2
rabbitmqctl -n rabbit2 stop_app
rabbitmqctl -n rabbit2 reset
rabbitmqctl -n rabbit2 join_cluster rabbit1@ydt1 --ram #主机名换成自己的
rabbitmqctl -n rabbit2 start_app #应用启动

#rabbit3
rabbitmqctl -n rabbit3 stop_app
rabbitmqctl -n rabbit3 reset
rabbitmqctl -n rabbit3 join_cluster rabbit1@ydt1 --ram #主机名换成自己的
rabbitmqctl -n rabbit3 start_app  #应用启动
```

查看集群状态：

```
[root@ydt1 ~]# rabbitmqctl cluster_status -n rabbit1
Cluster status of node rabbit1@ydt1 ...
[{nodes,[{disc,[rabbit1@ydt1,rabbit2@ydt2,rabbit3@ydt3]}]},
 {running_nodes,[rabbit3@ydt3,rabbit2@ydt2,rabbit1@ydt1]},
 {cluster_name,<<"rabbit1@ydt1">>},
 {partitions,[]},
 {alarms,[{rabbit3@ydt3,[]},{rabbit2@ydt2,[]},{rabbit1@ydt1,[]}]}]
```

登录任意节点管理控制台查看



3、集群管理

rabbitmqctl join_cluster {cluster_node} [--disc/ram]
将节点加入指定集群中。在这个命令执行前需要停止RabbitMQ应用并重置节点。

rabbitmqctl cluster_status
显示集群的状态。

rabbitmqctl change_cluster_node_type {disc|ram}
修改集群节点的类型（磁盘还是内存）。在这个命令执行前需要停止RabbitMQ应用。

#节点类型
集群中的节点一般有两种,一种是内存节点,一种是磁盘节点,内存节点由于没有磁盘读写,性能比磁盘节点好,磁盘节点可以将状态持久化到磁盘,可用性比内存节点好,需要权衡考虑.本次用的一台作为磁盘节点,做数据备份,两台内存节点,用于提高性能.

Rabbitmq要求在集群中至少有一个磁盘节点,所有其他节点可以是内存节点,当节点加入或者离开集群时,必须要将该变更通知到至少一个磁盘节点.如果集群中唯一的磁盘节点崩溃的话,集群仍然可以保持运行,但是无法进行其他的操作(增删改查),直到节点恢复.

```
rabbitmqctl -n rabbit3 stop_app  #先停止应用服务
rabbitmqctl change_cluster_node_type ram #修改节点类型
rabbitmqctl -n rabbit3 start_app #启动应用服务生效
```

rabbitmqctl forget_cluster_node [--offline]
将节点从集群中删除，允许离线执行（停机后执行）。

```
rabbitmqctl -n rabbit3 stop_app
rabbitmqctl -n rabbit1 forget_cluster_node rabbit3@ydt3 #在rabbit1节点移除集群中的
rabbit3节点
```

```
#当rabbit3再次启动时会报错如下：
[root@ydt3 ~]# rabbitmqctl -n rabbit3 start_app
Starting node rabbit3@ydt3 ...

BOOT FAILED
===========

Error description:
   {error,{inconsistent_cluster,"Node rabbit3@ydt3 thinks it's clustered with
node rabbit2@ydt2, but rabbit2@ydt2 disagrees"}}

Log files (may contain more information):
   /var/log/rabbitmq/rabbit3.log
   /var/log/rabbitmq/rabbit3-sasl.log

Stack trace:
   [{rabbit_mnesia,check_cluster_consistency,0,
                   [{file,"src/rabbit_mnesia.erl"},{line,598}]},
    {rabbit,'-start/0-fun-0-',0,[{file,"src/rabbit.erl"},{line,260}]},
    {rabbit,start_it,1,[{file,"src/rabbit.erl"},{line,403}]},
    {rpc,'-handle_call_call/6-fun-0-',5,[{file,"rpc.erl"},{line,206}]}]

Error: {error,{inconsistent_cluster,"Node rabbit3@ydt3 thinks it's clustered
with node rabbit2@ydt2, but rabbit2@ydt2 disagrees"}}

#再次加入集群需如下执行命令
rabbitmqctl -n rabbit3 stop_app
rabbitmqctl -n rabbit3 reset #等同于:rm -rf /var/lib/rabbitmq/mnesia/
rabbitmqctl -n rabbit3 join_cluster rabbit1@ydt1 --ram #主机名换成自己的
rabbitmqctl -n rabbit3 start_app  #应用启动
```

rabbitmqctl update_cluster_nodes {clusternode}

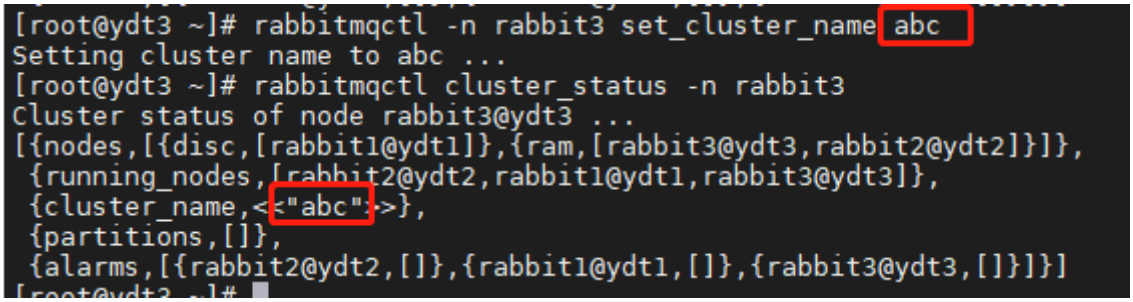
在集群中的节点应用启动前咨询clusternode节点的最新信息，并更新相应的集群信息。这个和join_cluster不同，它不加入集群。考虑这样一种情况，节点A和节点B都在集群中，当节点A离线了，节点C又和节点B组成了一个集群，然后节点B又离开了集群，当A醒来的时候，它会尝试联系节点B，但是这样会失败，因为节点B已经不在集群中了。

```
[root@ydt3 ~]# rabbitmqctl -n rabbit3 stop_app
Stopping node rabbit3@ydt3 ...
[root@ydt3 ~]# rabbitmqctl -n rabbit3 update_cluster_nodes rabbit1@ydt1 #检查节点
rabbit1还在不在集群中
Updating cluster nodes for rabbit3@ydt3 from rabbit1@ydt1 ...
```

rabbitmqctl cancel_sync_queue [-p vhost] {queue}取消队列queue同步镜像的操作

```
[root@ydt3 ~]# rabbitmqctl -n rabbit3 cancel_sync_queue -p cluster
cluster_queue2
Stopping synchronising queue 'cluster_queue2' in vhost 'cluster' ...
{"init terminating in do_boot",{function_clause,
[{rabbit_control_misc,print_cmd_result,[cancel_sync_queue,not_syncing],
[{file,"src/rabbit_control_misc.erl"},{line,92}]},{rabbit_cli,main,3,
[{file,"src/rabbit_cli.erl"},{line,89}]},{init,start_it,1,[]},{init,start_em,1,
[]}]}}
init terminating in do_boot ()
```

rabbitmqctl set_cluster_name {name}
设置集群名称。集群名称在客户端连接时会通报给客户端。Federation和Shovel插件也会有用到集群名称的地方。集群名称默认是集群中第一个节点的名称，通过这个命令可以重新设置。

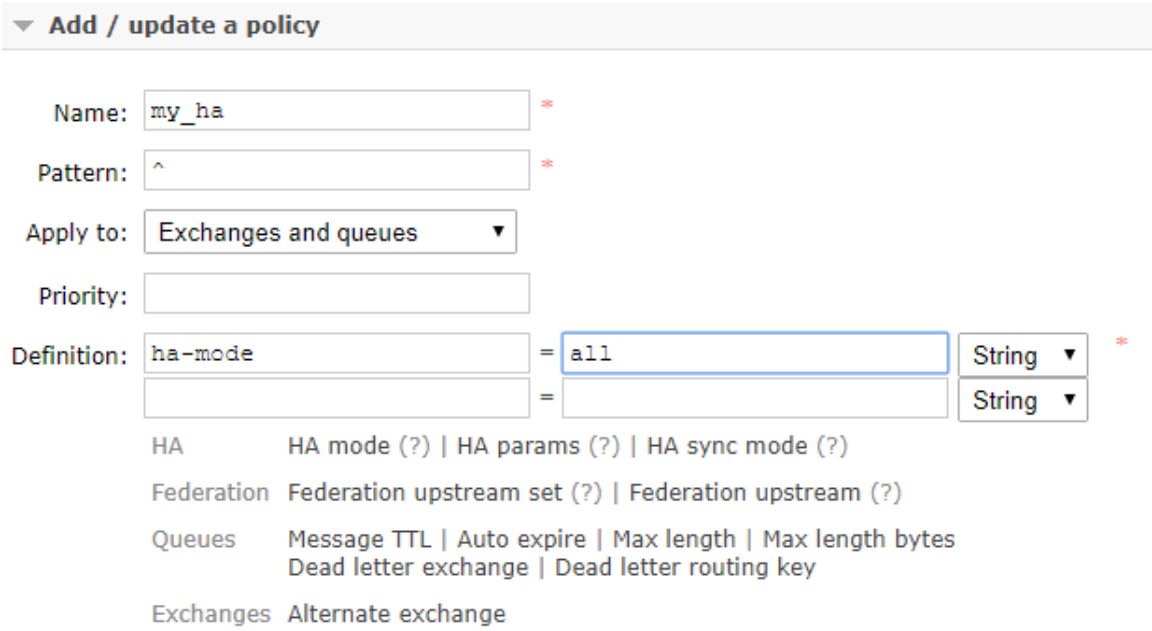


4、RabbitMQ镜像集群配置

> 默认的集群模式，queue创建之后，如果没有其它policy，则queue就会按照普通模式集群。对于Queue来说，消息实体只存在于其中一个节点，A、B两个节点仅有相同的元数据，即队列结构，但队列的元数据仅保存有一份，即创建该队列的rabbitmq节点（A节点），当A节点宕机，你可以去其B节点查看，./rabbitmqctl list_queues发现该队列已经丢失，但声明的exchange还存在。
>
> 当消息进入A节点的Queue中后，consumer从B节点拉取时，RabbitMQ会临时在A、B间进行消息传输，把A中的消息实体取出并经过B发送给consumer，所以consumer应平均连接每一个节点，从中取消息。该模式存在一个问题就是当A节点故障后，B节点无法取到A节点中还未消费的消息实体。如果做了队列持久化或消息持久化，那么得等A节点恢复，然后才可被消费，并且在A节点恢复之前其它节点不能再创建A节点已经创建过的持久队列；如果没有持久化的话，消息就会失丢。这种模式更适合非持久化队列，只有该队列是非持久的，客户端才能重新连接到集群里的其他节点，并重新创建队列。假如该队列是持久化的，那么唯一办法是将故障节点恢复起来。
>
> 镜像队列是基于普通的集群模式的，然后再添加一些策略，所以你还是得先配置普通集群，然后才能设置镜像队列，我们就以上面的集群接着做。
>
> 镜像模式：把需要的队列做成镜像队列，存在于多个节点，属于RabbitMQ的HA方案
>
> 该模式解决了上述问题，其实质和普通模式不同之处在于，消息实体会主动在镜像节点间同步，而不是在consumer取数据时临时拉取。该模式带来的副作用也很明显，除了降低系统性能外，如果镜像队列数量过多，加之大量的消息进入，集群内部的网络带宽将会被这种同步通讯大大消耗掉。所以在对可靠性要求较高的场合中适用，一个队列想做成镜像队列，需要先设置policy，然后客户端创建队列的时候，rabbitmq集群根据“队列名称”自动设置是普通集群模式或镜像队列。

设置的镜像队列可以通过开启的网页的管理端Admin->Policies，也可以通过命令。

> rabbitmqctl set_policy my_ha "^" '{"ha-mode":"all"}'



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

5 、负载均衡-HAProxy

HAProxy提供高可用性、负载均衡以及基于TCP和HTTP应用的代理，支持虚拟主机，它是免费、快速并且可靠的一种解决方案,包括Twitter，Reddit，StackOverflow，GitHub在内的多家知名互联网公司在使用。HAProxy实现了一种事件驱动、单一进程模型，此模型支持非常大的并发连接数。

5.1 安装HAProxy

```
#下载依赖包，有就不用装了
yum install gcc vim wget
//上传haproxy源码包 haproxy-1.6.5.tar.gz，已提供

//解压
tar -zxvf haproxy-1.6.5.tar.gz -C /usr/local
//进入目录、进行编译、安装
cd /usr/local/haproxy-1.6.5
make TARGET=linux31 PREFIX=/usr/local/haproxy
make install PREFIX=/usr/local/haproxy
mkdir /etc/haproxy
//赋权
groupadd -r -g 149 haproxy
useradd -g haproxy -r -s /sbin/nologin -u 149 haproxy
//创建haproxy配置文件
vim /etc/haproxy/haproxy.cfg
```

5.2 配置HAProxy

配置文件路径：/etc/haproxy/haproxy.cfg

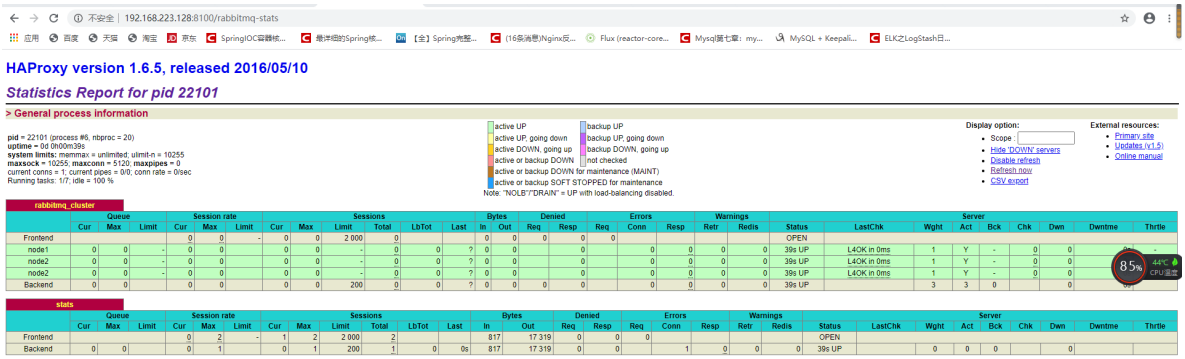
```
#logging options
global
        log 127.0.0.1 local0 info
        maxconn 5120 #最大连接数，默认4000
        chroot /usr/local/haproxy
        uid 99
        gid 99
        daemon
        quiet
        nbproc 20
        pidfile /var/run/haproxy.pid #haproxy的pid存放路径,启动进程的用户必须有权限访
问此文件

defaults
        log global

        mode tcp

        option tcplog
        option dontlognull
        retries 3
        option redispatch
        maxconn 2000
        contimeout 5s

     clitimeout 60s

     srvtimeout 15s
#front-end IP for consumers and producters

listen rabbitmq_cluster
        bind 192.168.223.128:5671 #监听端口
        mode tcp
        #balance url_param userid
        #balance url_param session_id check_post 64
        #balance hdr(User-Agent)
        #balance hdr(host)
        #balance hdr(Host) use_domain_only
        #balance rdp-cookie
        #balance leastconn
        #balance source //ip

        balance roundrobin

        server node1 192.168.223.128:5672 check inter 5000 rise 2 fall 2
        server node2 192.168.223.129:5672 check inter 5000 rise 2 fall 2
        server node2 192.168.223.130:5672 check inter 5000 rise 2 fall 2

listen stats
        bind 192.168.223.128:8100 #监听端口
        mode http
        option httplog
        stats enable
        stats uri /rabbitmq-stats #统计页面url
        stats refresh 5s #统计页面自动刷新时间
```

5.3 启动HAproxy负载

```
/usr/local/haproxy/sbin/haproxy -f /etc/haproxy/haproxy.cfg
//查看haproxy进程状态
ps -ef | grep haproxy
```

访问如下地址对mq节点进行监控

http://192.168.223.128:8100/rabbitmq-stats



6、java代码测试负载均衡代理

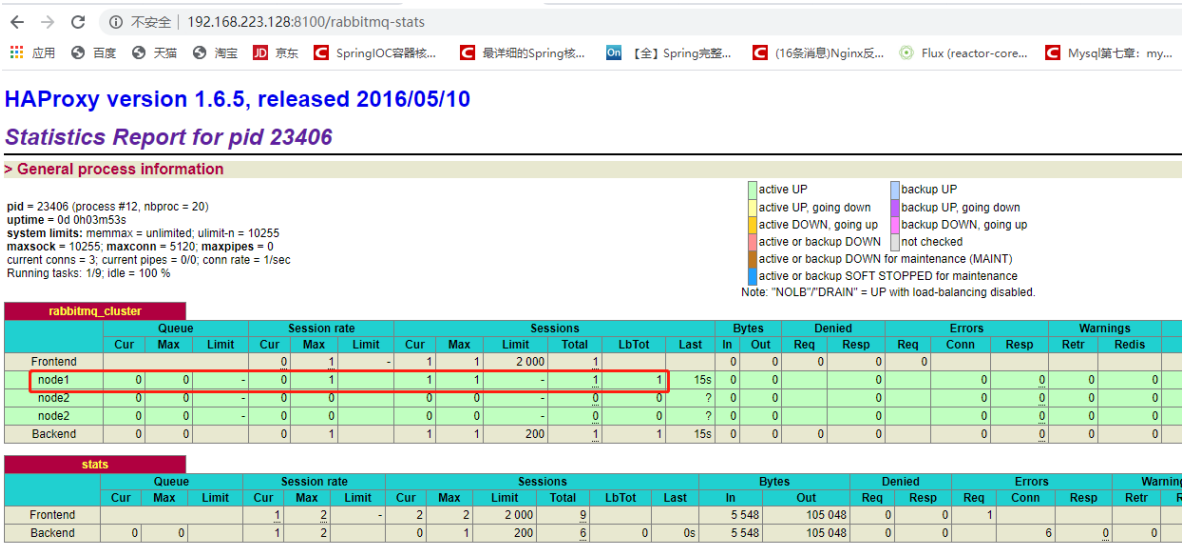
代码中访问mq集群地址，则变为访问haproxy地址:5671

```
package com.ydt.rabbitmq.cluster;

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

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


public class Producer {
    public static void main(String[]args) throws IOException,TimeoutException{
        ConnectionFactory factory=new ConnectionFactory();
        //配置连接属性
        factory.setHost("192.168.223.128");
        factory.setPort(5671);//haproxy监听端口
        factory.setVirtualHost("/");
        factory.setUsername("guest");
        factory.setPassword("guest");

        //得到连接
        Connection connection=factory.newConnection();
        //创建通道
        Channel channel=connection.createChannel();
        //声明（创建）队列
        String queueName="cluster_queue";
        channel.queueDeclare(queueName, true, false, false, null);
        //发送消息
        String message="Hello, RabbitMQ Cluster";
        for(int i=0;i<200;i++){
            channel.basicPublish("",queueName, null, message.getBytes("UTF-8"));
            try {
                Thread.sleep(500);
```

```
                System.out.println("Consumer: "+message+"---"+i);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        channel.close();
        connection.close();
    }
}
```

此时我们将客户端连接改为192.168.223.128的前端haproxy负载均衡服务器的IP和端口，测试发送消息，后端集群节点都能同步到消息

首先执行第一次消息投递，同时观察128上haproxy管控台后端连接到哪个rabbitmq节点



起多个生产者同时发送：