六星教育-2007_SRM-01-06.mq与mongodb等集群构建.md

0. 课程内容

-1. 问题

1. 在GoodsController中定义一个注解@Reference （pool="goods.pool"），它是如何加载到 Been容器中的？
2. $this->goodsService->getList(12, 'type'); 这样一行代码为什么就调用了Rpc？
3. 如何加载RpcProvider 的 ？

1. 商品client与consul负载均衡

修改客户端代码

1. `app\client\swoft\bin\bootstrap.php`

```php
/** @var \Composer\Autoload\ClassLoader $loader */
$loader = require dirname( path: __DIR__) . '/vendor/autoload.php';
$retCall = " ";
if ($extInitCalls = strstr($argv[$argc - 1], needle: 'ext_init')) {
    $initCalls = (explode( delimiter: '=', $extInitCalls))[1];
    if ($calls = strstr($initCalls, needle: '?')) {
        $calls = explode( delimiter: '?', $initCalls);
        foreach ($calls as $key => $call) {
            $retCall .= str_replace( search: ":", replace: " ", $call)." ";
        }
    } else {
        $retCall .= str_replace( search: ":", replace: " ", $initCalls)." ";
    }
}
echo $retCall;
// exit;
exec( command: 'sh /var/www/init.sh '.$retCall, &output: $out, &return_var: $status);
// $loader->addPsr4('Swoft\\Cache\\', 'vendor/swoft/cache/src/');
```

2. `app\client\swoft\bin\bootstrap.php`

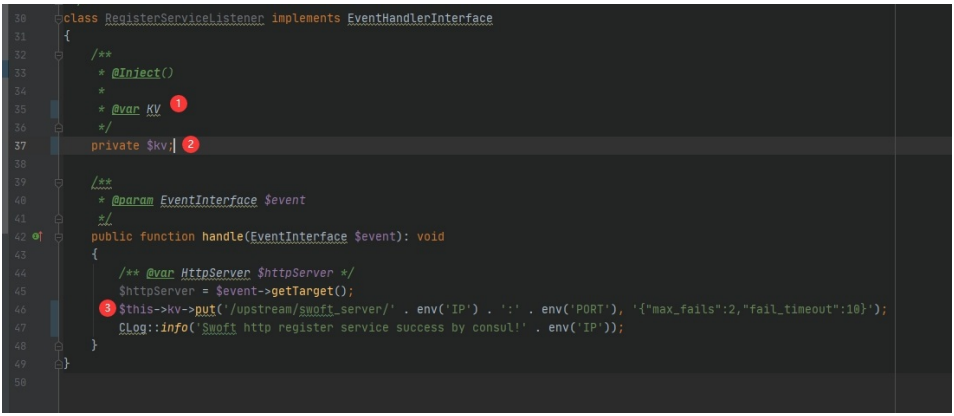
```php
]);

// Run application
$app = new \App\Application();
$app->setEnvFile( envFile: "/var/www/.env");
$app->run();
```

3. `init.sh` （需注意本地编辑器上传至服务器后脚本的格式 set ff=uinx）

```
#!/bin/sh
host=`ip addr | grep /24 | awk '{print $2}' | awk -F '/' '{print $1}'`
ip=127.0.0.1
type=tcp
port=80
while getopts ":i::t::p::" opt
do
    case $opt in
        i)
            ip=$OPTARG
            ;;
        t)
            type=$OPTARG
            ;;
        p)
            port=$OPTARG
            ;;
        ?)
            ;;
    esac
done
echo -e "HOST=$host\nIP=$ip\nTYPE=$type\nPORT=$port" > /var/www/.env
```

4. 注册客户端 `app\client\swoft\app\Listener\RegisterServiceListener.php`



5. 清除事件 `app\client\swoft\app\Listener\DeregisterServiceListener.php`

六星教育

```php
    /**
     * @Inject()
     *
     * @var KV
     */
    private $kv;

    /**
     * @param EventInterface $event
     */
    public function handle(EventInterface $event): void
    {
        /** @var HttpServer $httpServer */
        $httpServer = $event->getTarget();
        $this->kv->delete('/upstream/swoft_server/' . env('IP') . ':' . env('PORT'));
    }
}
```

6. consul的注册效果

六星教育

Key / Values upstream

swoft_server

Name

192.168.175.100:18316

192.168.175.100:18326

7. 配置nginx `nginx\conf\nginx.conf` 在另外一台虚拟机上 `192.168.175.110`

```
upstream swoft_server {
    server 192.168.175.100:18316 max_fails=2 fail_timeout=30s;
    upsync 172.33.33.60:8500/v1/kv/upstream/swoft_server upsync_timeout=10s upsync_interval=600ms upsync_type=consul strong_dependency=on;
    # 这是容器中的目录
    upsync_dump_path /nginx/conf/swoft_server.conf;
    include /nginx/conf/swoft_server.conf;
}
```

2. 代码热加载

swoft官方是有提供一个方式来实现热加载的，但是因为官方的方式并不是很适合与我们目前的场景；因此只能自己实现

`app\client\swoft\bin\inotify.php`

```
<?php
class Inotify
{
```


```
    // .. .
}

function watchEvent($call)
{
   // $type = $argv[1] ;
   return function($event) use ($call){
       // ...
       exec($call, $result, $status);
   };
}

array_shift($argv);
$call = implode(' ', $argv);
$swoftPath = __DIR__."/swoft ";

$pid = pcntl_fork();
if ($pid == 0) {
    exec("php ".$swoftPath.$call, $result, $status);
} else {
    $call = str_replace('start', 'restart', $call);
    $inotify = new Inotify(__DIR__.'/../', watchEvent("php ".$swoftPath.$call));
    $inotify->start();
}
?>
```

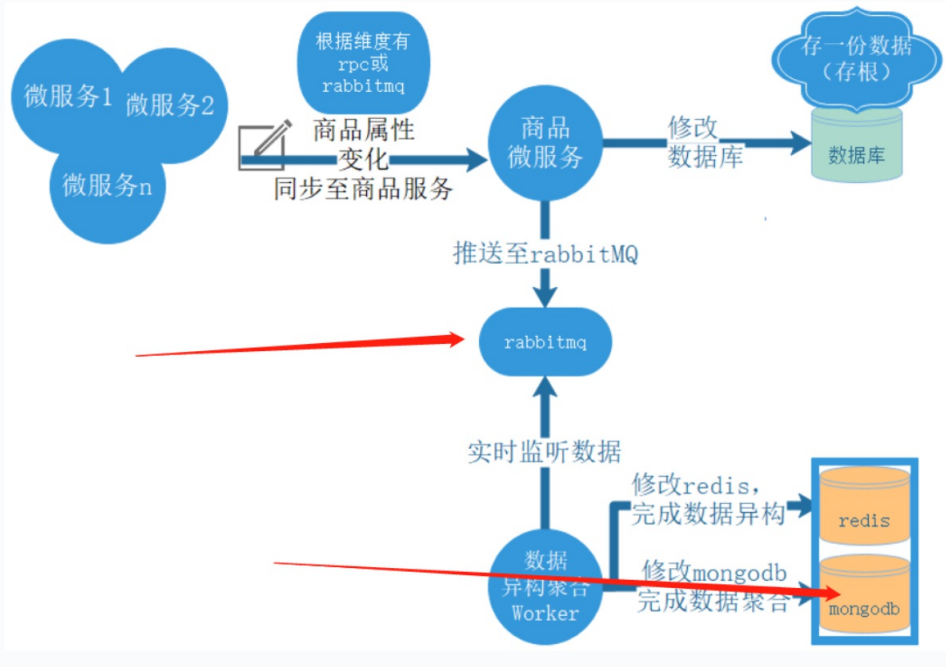
利用inotify对整个swoft进行监控，然后重启swoft:

思路：

1. 通过pcntl_fork创建出两个进程
2. 子进程运行swoft，主进程启动文件监听
3. 发送变化就重启swoft

需要注意此操作需要linux系统安装inotify-tools以及PHP扩展inotify

3. rabbitmq高可用集群



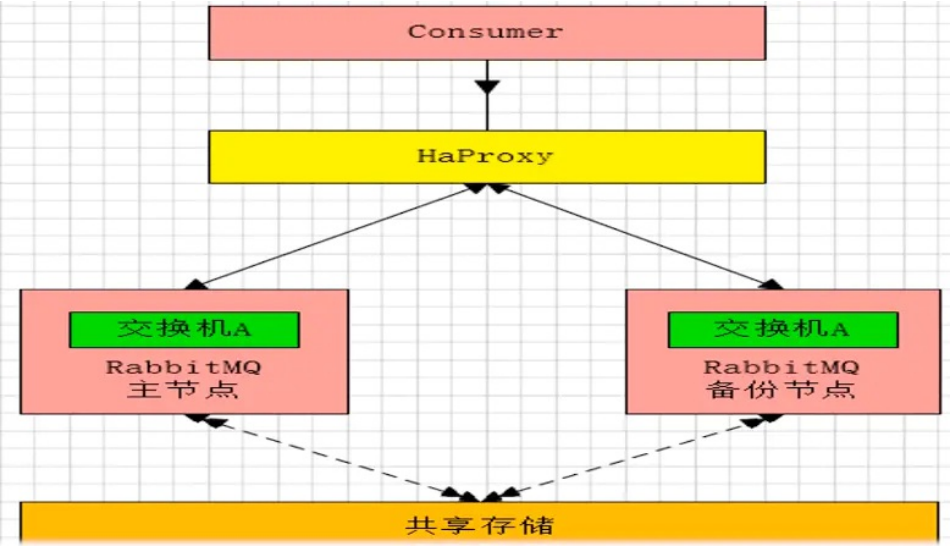
在服务之间会采用mq进行消息通信，而rabbitmq本身也如同consul一样，如果只有一个节点那么就可能出现宕机的问题，并且基于mq的特点我们是可以在多个服务之间使用同一个mq来相互通信，因此高可用的架构设计就必不可少

3.1. rabbitmq集群方案

1. 主备模式

一般在并发和数据量不高的情况下，这种模式非常的好用且简单。

也就是一个主/备方案，主节点提供读写，备用节点不提供读写。如果主节点挂了，就切换到备用节点，原来的备用节点升级为主节点提供读写服务，当原来的主节点恢复运行后，原来的主节点就变成备用节点，和 activeMQ 利用 zookeeper 做主/备一样，也可以一主多备



2. 远程模式

这是 rabbitMQ 比较早期的架构模型了，现在很少使用了

远程模式可以实现双活的一种模式，简称 shovel 模式，所谓的 shovel 就是把消息进行不同数据中心的复制工作，可以跨地域的让两个 MQ 集群互联，远距离通信和复制

Shovel 就是我们可以把消息进行数据中心的复制工作，我们可以跨地域的让两个 MQ 集群互联。

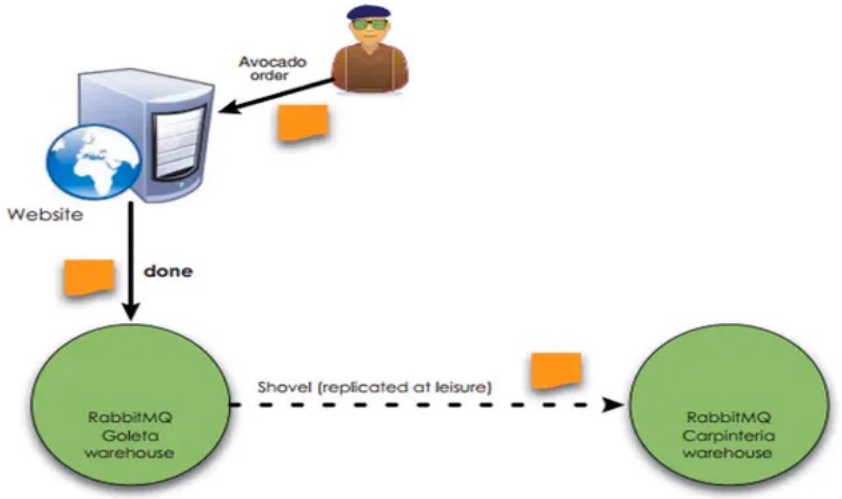


Figure 7.6 Order processing with Shovel

如图所示，有两个异地的 MQ 集群（可以是更多的集群），当用户在地区 1 这里下单了，系统发消息到 1 区的 MQ 服务器，发现 MQ 服务已超过设定的阈值，负载过高，这条消息就会被转到 地区 2 的 MQ 服务器上，由 2 区的去执行后面的业务逻辑，相当于分摊我们的服务压力

在使用了 shovel 插件后，模型变成了近端同步确认，远端异步确认的方式，大大提高了订单确认速度，并且还能保证可靠性

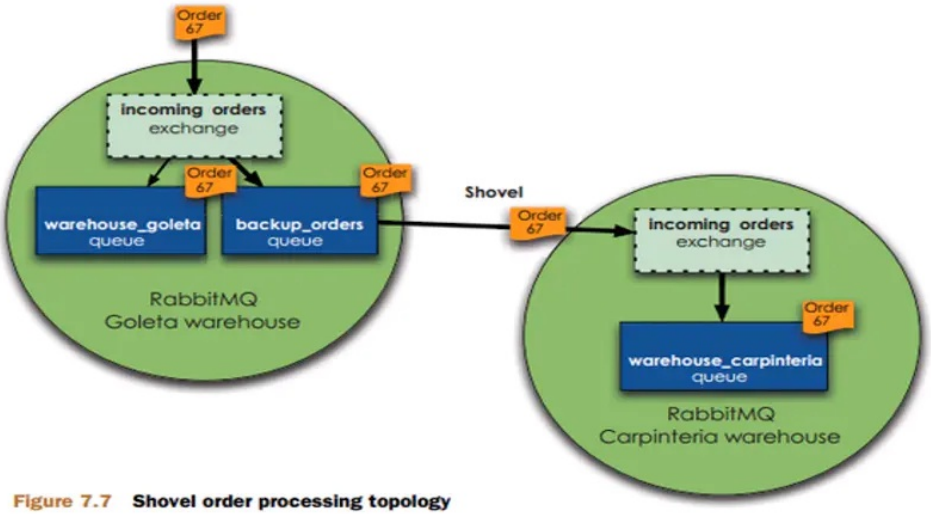


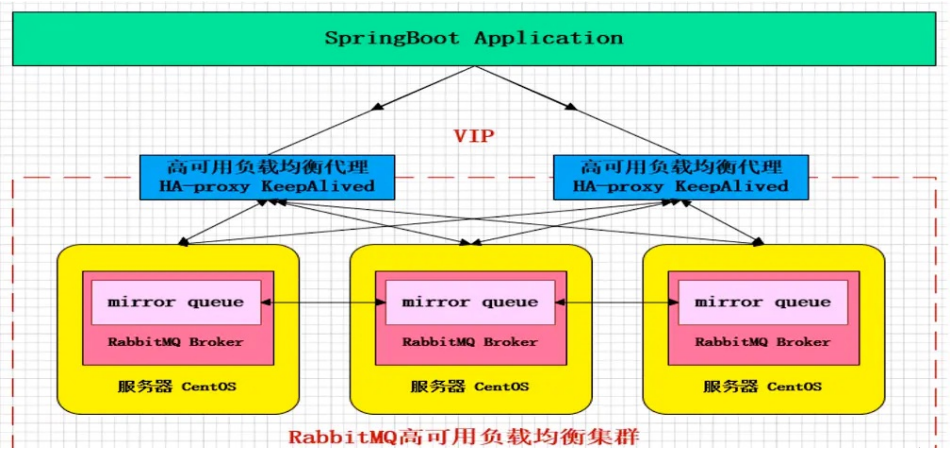
Figure 7.7 Shovel order processing topology

如上图所示，当我们的消息到达 exchange，它会判断当前的负载情况以及设定的阈值，如果负载不高就把消息放到我们正常的 warehouse_goleta 队列中，如果负载过高了，就会放到 backup_orders 队列中。backup_orders 队列通过 shovel 插件与另外的 MQ 集群进行同步数据，把消息发到第二个 MQ 集群上。

3. 远程模式

非常经典的 mirror 镜像模式，保证 100% 数据不丢失。在实际工作中也是用得最多的，并且实现非常的简单，一般互联网大厂都会构建这种镜像集群模式。

mirror 镜像队列，目的是为了保证 rabbitMQ 数据的高可靠性解决方案，主要就是实现数据的同步，一般来讲是 2 - 3 个节点实现数据同步。对于 100% 数据可靠性解决方案，一般是采用 3 个节点。



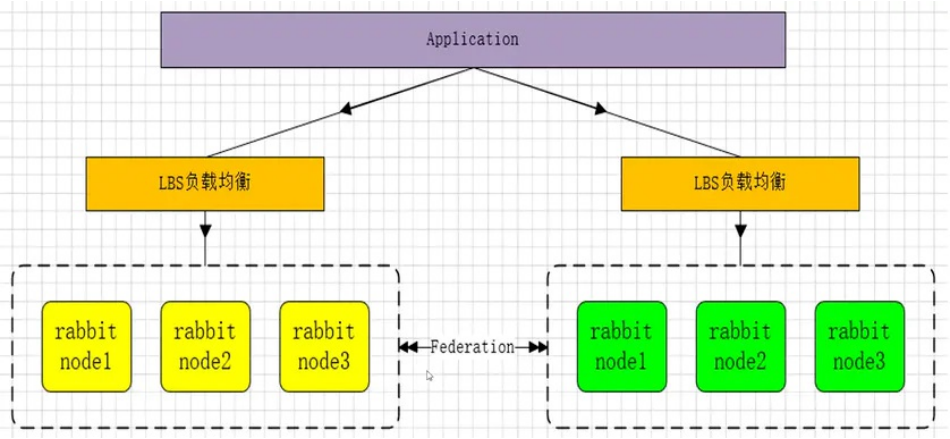
如上图所示，用 KeepAlived 做了 HA-Proxy 的高可用，然后有 3 个节点的 MQ 服务，消息发送到主节点上，主节点通过 mirror 队列把数据同步到其他的 MQ 节点，这样来实现其高可靠。

> 4. 多活模式

也是实现异地数据复制的主流模式，因为 shovel 模式配置比较复杂，所以一般来说，实现异地集群的都是采用这种双活 或者 多活模型来实现的。这种模式需要依赖 rabbitMQ 的 federation 插件，可以实现持续的，可靠的 AMQP 数据通信，多活模式在实际配置与应用非常的简单。

rabbitMQ 部署架构采用双中心模式(多中心)，那么在两套(或多套)数据中心各部署一套 rabbitMQ集群，各中心的rabbitMQ 服务除了需要为业务提供正常的消息服务外，中心之间还需要实现部分队列消息共享。

多活集群架构如下:



> 拉取镜像

```
docker pull rabbitmq:3.7-management
```

备注 alpine系统存在问题

3.2. 构建rabbitmq镜像集群

参考资料

https://blog.51cto.com/11134648/2155934

https://www.jianshu.com/p/5b2879fba25b

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

首先我们可以如下的方式构建

本次课程以3为主，2,4节点均是加入到3集群中

注意统一配置共享目录下的`.erlang.cookie`参数值如下

```
XXGQZIOGOIUETCEJIODM
```

这个参数值，并非固定值，而是基于erlang的cookie规则而编辑的，因此你也可以改为其他，最为重要的就是cookie一致

如下是docker-compose

```
# 编排php,redis,nginx容器
version: "3.6" # 确定docker-composer文件的版本
services: # 代表就是一组服务 - 简单来说一组容器
  # server
  rabbimq_server_172_3: # 这个表示服务的名称，课自定义；注意不是容器名称
    image: rabbitmq:3.7-management # 指定容器的镜像文件
    ports: # 配置容器与宿主机的端口
      - "15673:15672"
      - "5673:5672"
    networks: ## 引入外部预先定义的网段
       rabbimq:
         ipv4_address: 172.200.7.3   #设置ip地址
    hostname: mq3
    volumes:
      - "/www/wwwroot/2007_SRM/rabbimq/3:/var/lib/rabbitmq"
    container_name: rabbimq_server_172_3 # 这是容器的名称
  rabbimq_server_172_2: # 这个表示服务的名称，课自定义；注意不是容器名称
    image: rabbitmq:3.7-management # 指定容器的镜像文件
    ports: # 配置容器与宿主机的端口
      - "15672:15672"
      - "5672:5672"
    networks: ## 引入外部预先定义的网段
       rabbimq:
         ipv4_address: 172.200.7.2   #设置ip地址
    hostname: mq2
    volumes:
      - "/www/wwwroot/2007_SRM/rabbimq/2:/var/lib/rabbitmq"
    container_name: rabbimq_server_172_2 # 这是容器的名称
  rabbimq_server_172_4: # 这个表示服务的名称，课自定义；注意不是容器名称
    image: rabbitmq:3.7-management # 指定容器的镜像文件
    ports: # 配置容器与宿主机的端口
      - "15674:15672"
      - "5674:5672"
    networks: ## 引入外部预先定义的网段
       rabbimq:
         ipv4_address: 172.200.7.4   #设置ip地址
    hostname: mq4
    volumes:
      - "/www/wwwroot/2007_SRM/rabbimq/4:/var/lib/rabbitmq"
    container_name: rabbimq_server_172_4 # 这是容器的名称
# 设置网络模块
```



```
networks:
  # 自定义网络
  rabbimq:
    driver: bridge
    ipam: #定义网段
      config:
        - subnet: "172.200.7.0/24"
```

然后再构建容器 `docker-compose up -d`

```
docker exec -it rabbimq_server_172_2 bash
docker exec -it rabbimq_server_172_4 bash
```

一起修改里面的hosts与hostname

hosts

```
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.200.7.2     mq2
172.200.7.3     mq3
172.200.7.4     mq4
```

hostname

```
mq2
mq3
mq4
```

因为容器关系，不能进行vi，所以只能上传cp修改

```
[root@localhost rabbimq]# docker exec -it rabbimq_server_172_4 bash -c "cp /var/lib/rabbitmq/hosts /etc/hosts"
[root@localhost rabbimq]# docker exec -it rabbimq_server_172_4 bash -c "cp /var/lib/rabbitmq/hostname /etc/hostname"
```

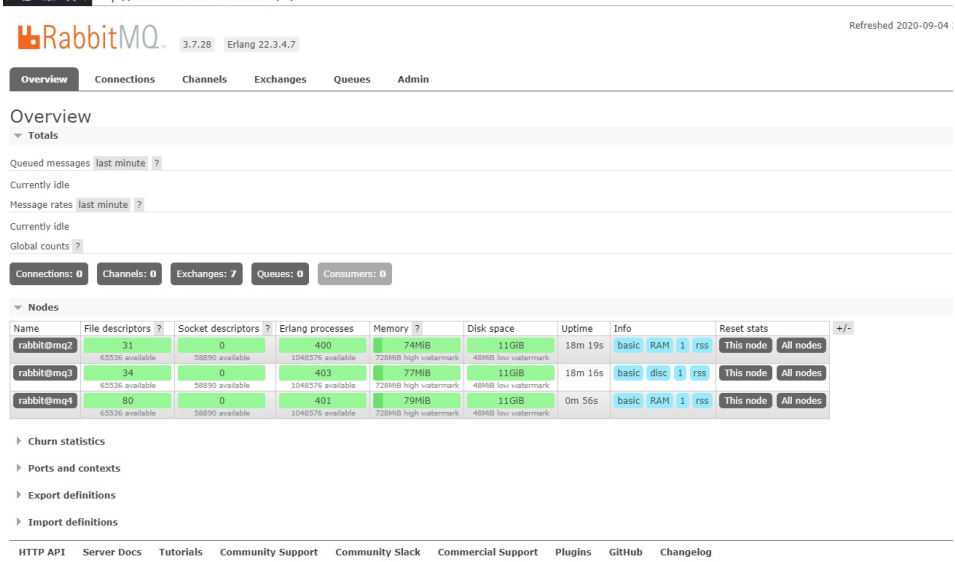
下一步配置rabbitmq在2,4节点中统一执行如下命令

```
root@mq2:/# rabbitmqctl stop_app
```


```
Stopping rabbit application on node rabbit@mq2 ...
root@mq2:/# rabbitmqctl reset
Resetting node rabbit@mq2 ...
root@mq2:/# rabbitmqctl join_cluster --ram rabbit@mq3
Clustering node rabbit@mq2 with rabbit@mq3
root@mq2:/# rabbitmqctl start_app
Starting node rabbit@mq2 ...
 completed with 3 plugins.
```

最后测试

http://192.168.169.110:15673/#/



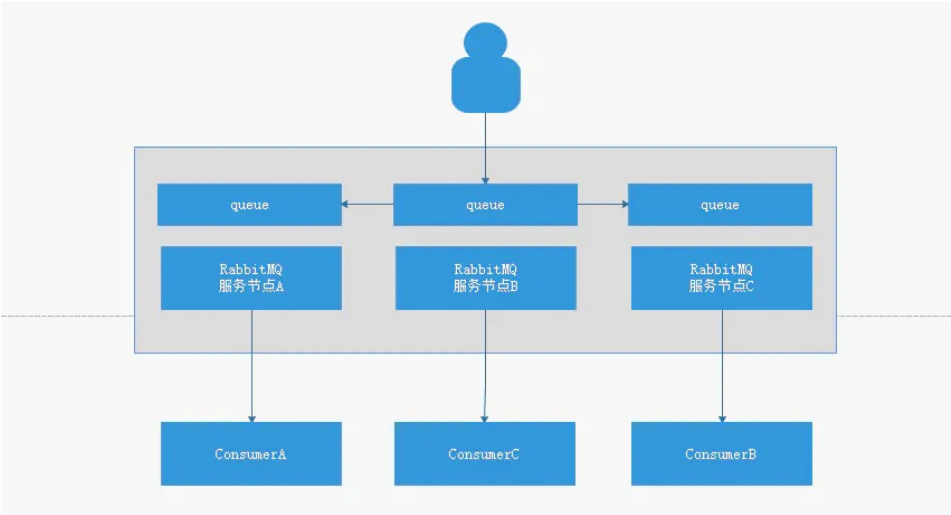
可以看到三个节点均正常运行

最后设置匹配的队列为高可用

```
root@mq4:/#  rabbitmqctl set_policy ha-all "^" '{"ha-mode":"all"}'
Setting policy "ha-all" for pattern "^" to "{"ha-mode":"all"}" with priority "0" for vhost "/" ...
```

注意：rabbitmq如果做高可用可能会性能降低，所以提供这样的功能来区分可以让一些对来不走高可用模式

3.3. 理解rabbitmq镜像集群通信过程



镜像集群模式特点

- 生产者向任一服务节点注册队列，该队列相关信息会同步到其他节点上
- 任一消费者向任一节点请求消费，可以直接获取到消费的消息，因为每个节点上都有相同的实际数据
- 任一节点宕机，不影响消息在其他节点上进行消费

3.4. haproxy实现rabbitmq负载均衡

本次课程中会采用haproxy来实现对rabbitmq的反向代理与负载均衡，当然实际上我们也可以利用nginx来处只是主流均会以haproxy来做

3.4.1. haproxy介绍

1. HAProxy特别适用于那些负载特大的web站点，这些站点通常又需要会话保持或七层处理。HAProxy运行在时下的硬件上，完全可以支持数以万计的 并发连接。并且它的运行模式使得它可以很简单安全的整合进您当前的架构中， 同时可以保护你的web服务器不被暴露到网络上
2. 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。多进程或多线程模型受内存限制 、系统调度器限制以及无处不在的锁限制，很少能处理数千并发连接。事件驱动模型因为在有更好的资源和时间管理的用户端(User-Space) 实现所有这些任务，所以没有这些问题。此模型的弊端是，在多核系统上，这些程序通常扩展性较差。这就是为什么他们必须进行优化以 使每个CPU时间片(Cycle)做更多的工作。
3. HAProxy 支持连接拒绝 : 因为维护一个连接的打开的开销是很低的，有时我们很需要限制***蠕虫（attack bots），也就是说限制它们的连接打开从而限制它们的危害。 这个已经为一个陷于小型DDoS***的网站开发了而且已经拯救了很多站点，这个优点也是其它负载均衡器没有的。
4. HAProxy 支持全透明代理（已具备硬件防火墙的典型特点:可以用客户端IP地址或者任何其他地址来连接后端服务器. 这个特性仅在linux 2.4/2.6内核打了cttproxy补丁后才可以使用. 这个特性也使得为某特殊服务器处理部分流量同时又不修改服务器的地址成为可能。

3.4.2. haproxy与nginx对比

> nginx

1. 工作在网络的7层之上，可以针对http应用做一些分流的策略，比如针对域名、目录结构；
2. Nginx对网络的依赖比较小；
3. Nginx安装和配置比较简单，测试起来比较方便；
4. 也可以承担高的负载压力且稳定，一般能支撑超过1万次的并发；
5. Nginx可以通过端口检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点，不过其中缺点就是不支持url来检测；
6. Nginx对请求的异步处理可以帮助节点服务器减轻负载；
7. Nginx能支持http和Email，这样就在适用范围上面小很多；
8. 不支持Session的保持、对Big request header的支持不是很好，另外默认的只有Round-robin和IP-hash两种负载均衡算法。

> haproxy

1. HAProxy是工作在网络7层之上。

2. 能够补充Nginx的一些缺点比如Session的保持，Cookie的引导等工作
3. 支持url检测后端的服务器出问题的检测会有很好的帮助。
4. 更多的负载均衡策略比如：动态加权轮循(Dynamic Round Robin)，加权源地址哈希(Weighted Source Hash)，加权URL哈希和加权参数哈希(Weighted Parameter Hash)已经实现
5. 单纯从效率上来讲HAProxy更会比Nginx有更出色的负载均衡速度。
6. HAProxy可以对Mysql进行负载均衡，对后端的DB节点进行检测和负载均衡。

3.4.3. 基于haproxy构建rabbitmq反向代理

拉取镜像：

```
docker pull haproxy:1.7-alpine
```

haproxy配置

```
global
  #日志输出配置，所有日志都记录在本机，通过local0输出
  log 127.0.0.1 local0 info
  #最大连接数
  maxconn 10240
  #以守护进程方式运行
  daemon

defaults
  #应用全局的日志配置
  log global
  mode http
  #超时配置
  timeout connect 5000
  timeout client 5000
  timeout server 5000
  timeout check 2000

listen http_front #haproxy的客户页面
  bind 0.0.0.0:8100
  mode http
  option httplog
  stats uri /haproxy
  stats auth root:0000
  stats refresh 5s
  stats enable

listen rabbimq_ha #负载均衡的名字
  bind 0.0.0.0:5600 #对外提供的虚拟的端口
  option tcplog
  mode tcp
```


```
    #轮询算法 循环轮询
    balance roundrobin
    server rabbit1 192.168.169.110:5672 check inter 5000 rise 2 fall 2
    server rabbit2 192.168.169.110:5673 check inter 5000 rise 2 fall 2
    server rabbit3 192.168.169.110:5674 check inter 5000 rise 2 fall 2
```

配置说明

1. global：全局配置参数，进程级的，用来控制Haproxy启动前的一些进程及系统设置
2. defaults：配置一些默认的参数，可以被frontend，backend，listen段继承使用
3. frontend：用来匹配接收客户所请求的域名，uri等，并针对不同的匹配，做不同的请求处理
4. backend：定义后端服务器集群，以及对后端服务器的一些权重、队列、连接数等选项的设置，我将其理解为Nginx中的upstream块
5. listen：我将其理解为frontend和backend的组合体

对应的docker-compose

```
# 编排php,redis,nginx容器
version: "3.6" # 确定docker-composer文件的版本
services: # 代表就是一组服务 - 简单来说一组容器
  # server
  haproxy_server_173_3: # 这个表示服务的名称，课自定义；注意不是容器名称
    image: haproxy:1.7-alpine # 指定容器的镜像文件
    ports: # 配置容器与宿主机的端口
      - "8103:8100"
      - "5603:5600"
    networks: ## 引入外部预先定义的网段
       haproxy:
         ipv4_address: 173.200.7.3   #设置ip地址
    volumes:
      - "/www/wwwroot/2007_SRM/haproxy:/haproxy"
    container_name: haproxy_server_173_3 # 这是容器的名称
    command: haproxy -f /haproxy/haproxy.cfg
  haproxy_server_173_2: # 这个表示服务的名称，课自定义；注意不是容器名称
    image: haproxy:1.7-alpine # 指定容器的镜像文件
    ports: # 配置容器与宿主机的端口
      - "8102:8100"
      - "5602:5600"
    networks: ## 引入外部预先定义的网段
       haproxy:
         ipv4_address: 173.200.7.2   #设置ip地址
    volumes:
      - "/www/wwwroot/2007_SRM/haproxy:/haproxy"
    container_name: haproxy_server_173_2 # 这是容器的名称
    command: haproxy -f /haproxy/haproxy.cfg
# 设置网络模块
networks:
  # 自定义网络
  haproxy:
    driver: bridge
```


```
  ipam: #定义网段
    config:
      - subnet: "173.200.7.0/24"
```

其中 `haproxy -f /haproxy/haproxy.cfg` 是启动haproxy的命令

配置好之后，我们就可以利用访问浏览器选择其中一个haproxy的web监听端口访问可以看到监控的rabbitmq运行状况 `http://ip:8103/haproxy`

Statistics Report for pid 9

> General process information

pid = 9 (process #1, nbproc = 1)
uptime = 0d 2h17m26s
system limits: memmax = unlimited; ulimit-n = 20495
maxsock = 20495; **maxconn =** 10240; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/7; idle = 100 %

- active UP
- active UP, going down
- active DOWN, going up
- active or backup DOWN
- active or backup DOWN for maintenance (MAINT)
- active or backup SOFT STOPPED for maintenance
- backup UP
- backup UP, going down
- backup DOWN, going up
- not checked

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Disable refresh
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.7)
- Online manual

http_front

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtl
Frontend				1	2	-	1	2	2 000	613			193 141	9 321 402	0	0	64					OPEN									
Backend	0	0		0	1		0	1	200	5	0	0s	193 141	9 321 402	0	0		5	0	0	0	2h17m UP		0	0	0		0			

rabbitmq_ha

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtl
Frontend				0	1	-	0	1	2 000	3			1 233	1 704	0	0	0					OPEN									
rabbit1	0	0	-	0	1		0	1	-	1	1	54m14s	411	568		0		0	0	0	0	2h17m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
rabbit2	0	0	-	0	1		0	1	-	1	1	54m5s	411	568		0		0	0	0	0	2h17m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
rabbit3	0	0	-	0	1		0	1	-	1	1	53m18s	411	568		0		0	0	0	0	2h17m UP	L4OK in 0ms	1	Y	-	0	0	0s	-
Backend	0	0		0	1		0	1	200	3	3	53m18s	1 233	1 704	0	0		0	0	0	0	2h17m UP		3	3	0		0	0s	

利用php请求haproxy操作rabbitmq;因为rabbitmq的扩展安装比较麻烦因此采用composer的组件处理

```
  composer require php-amqplib/php-amqplib
```

对应的PHP代码

```
<?php
require "vendor/autoload.php";


use PhpAmqpLib\Connection\AMQPStreamConnection;
use PhpAmqpLib\Message\AMQPMessage;
```

```
$con = new AMQPStreamConnection('192.168.169.110', 5603, 'guest', 'guest');
// var_dump($con);
$re = $con->channel();
var_dump($re);
?>
```

3.5. 实现keepalived对haproxy监控

详细介绍：http://keepalived.readthedocs.io/en/latest/introduction.html

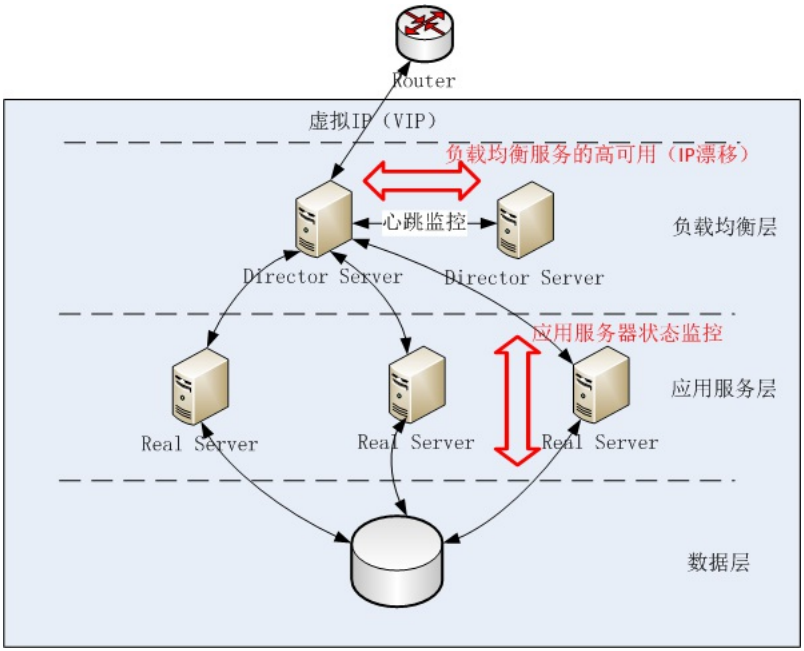
3.5.1. keepalived介绍

1.

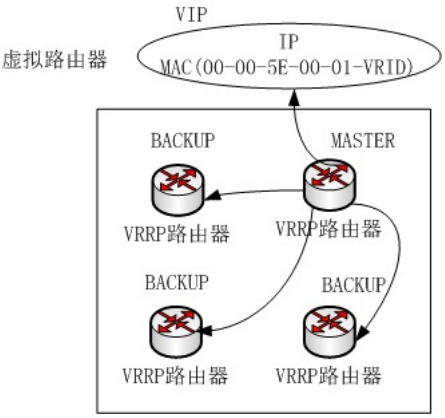
Keepalived是一款高可用软件，它的功能主要包括两方面：

1）通过IP漂移，实现服务的高可用：服务器集群共享一个虚拟IP，同一时间只有一个服务器占有虚拟IP并对外提供服务，若该服务器不可用，则虚拟IP漂移至另一台服务器并对外提供服务

2）对应用服务层的应用服务器集群进行状态监控：若应用服务器不可用，则keepalived将其从集群中摘除，若应用服务器恢复，则keepalived将其重新加入集群中。



Keepalived的实现基于VRRP（Virtual Router Redundancy Protocol，虚拟路由器冗余协议），而VRRP是为了解决静态路由的高可用。VRRP的基本架构如图所示：



虚拟路由器由多个VRRP路由器组成，每个VRRP路由器都有各自的IP和共同的VRID(0-255)，其中一个VRRP路由器通过竞选成为MASTER，占有VIP，对外提供路由服务，其他成为BACKUP，MASTER以IP组播（组播地址：224.0.0.18）形式发送VRRP协议包，与BACKUP保持心跳连接，若MASTER不可用（或BACKUP接收不到VRRP协议包），则BACKUP通过竞选产生新的MASTER并继续对外提供路由服务，从而实现高可用。

> 2.

关于 Keepalived 的详细介绍可以参考：Keepalived Introduction

Keepalived 是一个用于负载均衡和高可用的路由软件。

其负载均衡（Load balancing）的特性依赖于 Linux 虚拟服务器（LVS）的 IPVS 内核模块，提供了 Layer 4 负载均衡器（TCP 层级，Layer 7 是 HTTP 层级，即计算机网络中的OSI 七层网络模型与 TCP/IP 四层网络模型）。

Keepalived 实现了虚拟冗余路由协议（VRRP, Virtual Redundancy Routing Protoco），VRRP 是路由故障切换(failover)的基础。

简单来说，Keepalived 主要提供两种功能：

- 依赖 IPVS 实现服务器的健康检查；
- 实现 VRRPv2 协议来处理路由的故障切换。

我们接下来会用个简单的配置来描述后者的工作原理：

一般的配置

master

```
vrrp_script chk_haproxy {
    script "killall -0 haproxy"  # verify haproxy's pid existance
    interval 2                   # check every 2 seconds
    weight -2                    # if check failed, priority will minus 2
}

vrrp_instance VI_1 {
    state MASTER                 # Start-up default state
    interface ens18              # Binding interface
    virtual_router_id 51         # VRRP VRID(0-255), for distinguish vrrp's multicast
    priority 105                 # VRRP PRIO
    virtual_ipaddress {          # VIP, virtual ip
        192.168.0.146
    }
    track_script {               # Scripts state we monitor
        chk_haproxy
    }
}
```

backup

```
vrrp_script chk_haproxy {
   script "killall -0 haproxy"
    interval 2
    weight -2
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens18
    virtual_router_id 51
    priority 100
    virtual_ipaddress {
        192.168.0.146
    }
    track_script {
        chk_haproxy
    }
}
```

我们为两台机器（master、backup）安装了 Keepalived 服务并设定了上述配置。

可以发现，我们绑定了一个虚拟 IP (VIP, virtual ip): 192.168.0.146，在 haproxy-master + haproxy-backup 上用 Keepalived 组成了一个集群。在集群初始化的时候，

haproxy-master 机器的 `<state>` 被初始化为 MASTER。

间隔 2 seconds(`<interval>`) 会定时执行 `<script>` 脚本，每个 `<vrrp_instance>` 会记录脚本的 exit code。

关于 `<weight>` 参数的使用：

- 检测失败，并且 weight 为正值：无操作
- 检测失败，并且 weight 为负值：priority = priority - abs(weight)
- 检测成功，并且 weight 为正值：priority = priority + weight
- 检测成功，并且 weight 为负值：无操作

weight 默认值为 0，对此如果感到迷惑可以参考：HAProxy github code

故障切换工作流程：

- 当前的 MASTER 节点 `<script>` 脚本检测失败后，如果“当前 MASTER 节点的 priority” < “当前 BACKUP 节点的 priority” 时，会发生路由故障切换。
- 当前的 MASTER 节点脚本检测成功，无论 priority 是大是小，不会做故障切换。

其中有几处地方需要注意：

1. 一个 Keepalived 服务中可以有个 0 个或者多个 vrrp_instance
2. 可以有多个绑定同一个 VIP 的 Keepalived 服务（一主多备），本小节中只是写了两个
3. 注意 `<virtual_router_id>`，同一组 VIP 绑定的多个 Keepalived 服务的 `<virtual_router_id>` 必须相同；多组 VIP 各自绑定的 Keepalived 服务一定与另外组不相同。否则前者会出现丢失节点，后者在初始化的时候会出错。

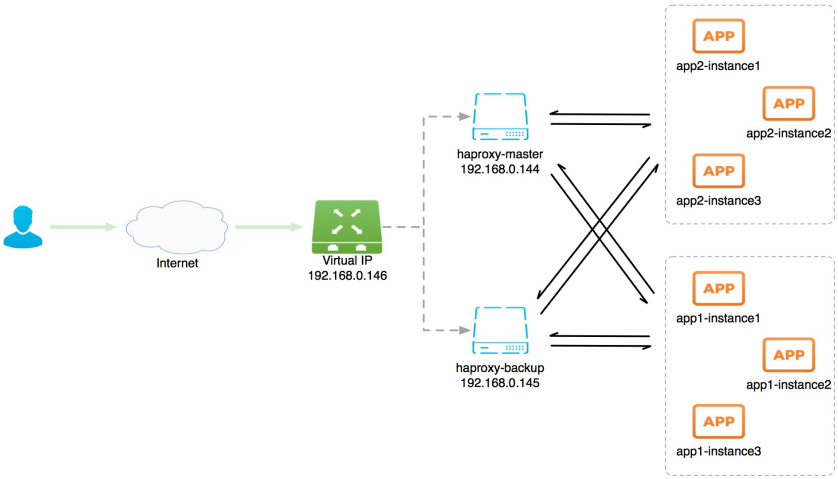


图 3 整体结构图

3.5.2. keepalived构建

因为keepalived是需要安装在对应的容器上才能发挥效果

增加haproxy的dockerfile

```
FROM haproxy:2.3-dev3-alpine
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.aliyun.com/g' /etc/apk/repositories \
  && apk add keepalived
```

修改docker-compose

```
# 编排php,redis,nginx容器
```


```
version: "3.6" # 确定docker-composer文件的版本
services: # 代表就是一组服务 - 简单来说一组容器
  # server
  haproxy_server_173_3: # 这个表示服务的名称，课自定义；注意不是容器名称
    image: haproxy # 指定容器的镜像文件
    ports: # 配置容器与宿主机的端口
      - "8103:8100"
      - "5603:5600"
    networks: ## 引入外部预先定义的网段
       haproxy:
         ipv4_address: 173.200.7.3   #设置ip地址
    privileged: true
    volumes:
      - "/www/wwwroot/2007_SRM/haproxy:/haproxy"
      - "/www/wwwroot/2007_SRM/haproxy/keepalived/3:/keepalived"
    container_name: haproxy_server_173_3 # 这是容器的名称
    command: haproxy -f /haproxy/haproxy.cfg
  haproxy_server_173_2: # 这个表示服务的名称，课自定义；注意不是容器名称
    image: haproxy # 指定容器的镜像文件
    ports: # 配置容器与宿主机的端口
      - "8102:8100"
      - "5602:5600"
    networks: ## 引入外部预先定义的网段
       haproxy:
         ipv4_address: 173.200.7.2   #设置ip地址
    privileged: true
    volumes:
      - "/www/wwwroot/2007_SRM/haproxy:/haproxy"
      - "/www/wwwroot/2007_SRM/haproxy/keepalived/2:/keepalived"
    container_name: haproxy_server_173_2 # 这是容器的名称
    command: haproxy -f /haproxy/haproxy.cfg
# 设置网络模块
networks:
  # 自定义网络
  haproxy:
    driver: bridge
    ipam: #定义网段
      config:
        - subnet: "173.200.7.0/24"
```

对应的 keepalievd 配置 *简化版本*

master

```
  vrrp_instance  VI_1 {
      state  MASTER
      interface  eth0
      virtual_router_id  100
      priority  100
      advert_int  1
```



```
    authentication {
        auth_type  PASS
        auth_pass  0000
    }
    virtual_ipaddress {
        173.200.7.100
    }
}
```

backup

```
vrrp_instance haproxy_ha {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 0000
    }
    virtual_ipaddress {
        173.200.7.100
    }
}
```

启动命令 `keepalievd -f /keepalievd/keepalievd.conf`

但是这儿keepalievd执行并没有生成虚拟ip | 但有时候又可以，暂不清楚是否与当前容器还是docker本身的问题；只能手动执行如下命令增加虚拟ip

```
ifconfig eth0:0 173.200.7.100 netmask 255.255.255.0 up
```

最后因为本身 `173.200.7.100` 是在docker容器内部的无法对外因此这个时候我们需要借助与nginx做对外；实际项目中一般做了集群之后并不会存在这样的情况因为主要是内网相通

需要采用最新的nginx因为新增了stream模块

```
docker pull nginx
```

对应的nginx配置

```
user  nginx;
```



```
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}
# rabbitmq通信
stream{
    server {
        listen 5600;
        proxy_pass 173.200.7.100:5600;
    }
}
```

然后再构建容器即可最后的测试是基于php访问nginx的容器-》haproxy-》mq

keepalived其他参数配置

```
! Configuration File for keepalived

global_defs {
    router_id node4 ##标识节点的字符串，通常为hostname
}

vrrp_script chk_haproxy{
    script "/etc/keepalived/haproxy_check.sh"   ## 执行脚本位置
    interval 2  ##检查时间间隔
    weight -20 ##如果条件成立则权重减20
}

vrrp_instance VI_1 {
    state MASTER##主节点为MASTER,备份节点为BACKUP
    interface ens33 ##绑定虚拟ip的网络接口(网卡)
    virtual_router_id 13    ##虚拟路由id号，主备节点相同
    mcast_src_ip 192.168.174.13 ##本机ip地址
    priority 100    ##优先级(0-254)
    nopreempt
    advert_int 1    ##组播信息发送间隔，两个节点必须一致,默认1s
    authentication {    ##认证匹配
        auth_type PASS
        auth_pass bhz
    }
    track_script {
        chk_haproxy
    }
    virtual_ipaddress {
        192.168.174.70 ##虚拟ip,可以指定多个
```



六星教育

```
    }
}
```



六星教育