容器网络的灵活...

本篇是第七部分“网络篇”的第二篇。在这个部分，我会为你由浅入深的介绍 Docker 网络相关的内容。包括 Docker 网络基础及其实现和内部原理等。上篇，我为你介绍了如何使用用户自定义的 bridge 网络。本篇，我们将学习如何灵活的使用容器网络。

Docker 在网络方面也提供了多种功能，可用于满足不同的需求。本篇，我来为你介绍几种灵活使用 Docker 网络的方法。

域名解析

上篇我为你介绍过，通过 `docker network create 网络名` 以及在启动容器时，通过 `--network 网络名` 可以让容器使用自定义的 bridge 网络。

同时，通过使用这种方式也可以使用 Docker 内置 DNS，以便于可以使用名称来互联容器。

复制

```
(MoeLove) ➜  ~ docker network create -d bridge moelove
35c6f3c23927d3331480f0b365b86b9af4ef6d0e3f05b58be111028d44c66090
(MoeLove) ➜  ~ docker run --rm -d --network moelove --name redis redis:alpine
efdb4eea88feaa3459f6762d250a48d1c61c0cb5264f00dc2607bf22f8a78f3e
(MoeLove) ➜  ~ docker run --rm -it --network moelove alpine sh
/ # ping -c 1 redis
PING redis (172.23.0.2): 56 data bytes
64 bytes from 172.23.0.2: seq=0 ttl=64 time=0.098 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.098/0.098/0.098 ms
```

这种情况是提前预设好，在容器创建时候便指定了其所属的网络。

但如果是已经在运行的容器那该如何处理呢？

将运行中容器加入现有网络

启动一个容器，不使用自定义的 bridge 网络，而是使用默认的网络。

复制

```
(MoeLove) ➜  ~ docker run --rm -d --name db redis:alpine
79976a18c1f93c0100e5c432336c2efaee37a62406897c1dcc5b7606040e0364
```

使用前面自定义的 bridge 网络，启动容器，并尝试连接刚创建的容器：

```
(MoeLove) ➜  ~ docker run --rm --network moelove -it alpine
/ # ping -c 1 db
ping: bad address 'db'
```

可以看到，是无法正常连接的。

Docker 为我们提供了 `docker network connect` 的命令，可用于将运行中容器加入现有网络中。

```
(MoeLove) ➜  ~ docker network connect moelove db
(MoeLove) ➜  ~ docker run --rm --network moelove -it alpine
/ # ping -c 1 db
PING db (172.23.0.3): 56 data bytes
64 bytes from 172.23.0.3: seq=0 ttl=64 time=0.124 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.124/0.124/0.124 ms
```

当加入成功后，便可继续使用 Docker 内置的 DNS 提供域名解析能力了。

当然，也可以使用 `docker network disconnect` 将某个容器从网络中摘除：

```
(MoeLove) ➜  ~ docker network disconnect moelove db
(MoeLove) ➜  ~ docker run --rm --network moelove -it alpine
/ # ping -c 1 db
ping: bad address 'db'
```

提供别名

在启动容器时候，可以通过提供 `--network-alias` 的参数为容器设置一个不同于其名称的别名，可用于域名解析。

这个功能在实际部署应用时会非常有用。为容器设置带有特定规则的名称，用于区分不同的业务线或不同作用。通过提供别名，可保持业务代码不变，仍然可以正常连接对应的容器。

复制

```
(MoeLove) ➜  ~ docker run --network moelove --rm -d --name new-redis --network-a
9aa7b2e0bb09fbee67be18ff00c8daffff6f7141daf71930f541fd988337c20d
(MoeLove) ➜  ~ docker run --rm -it --network moelove alpine
/ # ping -q -c 1 new-redis
PING new-redis (172.23.0.3): 56 data bytes

--- new-redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.135/0.135/0.135 ms
/ # ping -q -c 1 new-db
PING new-db (172.23.0.3): 56 data bytes

--- new-db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.079/0.079/0.079 ms
```

除了上述方法外，通过设置 hostname 也是可以达到类似效果的。例如：

复制

```
(MoeLove) ➜  ~ docker run --network moelove --rm -d --name moelove-redis --hostna
4698b62e9dcf90261123ea799626bbee364b0759048f6ccaab27f1f91a8f6c0c
(MoeLove) ➜  ~ docker run --rm -it --network moelove alpine
/ # ping -q -c1 moelove
PING moelove (172.23.0.5): 56 data bytes

--- moelove ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.117/0.117/0.117 ms
/ # ping -q -c1 moelove-redis
PING moelove-redis (172.23.0.5): 56 data bytes

--- moelove-redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.095/0.095/0.095 ms
/ # ping -q -c1 moelove-db
PING moelove-db (172.23.0.5): 56 data bytes

--- moelove-db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.078/0.078/0.078 ms
/ # ping -q -c1 moelove-cache
PING moelove-cache (172.23.0.5): 56 data bytes

--- moelove-cache ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.100/0.100/0.100 ms
```

也可以通过以下命令直接进行查看：

复制

```
(MoeLove) ➜  ~ docker inspect --format "{{ .NetworkSettings.Networks.moelove.Ali
[moelove-db moelove-cache 4698b62e9dcf moelove]
```

总结

本篇，我为你介绍了几种方法可以通过这样来使用到 Docker 内置的 DNS 以实现容器之间的互联（或者说“服务发现”）。

这些能力都是基于使用自定义网络提供的，在实际使用中，自定义 bridge 网络，相比默认的 bridge 网络可以提供更好的隔离性，也带来了更多的便利性，推荐使用。

下篇，我将为你介绍 Docker 与 iptables 之间的联系，分析为何 Docker 能为我们提供如此方便的网络能力。