容器生命周期管...

容器生命周期管理（下）

> 这是本专栏的第二部分：容器篇，共 6 篇，帮助大家由浅入深地认识和掌握容器。上一篇和本篇，我会为你介绍容器生命周期管理相关的内容，带你掌握容器生命周期。下面我们一起进入第二篇的内容，主要涉及容器状态的变化。

删除容器

上一篇已经介绍了容器的创建，暂停以及停止等操作，并且一般情况下即使容器被 stop 掉它仍然存在于机器上，通过 `docker ps -a` 仍然可以查看到其记录，并且可随时通过 `docker start` 将其从 Exited 的状态更换为 Running。

那如何可以删除掉一个不需要的已经停止的容器呢？

使用 `docker rm` 即可。

仍然以一个实际例子入手：

复制

```
(MoeLove) ➜  ~ docker run -d redis
516b4ac5d642e9766e40f5b41e4b8bbf3ebfcfcc8bd134dfea94a533678e8800
# 查询容器状态
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
516b4ac5d642        redis               "docker-entrypoint.s…"   5 seconds ago
```

先停止该容器，然后对它做删除操作：

复制

```
# 停止该容器
(MoeLove) → ~ docker stop $(docker ps -ql)
do516b4ac5d642
# 查询状态已经为 Exited
(MoeLove) → ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
516b4ac5d642        redis               "docker-entrypoint.s…"   2 minutes ago
# 删除容器
(MoeLove) → ~ docker rm 516b4ac5d642
516b4ac5d642
# 验证查询无结果
(MoeLove) → ~ docker ps -a |grep 516b4ac5d642
(MoeLove) → ~
```

是否还有其他方式呢？

有，可以在 `docker run` 或者 `docker create` 时，传递 `--rm` 的选项，以便在容器退出时可自动删除。

复制

```
(MoeLove) → ~ docker create --rm redis
998381d619b2e105043b169a2abb635d3ad8594c68fff52cbf5decb38496d80b
(MoeLove) → ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
998381d619b2        redis               "docker-entrypoint.s…"   4 seconds ago
(MoeLove) → ~ docker start $(docker ps -ql)
998381d619b2
(MoeLove) → ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
998381d619b2        redis               "docker-entrypoint.s…"   21 seconds ago
(MoeLove) → ~ docker stop $(docker ps -ql)
998381d619b2
(MoeLove) → ~ docker ps -a |grep 998381d619b2
```

除此之外还有更方便的方式吗？

有，可以用 `docker container prune` 直接批量删除全部已经停止的容器。

复制

```
(MoeLove) → ~ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
```

当输入 `docker container prune` 后，会有个提示，输入 y 确认，则开始删除全部已停止的容器。

源码中的容器状态

前面的内容都是通过实践，让你体验到了如何管理容器的生命周期，现在我们深入源码来看看在源码中容器的状态。

关于容器状态相关的代码都在 container/state.go。

复制

```
func (s *State) StateString() string {
    if s.Running {
        if s.Paused {
            return "paused"
        }
        if s.Restarting {
            return "restarting"
        }
        return "running"
    }

    if s.RemovalInProgress {
        return "removing"
    }

    if s.Dead {
        return "dead"
    }

    if s.StartedAt.IsZero() {
        return "created"
    }

    return "exited"
}
```

可以看到其中的 Created、Running、Paused、Exited 等状态，在前面的实验中都已经接触过了，没有涉及到的状态包括 Dead、 Removing 和 Restarting，其实 Removing 我们也算接触过了，前面的 `docker rm` 执行过程中便会将容器状态设置为 Removing，只不过过程比较快，所以不容易捕获到。

至于 Restarting 状态，是在 `docker run` 或者 `docker create` 可通过传递 `--restart` 参数来设置重启策略，在容器执行异常，或是 Docker 后台进程重启之类的情况下，会按照重启策略进行重启。

最后就是 Dead 状态了，这个状态多数时候是个中间状态，比如要删除容器的时候，会将其状态设置为 Dead，但是**当删除过程失败，则容器会保持 Dead 状态了。**

总结

本篇介绍了容器的删除操作，以及通过源码展示了容器几种状态的关系：Paused 和 Restarting 时，容器的实际表现其实是在 Running 的。

下图展示了容器各种状态间的切换关系，希望能对你有所帮助。