容器的核心：Na...

这是本专栏的第二部分：容器篇，共 8 篇，帮助大家由浅入深地认识和掌握容器。前面，我为你介绍了容器生命周期和资源管理相关的内容，让你对容器有了更加灵活的控制。之后从进程的角度带你认识了容器以及容器核心技术之一的 cgroups。本篇，我来为介绍容器的另一项核心技术：namespace。

什么是 namespace

我们仍然以 Wiki 上对 namespace 的定义开始：

> Namespaces are a feature of the Linux kernel that partitions kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources. The feature works by having the same namespace for these resources in the various sets of processes, but those names referring to distinct resources.

namespace 是 Linux 内核的一项功能，它可以对内核资源进行分区，使得一组进程可以看到一组资源；而另一组进程可以看到另一组不同的资源。该功能是在各种进程集中对这些资源使用相同的 namespace，但是这些名称引用不同的资源。

这样的说法未免太绕了些，简单来说 namespace 是由 Linux 内核提供的，用于进程间资源隔离的一种技术。同时 Linux 也默认提供了多种 namespace，用于对多种不同资源进行隔离。

那么，到目前为止 Linux 有哪些 namespace 可用呢？

namespace 名称	系统调用参数	含义
Mount	CLONE_NEWNS	隔离挂载点
PID	CLONE_NEWPID	进程编号
Network	CLONE_NEWNET	网络设备，堆栈，端口等
IPC	CLONE_NEWIPC	系统 IPC, POSIX 消息队列
UTS	CLONE_NEWUTS	系统主机名和 NIS(Network Information Service) 主机名（有时称为域名）
User	CLONE_NEWUSER	用户和组 ID
Cgroup	CLONE_NEWCGROUP	Cgroup 根目录

根据上面的介绍，我们也看出来了，每种 namespace 各有自己管理的部分；对于现在 Linux 上的任意进程而言，它一定是每种 namespace 的一个实例，我们来实际看下：

复制

```
(MoeLove) ➜  ~ ls -l --time-style='+' /proc/self/ns
total 0
lrwxrwxrwx. 1 tao tao 0  cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 tao tao 0  ipc -> 'ipc:[4026531839]'
lrwxrwxrwx. 1 tao tao 0  mnt -> 'mnt:[4026531840]'
lrwxrwxrwx. 1 tao tao 0  net -> 'net:[4026531992]'
lrwxrwxrwx. 1 tao tao 0  pid -> 'pid:[4026531836]'
lrwxrwxrwx. 1 tao tao 0  pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx. 1 tao tao 0  user -> 'user:[4026531837]'
lrwxrwxrwx. 1 tao tao 0  uts -> 'uts:[4026531838]'
```

在 /proc/$PID 的目录下，有个 ns 文件夹，这里面的文件基本是以 namespace 的命名的，并且它们都是一些链接。（此目录首次出现在《深入剖析容器》那一篇，如果没有印象了可以回头看下）

如何使用 namespace

前面提到了 Linux 内核提供了以上 7 种 namespace，那如何去使用它们呢？Linux 提供了以下主要的 API 用于管理 namespace：

- clone()：如果是纯粹只使用 clone()，则会创建一个新进程；但如果我们传递一个或多个 CLONE_NEW* 标志给 clone()，则会根据每个标志创建对应的新 namespace 并且将子进程添加为其成员。
- setns()：允许进程加入一个已存在的 namespace 中。
- unshare()：允许进程（或线程）取消其执行上下文中，与其他进程（或线程）共享部分的关联，当然通俗点来说，也就是可以利用此系统调用来让当前的进程（或线程）移动至一个新的 namespace 中。

以上提到的这三个 API 或者说系统调用，看起来都很简单，但实际上它们可接收的参数很多，其意义也各有不同。不过这不是本篇的重点，我们暂且忽略。

验证 namespace 的作用

以上内容过于偏理论了，接下来我们以实际的操作来感受下 namespace 的作用。这里我们使用上面介绍过的 unshare() API 来进行 namespace 的隔离。

Linux 中包含了一组标准的工具包，名为 util-linux，其实我们经常会使用这组工具包，比如 kill 便是其中之一。

在这组工具包中还包含着很多其他有用的工具，其中一个名为 unshare 的工具，便通过封装 unshare() 这个 API 以便让我们以更直接的方式来使用它，而无需自己再单独写代码实现了。

隔离 PID namespace

通过给 unshare 传递 `--pid`、`--fork` 和 `--mount-proc` 参数来实现隔离 PID namespace：

复制

```
(MoeLove) ➜  ~ sudo unshare --pid --fork --mount-proc zsh
# 使用 ps -a 来验证
[root@bogon]/home/tao# ps -a
  PID TTY          TIME CMD
    1 pts/20   00:00:00 zsh
   28 pts/20   00:00:00 ps
```

PS：我的默认 Shell 是 zsh，所以我这里使用了 zsh，当然你也可以换成任何你喜欢的 Shell 或者其他程序/命令。

我们通过 `ps -a` 来查看当前全部的进程，可以发现只能看到 PID 为 1 的 zsh 进程，和我们执行的 ps 的进程。

通过这里我们就可以确认，已经与主机上的进程编号进行了隔离（因为主机上在已经存在 PID 为 1 的 init 进程了）。

为了加深印象，这里我们再次从 /proc 中读取下实际的信息来确认一次：

复制

```
# 在隔离 PID namespace 的环境中
[root@bogon]/home/tao# ls -l --time-style='+' /proc/self/ns
总用量 0
lrwxrwxrwx. 1 root root 0  cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 root root 0  ipc -> 'ipc:[4026531839]'
lrwxrwxrwx. 1 root root 0  mnt -> 'mnt:[4026532853]'
lrwxrwxrwx. 1 root root 0  net -> 'net:[4026531992]'
lrwxrwxrwx. 1 root root 0  pid -> 'pid:[4026532854]'
lrwxrwxrwx. 1 root root 0  pid_for_children -> 'pid:[4026532854]'
lrwxrwxrwx. 1 root root 0  user -> 'user:[4026531837]'
lrwxrwxrwx. 1 root root 0  uts -> 'uts:[4026531838]'
```

复制

```
# 在主机上
(MoeLove) ➜  ~ ls -l --time-style='+' /proc/self/ns
总用量 0
lrwxrwxrwx. 1 tao tao 0  cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 tao tao 0  ipc -> 'ipc:[4026531839]'
lrwxrwxrwx. 1 tao tao 0  mnt -> 'mnt:[4026531840]'
lrwxrwxrwx. 1 tao tao 0  net -> 'net:[4026531992]'
lrwxrwxrwx. 1 tao tao 0  pid -> 'pid:[4026531836]'
lrwxrwxrwx. 1 tao tao 0  pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx. 1 tao tao 0  user -> 'user:[4026531837]'
lrwxrwxrwx. 1 tao tao 0  uts -> 'uts:[4026531838]'
```

可以看到两者的 /proc/self/ns/pid 是不同的，再次确认了我们的想法。

隔离 UTS namespace

我们再来尝试对 UTS namespace 的隔离。前面已经介绍过了，这个 namespace 主要影响的是 hostname 相关的部分。当进行 UTS namespace 隔离时，隔离环境中修改 hostname 将不会影响主机上原本的 hostname。

可以通过给 unshare 传递 `-u` 参数来实现 UTS namespace 的隔离。

复制

```
(MoeLove) ➜  ~ sudo unshare -u zsh
[root@bogon]/home/tao# hostname
bogon
[root@bogon]/home/tao# hostname utsns
[root@bogon]/home/tao# hostname
utsns
```

现在新打开一个创建查看主机的 hostname：

复制

```
(MoeLove) ➜  ~ hostname
bogon
```

可以看到**在 UTS namespace 隔离环境中修改 hostname 并没有影响到主机。**

我们来看看隔离环境下的 namespace 信息，先通过 ps 命令查看下当前的进程：

复制

```
[root@bogon]/home/tao# ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root     13456 13923  0 00:37 pts/28   00:00:00 ps -f
root     13854 11123  0 00:08 pts/28   00:00:00 sudo unshare -u zsh
root     13923 13854  0 00:08 pts/28   00:00:00 zsh
```

我们没有指定隔离 PID namespace，所以这里的 PID 就是主机上的 PID。

由于我们当前是在 zsh 这个 shell 环境下，所以我们可以直接对比 13923（zsh）和 13854（sudo unshare -u zsh）这两个进程的 namespace 信息。

`sudo unshare -u zsh` 进程的 namespace 信息：

复制

```
(MoeLove) ➜  ~ sudo ls -l --time-style='+' /proc/13854/ns
总用量 0
lrwxrwxrwx. 1 root root 0  cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 root root 0  ipc -> 'ipc:[4026531839]'
lrwxrwxrwx. 1 root root 0  mnt -> 'mnt:[4026531840]'
lrwxrwxrwx. 1 root root 0  net -> 'net:[4026531992]'
lrwxrwxrwx. 1 root root 0  pid -> 'pid:[4026531836]'
lrwxrwxrwx. 1 root root 0  pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx. 1 root root 0  user -> 'user:[4026531837]'
lrwxrwxrwx. 1 root root 0  uts -> 'uts:[4026531838]'
```

zsh 进程的 namespace 信息：

复制

```
(MoeLove) ➜  ~ sudo ls -l --time-style='+' /proc/13923/ns
总用量 0
lrwxrwxrwx. 1 root root 0  cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 root root 0  ipc -> 'ipc:[4026531839]'
lrwxrwxrwx. 1 root root 0  mnt -> 'mnt:[4026531840]'
lrwxrwxrwx. 1 root root 0  net -> 'net:[4026531992]'
lrwxrwxrwx. 1 root root 0  pid -> 'pid:[4026531836]'
lrwxrwxrwx. 1 root root 0  pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx. 1 root root 0  user -> 'user:[4026531837]'
lrwxrwxrwx. 1 root root 0  uts -> 'uts:[4026532678]'
```

可以看到两者仅有 UTS namespace 信息是不同的。

前面的内容已经验证了如何通过 unshare 隔离 PID 和 UTS namespace 的操作，限于篇幅原因，你可以自行验证下如何隔离其他几类 namespace。

在 Docker 中如何利用 namespaces

我们先按一般情况使用 Docker 启动一个容器，并查看其进程信息：

```
(MoeLove) ➜  ~ docker run --rm -it alpine
/ # ps -f
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh
    6 root      0:00 ps -f
```

可以看到，当前容器与主机上的 PID namespace 是隔离的（它的 PID 从 1 开始），我们也可以从主机上直接查看容器的 namespace 信息：

```
(MoeLove) ➜  ~ sudo ls -l --time-style='+' /proc/$(docker inspect --format '{{.S
总用量 0
lrwxrwxrwx. 1 root root 0  cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 root root 0  ipc -> 'ipc:[4026532691]'
lrwxrwxrwx. 1 root root 0  mnt -> 'mnt:[4026532687]'
lrwxrwxrwx. 1 root root 0  net -> 'net:[4026532694]'
lrwxrwxrwx. 1 root root 0  pid -> 'pid:[4026532692]'
lrwxrwxrwx. 1 root root 0  pid_for_children -> 'pid:[4026532692]'
lrwxrwxrwx. 1 root root 0  user -> 'user:[4026531837]'
lrwxrwxrwx. 1 root root 0  uts -> 'uts:[4026532689]'
```

作为对比，再次列出主机的 namespace 信息：

```
(MoeLove) ➜  ~ sudo ls -l --time-style='+' /proc/$$/ns
总用量 0
lrwxrwxrwx. 1 tao tao 0  cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 tao tao 0  ipc -> 'ipc:[4026531839]'
lrwxrwxrwx. 1 tao tao 0  mnt -> 'mnt:[4026531840]'
lrwxrwxrwx. 1 tao tao 0  net -> 'net:[4026531992]'
lrwxrwxrwx. 1 tao tao 0  pid -> 'pid:[4026531836]'
lrwxrwxrwx. 1 tao tao 0  pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx. 1 tao tao 0  user -> 'user:[4026531837]'
lrwxrwxrwx. 1 tao tao 0  uts -> 'uts:[4026531838]'
```

可以看到容器与主机的 namespace 是隔离的。

那我们有没有办法启动容器，但是不与主机进行 PID namespace 的隔离？

答案是有，Docker 给我们提供了很强大的功能，允许我们在 `docker run` 的时候，通过一些参数来控制容器的 namespace。比如，**通过传递 `--pid host` 即可与主机共享 PID namespace。**

复制

```
(MoeLove) ➜  ~ docker run --rm -it --pid host alpine
/ # ls -l /proc/$$/ns
total 0
lrwxrwxrwx    1 root     root             0 Nov  2 00:39 cgroup -> cgroup:[4026531
lrwxrwxrwx    1 root     root             0 Nov  2 00:39 ipc -> ipc:[4026532787]
lrwxrwxrwx    1 root     root             0 Nov  2 00:39 mnt -> mnt:[4026532785]
lrwxrwxrwx    1 root     root             0 Nov  2 00:39 net -> net:[4026532789]
lrwxrwxrwx    1 root     root             0 Nov  2 00:39 pid -> pid:[4026531836]
lrwxrwxrwx    1 root     root             0 Nov  2 00:39 pid_for_children -> pid:[
lrwxrwxrwx    1 root     root             0 Nov  2 00:39 user -> user:[4026531837]
lrwxrwxrwx    1 root     root             0 Nov  2 00:39 uts -> uts:[4026532786]
```

对比我们前面看到的主机上的 namespace 信息，可以看到它与主机的 PID namespace 是完全相同的。

我们是否还可以给容器指定其他 PID namespace 呢？答案是有的，我们可以从源码中得到确认：

复制

```
# api/types/container/host_config.go#L246-L259
func (n PidMode) Valid() bool {
    parts := strings.Split(string(n), ":")
    switch mode := parts[0]; mode {
    case "", "host":
    case "container":
        if len(parts) != 2 || parts[1] == "" {
            return false
        }
    default:
        return false
    }
    return true
}
```

另一种模式便是给 `--pid` 传递 `container:容器 ID`，以便与其他容器共享 namespace。

先正常启动一个容器：

复制

```
(MoeLove) ➜  ~ docker run --rm -it alpine
/ # ls -l /proc/$$/ns/pid
lrwxrwxrwx    1 root     root             0 Nov  2 00:51 /proc/1/ns/pid -> pid:[40
```

与刚才的容器进行 PID namespace 共享：

复制

```
(MoeLove) ➜  ~ docker run --rm -it --pid container:$(docker ps -ql) alpine
/ # ls -l /proc/$$/ns/pid
lrwxrwxrwx    1 root     root             0 Nov  2 18:51 /proc/7/ns/pid -> pid:[40
```

可以看到两者的 PID namespace 是相同的了。

篇幅原因，这里只介绍对 PID namespace 相关的操作，其实对于其他的 namespace，Docker 也提供了相应的能力。除 PID namespace 外，一般场景下，可能会用到的就是 Network namespace 了，这部分相关的知识，我们会在后续内容的“网络篇” 中另行介绍。

总结

在本篇中，我为你介绍了 namespace，不过我没有过多地去介绍如何利用 Linux 内核提供的系统调用去进行开发。我选择了直接使用 Linux 提供的 unshare 工具，我希望这样可以便于你直观的感受。

namespace 为 Docker 提供了各类资源隔离的功能，而这些功能也不仅限于在 Docker 中使用。在实际的生产中，我们也可以利用 namespace 相关的能力为我们提供更多便利。

通过上篇及本篇，我已经为你介绍了 Docker 进行容器资源管理和隔离所需的最主要的两项核心技术。下一篇，我们进入实践环节，自己动手来写容器。