容器的核心：Cgr...

> 这是本专栏的第二部分：容器篇，共 6 篇，帮助大家由浅入深的认识和掌握容器。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的本质还是一组进程。本篇，我来为介绍容器的核心——cgroups。

什么是 cgroups

先引用一句 Wiki 上对 cgroups 的定义：

> cgroups，其名称源自控制组群（control groups）的简写，是 Linux 内核的一个功能，用来限制、控制与分离一个进程组的资源（如 CPU、内存、磁盘输入输出等）。

这句定义虽然比较抽象，但却也道出了 cgroups 的主要功能，限制、控制与分离一个进程组的资源。

这也是我们在上篇中从进程的角度分析容器时，得出容器应该具备的属性：“具备相同特性，也受相同限制的进程组”，它们所受的控制恰好就是由 cgroups 来完成的。

你可能会想知道，cgroups 可以控制哪些资源呢?

- CPU
- 内存
- IO
- 网络

这几类资源是我们平时会比较关注，cgroups 都可以控制，但在本文中，我们重点只看我们之前介绍过的两类：CPU 和内存。

注：一般我们将 cgroups 的技术实现称之为 cgroup，对其整体会统称为 cgroups；为避免术语混用带来的理解成本，本文中会统一使用 cgroups 进行描述。

使用 cgroups 控制 CPU 资源

在之前内容中，我们通过 `--cpus` 或 `--cpuset-cpus` 的方式，来限制容器可使用的 CPU 资源，或是将容器限定只可以在固定的 CPU 上进行调度。这恰好就是针对 CPU 资源最通用的两种控制方式了。

Docker 对容器资源的这种控制能力，实际是通过 cgroups 完成的，在对 cgroups 进行更细致介绍前，我们直接来看看 Docker 这种对 CPU 控制的方式，在 cgroups 上的具体表现是什么。

复制

```
(MoeLove) ➜  ~ docker run --rm -it --cpus "1.5" --cpuset-cpus 0,1 alpine
/ #
```

限制容器只能在前两个 CPU 核上调度，同时限制容器只能使用 1.5 核。

在容器内，查看容器的根进程（PID 为 1 的进程）的 cgroups 相关信息，位置在 /proc/1/cgroup（这个文件是从 Linux 2.6.24 开始加入内核的）：

复制

```
/ # cat /proc/1/cgroup
11:perf_event:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d7056
10:freezer:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705654d
9:blkio:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705654dd46
8:devices:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705654dd
7:net_cls,net_prio:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679
6:pids:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705654dd463
5:cpu,cpuacct:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d7056
4:cpuset:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705654dd4
3:hugetlb:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705654dd
2:memory:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705654dd4
1:name=systemd:/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705
0::/system.slice/docker-749ff2b1fdecdb372423b8fc2f989c3f0a0e400679d705654dd46351eb
```

这个文件我们之前看到过，但没有对它做太深入的介绍，现在我们既然有了前面内容的积累，现在就来对此文件做下介绍。

这个文件以 `:` 分割成为三列，从左到右都有其具体含义。

第一列：表示 cgroups 中的层次结构

对于 v1 版本，这里是个层次结构的唯一索引。而对于 cgroups v2 这里则是零值。

这里的重点分两部分，一个是**层次结构的唯一索引**。这个索引怎么找呢？可以在 /proc/cgroups 中直接查看。可以看到与我们上方的内容是完全对应的。

复制

```
# 为了方便查看，我对输出内容做了格式化
/ # cat /proc/cgroups
#subsys_name    hierarchy       num_cgroups     enabled
cpuset          4               52              1
cpu             5               158             1
cpuacct         5               158             1
blkio           9               158             1
memory          2               557             1
devices         8               158             1
freezer         10              52              1
net_cls         7               52              1
perf_event      11              52              1
net_prio        7               52              1
hugetlb         3               52              1
pids            6               171             1
```

第二个重点，你可以比较困惑，我如何知道自己使用的是 v1 版本还是 v2 版本呢？

在宿主机上查看是否有 cgroups v2 的支持：

复制

```
(MoeLove) ➜  ~ grep cgroup /proc/filesystems
nodev   cgroup
nodev   cgroup2
```

从上面输出可以看到是同时有 v1 和 v2 的。（当然，如果你测试没有看到 cgroup2 那说明你的内核并不支持 cgroups v2）

查看具体哪些位置使用了哪个版本：

复制

```
(MoeLove) ➜  ~ mount |grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,seclabel,mode=755)
cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,se
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,sec
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,secl
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,sec
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,secl
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,seclabe
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,rel
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,sec
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,seclab
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,sec
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,s
```

type 是 cgroup2 的，则表示使用了 v2，中间那个路径表示挂载点，而最后的内容则表示权限和一些其他属性。

第二列：对 cgroups v1 而言，这里是以逗号分割的控制器的名称，而对于 v2 版本这列则为空。

第三列：表示进程所属的层次结构中控制组的路径名称，当然需要注意的是，这个路径名称是相对于 cgroups 的挂载点而言的。

关于第三列这样解释后，想必你会明白在之前内容中我们查看资源限制的命令的含义了。这里再来说明一次，顺便也做个复习，比如，当我们想要查看 cgroups 对容器的 CPU 资源的限制时，使用 `mount |grep cgroup | grep cpu` 找到与 CPU 资源限制相关的挂载点：

复制

```
(MoeLove) ➜  ~ mount |grep cgroup|grep cpu
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,secl
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,
```

将此挂载点和刚才在容器内执行 `cat /proc/1/cgroup` 得到的内容的第三列的路径结合起来，便可以得到在宿主机上的真实 cgroups 配置信息了。

复制

```
(MoeLove) ➜  ~ ls /sys/fs/cgroup/cpu,cpuacct/system.slice/docker-749ff2b1fdecdb37
cgroup.clone_children  cpuacct.stat   cpuacct.usage_all     cpuacct.usage_percpu_s
cgroup.procs           cpuacct.usage  cpuacct.usage_percpu  cpuacct.usage_percpu_u
```

查看其中与 Linux 内核的 CPU 调度 CFS 相关的两个文件内的信息：

复制

```
(MoeLove) ➜  ~ cat /sys/fs/cgroup/cpu,cpuacct/system.slice/docker-749ff2b1fdecdb3
100000
(MoeLove) ➜  ~ cat /sys/fs/cgroup/cpu,cpuacct/system.slice/docker-749ff2b1fdecdb3
150000
```

可以使用的 CPU 资源量是由 `cpu.cfs_quota_us/cpu.cfs_period_us` 得到的，也就是 150000/100000=1.5，与我们的配置相符。

接下来以同样的方式来验证可使用的 CPU 核心的信息：

复制

```
(MoeLove) ➜  ~ cat /sys/fs/cgroup/cpuset/system.slice/docker-749ff2b1fdecdb372423
0-1
```

可以看到与我们的配置都是符合的。

留个课后作业吧：

> 在宿主机上验证 Docker 给容器设置的内存限制是否应用到了 cgroups 上。

扩展

前面只说了对 CPU 资源的限制，其实 cgroups 能做的也不只是这个。还记得在容器生命周期管理中，我们提到的暂停容器吗？ `docker pause` 命令，其实它的功能也是通过 cgroups 完成的。

这里，我们暂停刚才启动的容器：

复制

```
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
749ff2b1fdec        alpine              "/bin/sh"           About an hour ago   Up
(MoeLove) ➜  ~ docker pause $(docker ps -ql)
749ff2b1fdec
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
749ff2b1fdec        alpine              "/bin/sh"           About an hour ago   Up
```

查看 cgroups 的 freezer 挂载点上的信息：

复制

```
(MoeLove) ➜  ~ cat /sys/fs/cgroup/freezer/system.slice/docker-749ff2b1fdecdb37242
FROZEN
```

可以看到当前是 FROZEN 冻结的状态。接下来，我们将它恢复，并查看相关信息：

复制

```
(MoeLove) ➜  ~ docker unpause $(docker ps -ql)
749ff2b1fdec
(MoeLove) ➜  ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
749ff2b1fdec        alpine              "/bin/sh"           About an hour ago   Up
(MoeLove) ➜  ~ cat /sys/fs/cgroup/freezer/system.slice/docker-749ff2b1fdecdb37242
THAWED
```

可以看到完全一致。

总结

在本篇中，我为你介绍了 cgroups，不过我没有给你去介绍 cgroups 的历史或者 v1 和 v2 之前的改造升级之类的信息。我选择直接从 Docker 容器的资源限制入手，我希望这样可以便于你的理解。

cgroups 涉及的内容，对于使用 Docker 来说，并不算特别多。但本篇中的内容如果你能全部掌握，对于你在生产中使用 Docker 或者对 Docker 进行管理会大有帮助。

另外，cgroups 只是 Docker 容器核心技术的一部分，主要在资源管理相关的部分。下一篇，我会为你介绍另一项核心技术 namespace，让你了解容器的隔离是如何做到的。