一、cgroup是什么?

控制族群（control group），简称 cgroup或cgroups。cgroup可为系统中所运行任务（进程）的用户定义组群分配资源，比如 CPU 时间、系统内存、网络和IO带宽或者这些资源的组合。

> By using cgroups, system administrators gain fine-grained control over allocating, prioritizing, denying, managing, and monitoring system resources. Hardware resources can be appropriately divided up among tasks and users, increasing overall efficiency.（使用cgroup，系统管理员可更具体地控制对系统资源的分配、优先顺序、拒绝、管理和监控。可更好地根据任务和用户分配硬件资源，提高总体效率。）

使用 cgroup，系统管理员可更具体地控制对系统资源的分配、优先顺序、拒绝、管理和监控。可更好地根据任务和用户分配硬件资源，提高总体效率。

二、cgroup v1示例

看个实例：

```
# 找到cgroup的挂载点
[root@localhost ~]# mount | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,seclabel,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/s
ystemd-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/hugetlb type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,hugetlb)
cgroup on /sys/fs/cgroup/rdma type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,rdma)
cgroup on /sys/fs/cgroup/memory type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,memory)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,net_cls,net_prio)
cgroup on /sys/fs/cgroup/freezer type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,freezer)
cgroup on /sys/fs/cgroup/pids type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,pids)
```

```
cgroup on /sys/fs/cgroup/cpuset type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpuset)
cgroup on /sys/fs/cgroup/devices type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,devices)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)

# 查看虚拟文件系统下的文件
[root@localhost ~]# cd /sys/fs/cgroup/
[root@localhost cgroup]# ls -l
drwxr-xr-x.  2 root root  40 12月 15 02:45 blkio
lrwxrwxrwx.  1 root root  11 12月 15 02:45 cpu -> cpu,cpuacct
lrwxrwxrwx.  1 root root  11 12月 15 02:45 cpuacct -> cpu,cpuacct
drwxr-xr-x.  2 root root  40 12月 15 02:45 cpu,cpuacct
dr-xr-xr-x.  2 root root   0 12月 15 02:45 cpuset
dr-xr-xr-x.  3 root root   0 12月 15 02:45 devices
dr-xr-xr-x.  2 root root   0 12月 15 02:45 freezer
dr-xr-xr-x.  2 root root   0 12月 15 02:45 hugetlb
dr-xr-xr-x.  6 root root   0 12月 15 02:45 memory
lrwxrwxrwx.  1 root root  16 12月 15 02:45 net_cls -> net_cls,net_prio
dr-xr-xr-x.  2 root root   0 12月 15 02:45 net_cls,net_prio
lrwxrwxrwx.  1 root root  16 12月 15 02:45 net_prio -> net_cls,net_prio
dr-xr-xr-x.  2 root root   0 12月 15 02:45 perf_event
dr-xr-xr-x.  6 root root   0 12月 15 02:45 pids
dr-xr-xr-x.  2 root root   0 12月 15 02:45 rdma
dr-xr-xr-x.  6 root root   0 12月 15 02:45 systemd

# 查看cpu子系统，注意顶层下有个“test”目录，这是我创建的子层。
[root@localhost cpu]# ls -l
-rw-r--r--.  1 root root   0 12月 15 06:07 cgroup.clone_children
-rw-r--r--.  1 root root   0 12月 15 06:03 cgroup.procs
-r--r--r--.  1 root root   0 12月 15 06:07 cgroup.sane_behavior
-r--r--r--.  1 root root   0 12月 15 06:07 cpuacct.stat
-rw-r--r--.  1 root root   0 12月 15 06:07 cpuacct.usage
-r--r--r--.  1 root root   0 12月 15 06:07 cpuacct.usage_all
-r--r--r--.  1 root root   0 12月 15 06:07 cpuacct.usage_percpu
-r--r--r--.  1 root root   0 12月 15 06:07 cpuacct.usage_percpu_sys
-r--r--r--.  1 root root   0 12月 15 06:07 cpuacct.usage_percpu_user
-r--r--r--.  1 root root   0 12月 15 06:07 cpuacct.usage_sys
-r--r--r--.  1 root root   0 12月 15 06:07 cpuacct.usage_user
-rw-r--r--.  1 root root   0 12月 15 06:07 cpu.cfs_period_us
-rw-r--r--.  1 root root   0 12月 15 06:07 cpu.cfs_quota_us
-rw-r--r--.  1 root root   0 12月 15 06:07 cpu.rt_period_us
-rw-r--r--.  1 root root   0 12月 15 06:07 cpu.rt_runtime_us
-rw-r--r--.  1 root root   0 12月 15 06:07 cpu.shares
-r--r--r--.  1 root root   0 12月 15 06:07 cpu.stat
-rw-r--r--.  1 root root   0 12月 15 06:07 notify_on_release
-rw-r--r--.  1 root root   0 12月 15 06:07 release_agent
-rw-r--r--.  1 root root   0 12月 15 06:07 tasks
drwxr-xr-x. 2 yz   yz   0 12月 15 06:10 test

# 切换到test组
[root@localhost cpu]# cd test/

# 查看CPU使用率限制：
[root@localhost test]# cat cpu.cfs_period_us
100000
[root@localhost test]# cat cpu.cfs_quota_us
```

```
-1

# 假设这个系统上只有一个CPU核。修改CPU使用率限制为50%。
[root@localhost test]# echo 50000 > cpu.cfs_quota_us
[root@localhost test]# cat cpu.cfs_quota_us
50000

# 将被限制进程的PID加入cgroup.procs，这样进程45992的CPU使用率将被限制为不超过50%。
[root@localhost test]# echo 45992 > cgroup.procs
[root@localhost test]# cat cgroup.procs
45992
```

要使用cgroup，Linux内核版本需为`2.5.32`及更高版本。

三、cgroup v1的问题

- v1实现较早，功能比较多，但是由于功能都是零零散散的实现的，所以规划的不是很好，导致了一些使用和维护上的不便。一些bug内核不再进行修复了。
- v1的设计并没有前瞻性，因此后面引入了大量的怪异特性和不一致性。
- memory的`swappiness`实现有缺陷，导致子级组始终使用默认值，而默认情况下，用户进程是由子级组控制的。详见参考文献[1]。
- 已知在高版本内核的Linux发行版中（已知CentOS 8、openEuler 20.03~23.03等），cgroup v1的blkio和device存在bug：blkio和device子系统下的用户层级在某些情况下会被系统删除。详见参考文献[2, 3]。
- 不支持缓存IO限制。

cgroup v2的重要特性

- v2在设计上进行了统一规划。
- 在4.5内核中，cgroup v2声称已经可以用于生产环境了，但所支持的功能还很有限。
- v1和v2可以混合使用，但是这样会更复杂，所以一般没人会这样用。
- v2的子系统层级以及接口发生了重大变更，与v1不兼容，需要应用程序自行适配。
- v2支持缓存IO限制。

四、CentOS 8启用cgroup v2

建议内核版本4.15或更高。

相对来说，CentOS 8切换到cgroup v2很容易。CentOS 7虽然也可以切换到cgroup v2，但是比较麻烦，需要更换内核、更换systemd，而systemd相关组件也需要更换，且有不稳定的风险，不建议生产环境上使用。

以下仅给出CentOS 8的切换方法。

```
[root@bogon ~]# vim /etc/default/grub
GRUB_CMDLINE_LINUX添加"cgroup_no_v1=all systemd.unified_cgroup_hierarchy=1"
[root@bogon ~]# reboot
```

- `cgroup_no_v1=all`是为了关闭v1和v2的混合模式。有三种模式可选：
  - `Legacy（遗留的）`：即仅使用cgroup v1，是为了向前兼容。
  - `Unified（统一的）`：完全使用cgroup v2。
  - `Hybrid（混合的）`：混合模式，即cgroup v1和v2同时使用。
- `systemd.unified_cgroup_hierarchy=1`表示启用cgroup v2。

所以，以上配置后，CentOS 8将完全使用cgroup v2资源管理功能。

三种模式的设置方法：

unified模式：

```
GRUB_CMDLINE_LINUX="systemd.unified_cgroup_hierarchy=1
systemd.legacy_systemd_cgroup_controller=0 cgroup_no_v1=all"
```

legacy模式：

```
GRUB_CMDLINE_LINUX="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller=1"
```

hybrid模式：

```
GRUB_CMDLINE_LINUX="systemd.unified_cgroup_hierarchy=1
systemd.legacy_systemd_cgroup_controller=1"
```

五、cgroup v2层级概览

先看看cgroup v2层级结构：

```
# 找到cgroup v2的挂载点
[root@bogon ~]# mount -l | grep cgroup2
cgroup2 on /sys/fs/cgroup type cgroup2
(rw,nosuid,nodev,noexec,relatime,seclabel,nsdelegate)

# 查看虚拟文件系统下的文件，目录结构与v1完全不同。test是我创建的子级。
[root@localhost ~]# cd /sys/fs/cgroup
[root@localhost cgroup]# ls -l
-r--r--r--. 1 root root 0 12月 15 07:38 cgroup.controllers
-rw-r--r--. 1 root root 0 12月 15 07:23 cgroup.max.depth
-rw-r--r--. 1 root root 0 12月 15 07:23 cgroup.max.descendants
-rw-r--r--. 1 root root 0 12月 15 07:23 cgroup.procs
-r--r--r--. 1 root root 0 12月 15 07:23 cgroup.stat
-rw-r--r--. 1 root root 0 12月 15 07:23 cgroup.subtree_control
-rw-r--r--. 1 root root 0 12月 15 07:23 cgroup.threads
-rw-r--r--. 1 root root 0 12月 15 07:23 cpu.pressure
-r--r--r--. 1 root root 0 12月 15 07:23 cpu.stat
-rw-r--r--. 1 root root 0 12月 15 07:23 io.pressure
-rw-r--r--. 1 root root 0 12月 15 07:23 memory.pressure
drwxr-xr-x. 2 yz   yz   0 12月 15 07:39 test
```

查看当前层级的全部可用子控制器（`cgroup.controllers`）：

```
[root@localhost cgroup]# cat cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

- 子层的`cgroup.controllers`必须是父层的`cgroup.controllers`的子集。
- 子层的`cgroup.controllers`由父层的`cgroup.subtree_control`确定。
- 子层不会自动继承父层的可用控制器。

看个例子：

```
[root@localhost cgroup]# pwd
/sys/fs/cgroup

# 父级的可用控制器：
[root@localhost cgroup]# cat cgroup.controllers
cpuset cpu io memory hugetlb pids rdma

# 当前还没有为子层配置控制器，所以子层test现在没有任何可用控制器：
[root@localhost cgroup]# cat cgroup.subtree_control
（输出为空）
[root@localhost cgroup]# cat test/cgroup.controllers
（输出为空）

# 我们为子层配置CPU控制器，这样子层就有了CPU控制器：
[root@localhost cgroup]# echo "+cpu" > cgroup.subtree_control
[root@localhost cgroup]# cat cgroup.subtree_control
cpu
# 子层test现在可以配置CPU相关的限制了。
[root@localhost cgroup]# cat test/cgroup.controllers
cpu

# 子层不会自动继承父层的可用控制器。
[root@localhost cgroup]# cat test/cgroup.subtree_control
（输出为空）
```

六、cgroup v2使用注意事项

6.1 接口特性

控制器的添加与删除，以最后一次的操作为准：

```
[root@localhost cgroup2]# echo "+cpu -cpu +memory +cpu -cpu -cpu -cpu" >
cgroup.subtree_control
[root@localhost cgroup2]# cat cgroup.subtree_control
（输出为空）
```

一次操作是一次“事务”，如果操作错误，将回滚本次操作：

```
[root@localhost cgroup2]# echo "+cpu +mm" > cgroup.subtree_control
-bash: echo: write error: Invalid argument
[root@localhost cgroup2]# cat cgroup.subtree_control
（输出为空）
```

6.2 theaded模式

v1版本中，将进程PID加入`cgroup.procs`从而控制进程的资源使用，而将线程的LWPID（Light Weight Process ID）加入`tasks`从而控制线程的资源使用。进程和线程可以同时控制。

到v2版本，就没有那么方便了，在任意非顶层的层级下都有一个`cgroup.type`，用于选择是进程还是线程模式，两者不能同时使用，其中，

- domain：对应默认的进程模式。
- domain threaded：线程域资源组，作为线程域子树。

- domain invalid：无效状态的cgroup。这种状态下无法被迁移或者启用控制器。可以变成一个线程域。
- threaded：对应线程模式。

更麻烦的是，模式选择改成theaded模式是单向不可逆的，也就是说，一个资源组一旦修改为theaded模式就无法再改为进程模式了。对应用层来说十分不友好。

6.3 无法使用CPU控制器

在CentOS 8中，当使用cgroup v1和v2混合模式时，我们发现CPU控制器无法被使用：

```
[root@localhost ~]# mkdir /mnt/cgroup2
[root@localhost ~]# mount -t cgroup2 cgroup2 /mnt/cgroup2/

[root@localhost ~]# umount /sys/fs/cgroup/cpu
[root@localhost ~]# cat /mnt/cgroup2/cgroup.controllers
cpu
[root@localhost ~]# cat /mnt/cgroup2/cgroup.subtree_control
（输出为空）
[root@localhost ~]# echo "+cpu" > /mnt/cgroup2/cgroup.subtree_control
-bash: echo: write error: Invalid argument
```

原因是，当前系统中有实时进程存在，无法将cpu控制器从cgroup v1切换到v2，需要先停止实时进程，详见参考文献[6]。

6.4 避免频繁在 cgroup 之间迁移进程

在 cgroup 之间迁移进程是一个开销相对较高的操作，而且 有状态资源（例如 memory）是不会随着进程一起迁移的。因此，不建议为了达到某种资源限制目的而频繁地在 cgroup 之间迁移进程。

七、cgroup v2的坑

7.1 cgroup.procs权限

非root用户想实际使用cgroup v2，首先面临的问题就是`cgroup.procs`权限问题，即使用户有权限的层级，也没有写`cgroup.procs`的权限，需要将根层级（root）的`cgroup.procs`设置为所有用户写权限。

7.2 没有与v1对应的net_cls和net_prio

cgroup v2没有与v1版的`net_cls`和`net_prio`相对应的控制器。

> Instead, support was added in iptables to allow BPF filters that hook on cgroup v2 pathnames to allow control of NW traffic on a per-cgroup basis (Since Linux 4.5(?))

请参阅文献[6]。

7.3 cgroup v2 iptables的bug

内核的bug，iptables使用`cgroup --path`参数时，路径识别错误。

请参阅文献[7]。

八、参考

1. system.slice swappiness is inconsistent with vm.swappiness sysctl
2. daemon-reload or daemon-reexec obliterates custom subdirectories within blkio controller
3. systemctl daemon-reload removes cgroups under devices and blkio
4. Why, after unmounting cgroup v1, do I still have empty directories under /sys/fs/cgroup?
5. The cpu controller in cgroup v2 can not be used in Red Hat Enterprise Linux 8
6. What is new in control groups v2
7. Iptables cgroup V2 marking with systemd