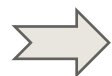


# Linux cgroup v2开发者参考

杨铸 • 东方金信 • 2025.05



- 概述
- *cgroup v1*的缺陷
- *cgroup v1 vs v2*
- *cgroup v2*
- 更多.....

## ● 概述 – 术语

### **cgroup = cgroups (控制组)**

- <https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html#terminology>

### **cgroup = cgroup v1**

- “cgroup”默认指cgroup第一版
- “cgroup v2”特指cgroup第二版

### **Subsystem = Controller**

- v1中的资源叫做子系统，v2中叫做控制器。

### **cgroup v1已停止维护**

- cgroup v2是大势所趋
- 已知bug官方不再修复

### **cgroup v2不是cgroup v1的升级版**

- cgroup v2不兼容cgroup v1，是全新设计。

# ● 概述 – cgroup v2现状

## 为了解决cgroup v1已知问题

- cgroups v1的控制器设计缺乏协调

## 官方Linux内核4.5正式发布

- 官方声称可以用于生产环境，但所支持的功能还很有限。
- 建议内核5.2以上使用cgroup v2。

## cgroup v1与v2共存

- 两个版本在内核中共存，可以选择使用其中之一，也可以混合使用。
- 但是，同一个控制器只能用于其中一个版本。

## cgroup v2权威文档

- <https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html>
- man cgroups



## ● 概述 – 启用cgroup v2的Linux

### 以下Linux发行版默认启用cgroup v2

- Fedora (since 31)
- Arch Linux (since April 2021)
- openSUSE Tumbleweed (since c. 2021)
- Debian GNU/Linux (since 11)
- Ubuntu (since 21.10)
- RHEL and RHEL-like distributions (since 9)

# ● 概述 – CentOS 8.5启用cgroup v2

## 内核要求

- 最小内核版本为4.15，推荐5.2或更新。
- 最小systemd版本是239。

## 修改启动参数

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

[root@bogon ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
[root@bogon ~]# reboot
```

## 验证是否切换

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

# ● 概述 – CentOS 8.5启用cgroup v2

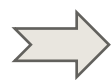
## 参数说明

- `cgroup_no_v1=all`: 是为了关闭v1和v2的混合模式。
- `systemd.unified_cgroup_hierarchy=1`: 表示启用cgroup v2。

## 三种模式可选

- Legacy (遗留的): 仅使用cgroup v1, 为了向前兼容。  
`systemd.unified_cgroup_hierarchy=0 systemd.legacy_systemd_cgroup_controller=1`
- Unified (统一的): 完全使用cgroup v2。  
`systemd.unified_cgroup_hierarchy=1 systemd.legacy_systemd_cgroup_controller=0 cgroup_no_v1=all`
- Hybrid (混合的): 混合模式, cgroup v1和v2同时使用。  
`systemd.unified_cgroup_hierarchy=1 systemd.legacy_systemd_cgroup_controller=1`

- *概述*



- **cgroup v1 的缺陷**

- *cgroup v1 vs v2*

- *cgroup v2*

- *更多.....*





## ● cgroup v1的缺陷 – 设计缺陷

### 功能零散

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内核已经放弃修复。

### 设计没有前瞻性

后续开发引入了大量的怪异特性和不一致性。

### 不支持带缓存的IO限制

实际使用中，带缓存IO限制方式应用场景很少。

## ● cgroup v1的缺陷 – Bug

### blkio和device子系统丢失

在CentOS 8、openEuler 20.03~23.03版本操作系统上，blkio和device子系统下的用户层级在某些情况下会被系统删除。

- [Red Hat Bugzilla – Bug 2174599](#)

### swappiness不生效

使用sysctl修改swappiness，很多情况下不会生效，因为大部分系统中，用户进程的memory cgroup层级都不是在memory子系统的根级，修改并不会影响cgroup子级配置。

- [system.slice swappiness is inconsistent with vm.swappiness sysctl](#)

# ● cgroup v1的缺陷 – 管理混乱

## v1的子系统功能实现各种不一致

- 实现反向引导设计。
- 有些子系统的子组属性继承自父组，而有些直接使用默认值。
- 有些子系统在根组有特殊属性。
- 最大值有的用-1表示，有的用“max”表示。
  - *pids.max*用“max”表示最大。
  - *memory.limit\_in\_bytes*显示值9223372036854771712表示不限制。
  - *memory.limit\_in\_bytes*写入-1表示不限制。

## v2控制器统一规则

- 文档指导实现

# ● cgroup v1的缺陷 – 释放通知

## 术语

- **释放通知** (Release Notification) : v1的术语, 等同于v2的闲置通知。
- **入驻通知** (Populated Notification) : v2控制组从没有进程到有进程迁入时的状态变化通知。
- **闲置通知** (Unpopulated Notification) : v2控制组从有进程到最后一个进程迁出时的状态变化通知。

## v1的释放通知机制有缺陷

- 每次释放通知都会启动一个进程, 代价太大。
- 在同一个控制器的层级中, 不同子树不能有不同的释放代理。
- 不能将释放通知委托给容器内的进程处理。
- 逃逸漏洞: [CVE-2022-0492](#)

## v2提供一种支持代理的轻量级解决方案

- 概述
- *cgroup v1*的缺陷
- ➡ ● **cgroup v1 vs v2**
- *cgroup v2*
- 更多.....

# ● cgroup v1 vs v2 – 层级对比

## v1 (截自: Red Hat Linux 7.6)

```
[root@bogon cgroup]# pwd
/sys/fs/cgroup
[root@bogon cgroup]# ls -al
total 0
drwxr-xr-x 13 root root 340 Sep  3 09:40 .
drwxr-xr-x  6 root root  60 Sep  6 09:32 ..
drwxr-xr-x  6 root root  60 Sep  3 09:40 blkio
lrwxrwxrwx  1 root root 11 Sep  3 09:40 cpu -> cpu,cpuacct
lrwxrwxrwx  1 root root 11 Sep  3 09:40 cpuacct -> cpu,cpuacct
drwxr-xr-x  8 root root  60 Sep  3 09:40 cpu,cpuacct
drwxr-xr-x  5 root root  60 Sep  3 09:40 cpuset
drwxr-xr-x  5 root root  60 Sep  3 09:40 devices
drwxr-xr-x  2 root root  60 Sep  3 09:40 freezer
drwxr-xr-x  2 root root  60 Sep  3 09:40 hugetlb
drwxr-xr-x  9 root root  60 Sep  3 09:40 memory
lrwxrwxrwx  1 root root 16 Sep  3 09:40 net_cls -> net_cls,net_prio
drwxr-xr-x  2 root root  60 Sep  3 09:40 net_cls,net_prio
lrwxrwxrwx  1 root root 16 Sep  3 09:40 net_prio -> net_cls,net_prio
drwxr-xr-x  4 root root  60 Sep  3 09:40 perf_event
drwxr-xr-x  5 root root  60 Sep  3 09:40 pids
drwxr-xr-x  5 root root  60 Sep  3 09:40 systemd
[root@bogon cgroup]# cat /proc/1/cgroup
11:devices:/
10:memory:/
9:blkio:/
8:hugetlb:/
7:pids:/
6:cpuacct,cpu:/
5:freezer:/
4:perf_event:/
3:net_prio,net_cls:/
2:cpuset:/
1:name=systemd:/
```

## v2 (源自: Ubuntu 22.04.4)

```
[root@bogon cgroup]# pwd
/sys/fs/cgroup
[root@bogon cgroup]# ls -al
总用量 0
dr-xr-xr-x. 11 root root 0  9月  4 15:50 .
drwxr-xr-x.  8 root root 0  9月  3 15:22 ..
-r--r--r--.  1 root root 0  9月  3 15:22 cgroup.controllers
-rw-r--r--.  1 root root 0  9月  3 15:22 cgroup.max.depth
-rw-r--r--.  1 root root 0  9月  3 15:22 cgroup.max.descendants
-rw-rw-rw-.  1 root root 0  9月  3 15:22 cgroup.procs
-r--r--r--.  1 root root 0  9月  3 15:22 cgroup.stat
-rw-r--r--.  1 root root 0  9月  3 15:23 cgroup.subtree_control
-rw-r--r--.  1 root root 0  9月  3 15:22 cgroup.threads
-r--r--r--.  1 root root 0  9月  3 15:22 cpuset.cpus.effective
-r--r--r--.  1 root root 0  9月  3 15:22 cpuset.mems.effective
-r--r--r--.  1 root root 0  9月  3 15:22 cpu.stat
drwxr-xr-x.  2 root root 0  9月  3 15:23 dev-hugepages.mount
drwxr-xr-x.  2 root root 0  9月  3 15:23 dev-mqueue.mount
drwxr-xr-x.  2 root root 0  9月  3 15:22 init.scope
-r--r--r--.  1 root root 0  9月  3 15:22 io.stat
-r--r--r--.  1 root root 0  9月  3 15:22 memory.numa_stat
--w-----.  1 root root 0  9月  3 15:22 memory.reclaim
-r--r--r--.  1 root root 0  9月  3 15:22 memory.stat
drwxr-xr-x.  2 root root 0  9月  3 15:23 sys-fs-fuse-connections.mount
drwxr-xr-x.  2 root root 0  9月  3 15:23 sys-kernel-config.mount
drwxr-xr-x.  2 root root 0  9月  3 15:23 sys-kernel-debug.mount
drwxr-xr-x.  2 root root 0  9月  3 15:23 sys-kernel-tracing.mount
drwxr-xr-x. 24 root root 0  9月  4 15:49 system.slice
drwxr-xr-x.  2 root root 0  9月  3 15:23 user.slice
[root@bogon cgroup]# cat /proc/1/cgroup
0::/init.scope
```

# ● cgroup v1 vs v2 – 层级

## v1与v2层级结构优缺点对比

- v1: 每个子系统有自己的层级结构 (下称“多层级”)
- v2: 多个控制器使用同一层级结构 (下称“单层级”)

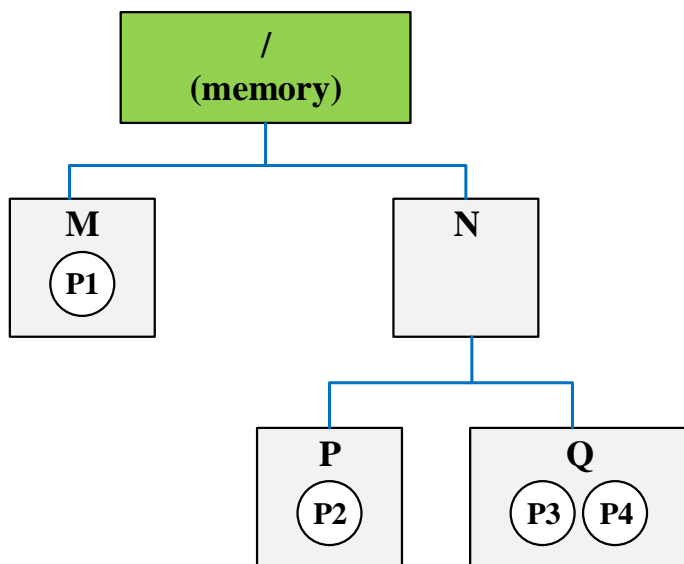
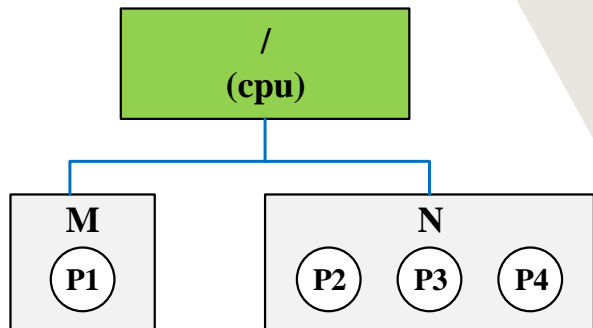
## v1的多层级结构

- 控制器之下分多层级。
- 非常灵活。
- 但是，并没有最初设想的那么实用。
- 这种层次结构有利也有弊，控制器之间缺乏协作。

## v2的单层级结构

- 为所有资源类型建立公共域，使控制器能够相互协作。
- 提供机制允许层次结构中的每个控制器粒度。
- 单层级也有局限性。

## ● cgroup v1 vs v2 – 层级实例



v1多层次

### ■ 优点:

控制器附加到单层级结构意味着可以以不同的粒度管理进程

- 内存方面，P2和P3+P4分成两个资源组。
- CPU方面，P2+P3+P4共享同一个资源组。

### ■ 缺点:

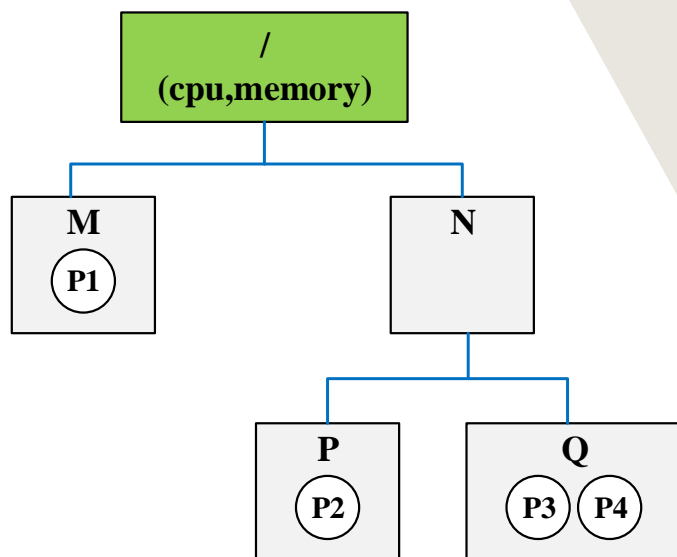
进程在控制组间迁移时，笨拙，性能差，非原子。



## ● cgroup v1 vs v2 – 层级实例

### ■ 优点：

多个控制器放在同一个层级中，就不需要在多个层级中复制移动操作。



v2单层级

### ■ 缺点：

控制器必须以相同的粒度分组管理资源。

- 如左图所示， $P2 + P3 + P4$ 不能再共享一个CPU分配。
- $P2$ 和 $P3 + P4$ ，CPU和内存两种资源的分组策略必须相同。

## ● cgroup v1 vs v2 – v1层级的典型问题

**有些子系统对所有层级都起作用，但实际上仅使其一。  
以freezer为例：**

- 欲冻结一个cpu组的所有进程，则必需要有相同结构的freezer组。
- 类似地，欲冻结一个memory组，也需要有相同结构的freezer组。

**多层还是单层？一直存在争议。**

- 因为各有优缺点。

# ● cgroup v1 vs v2 – 线程粒度

## **v1 允许进程和线程混合粒度的控制组**

- 导致内核管理困难
- 线程粒度对某些子系统（如memory）无意义

## **v2 线程粒度条件更严苛**

- 最初设计仅支持进程粒度
- 目前有条件地支持线程粒度控制

## **v2 是否支持线程粒度的对峙**

- v2 的开发者的坚持：“仅支持进程粒度”。
- 内核维护者则表示：“我们不可能合并不支持线程粒度的v2”。

## **妥协**

- Linux 4.14 之后支持线程粒度，但是附加了严格约束条件。

# ● cgroup v1 vs v2 – 进程管理

## **v1 允许父子层级同时管理进程**

- 父子层级资源限制存在逻辑矛盾
- 不同控制器对这个情况处理方式不一样

## **v2 仅叶子组可以管理进程**

- 进程不允许附着在非叶子组

- *概述*
- *cgroup v1的缺陷*
- *cgroup v1 vs v2*
- ➡ ● **cgroup v2**
- *更多.....*

# ● cgroup v2 – 是时候了

## 是时候切换到cgroup v2了

- **几乎**实现了v1的全部控制器
- v2的控制器是全新设计的，吸取了v1的教训。
- 文档十分详细

## net\_cls和net\_prio无等价替代物

- 据说在iptables中实现

## v2控制器的演进历程

v2控制器	初始内核版本	对应的v1子系统
memory		memory
io		blkio
pids		pids
perf_event	4.11	perf_event
rdma	4.11	rdma
cpu	4.15	cpu和cpuacct
devices	4.15	devices
cpuset	5.0	cpuset
cgroup.freeze	5.2	freezer
hugetlb	5.6	hugetlb
无		net_cls, net_prio

# ● cgroup v2 – 与v1的异同点

## 与v1的相同点

- 以目录形式组织层次结构
- 所有进程初始化在根组 (root)
- 通过将PID写入cgroup.procs把进程迁移至指定组
- 读取cgroup.procs可以获取一个资源组的所有进程PID
- fork出来的子进程继承父进程的资源组
- 没有 (非僵尸) 进程的资源组可以被删除

## 与v1的不同点

- 根组不含控制器接口，即根组不应该限制资源。
- 进程只能挂在叶子节点上
- 从内核4.14版开始加入“线程模式 (thread mode) ”，但是，
  - 有严苛的约束条件，不能随意切换。
  - 仅特定控制器支持，例如，如果启用了memory控制器，则无法启用线程模式。

## ● cgroup v2 – 叶子组入驻进程实例

```
[root@bogon cgroup]#
```



## ● cgroup v2 – 挂载v2文件系统

**要使用v2，需要挂载新的文件系统类型：**

```
[root@bogon ~]# mount -t cgroup2 none /path/to/mount
```

**所有v2控制器自动挂载为单层结构**

- 无需显式挂载控制器到挂载点
- 无需指定 “-o controller” 挂载选项

**没有被挂载到其它层级的控制器才会挂载到v2**

## ● cgroup v2 – 可用控制器

每个v2控制组都有cgroup.controllers文件，列出了可用控制器。

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

**Linux 4.6之后，可以使用内核引导参数**

- cgroup\_no\_v1=all：禁止所有控制器使用cgroup v1
- cgroup\_no\_v1=controller,...：禁止指定的控制器使用cgroup v1

**虽然v1和v2可以混合使用，但是不建议生产环境这样配置。**

## ● cgroup v2 – 启用/禁用控制器

将可用控制器写入cgroup.subtree\_control来启用/禁用控制器。

```
# echo "+ pids -memory" > cgroup.subtree_control
```

- +：启用控制器
- -：禁用控制器
- 一组操作要么都成功，要么都失败。
- 一个控制器有多个操作时，以最后一个为准。

**cgroup.subtree\_control启用控制器后：**

- 允许自组控制该资源
- 子组目录自动创建该控制器对应的控制接口文件

## ● cgroup v2 – 启用控制器实例

查看根组所有可以控制器：

```
# cat cgroup . controllers  
cpu io memory pids
```

创建一个子控制组：

```
# mkdir grp  
# ls grp  
cgroup.controllers cgroup.events cgroup.procs cgroup.subtree_control
```

启用新控制组的pids控制器：

```
# echo "+pids" > cgroup . subtree_control  
# ls grp  
cgroup.controllers cgroup.events cgroup.procs cgroup.subtree_control  
pids.current pids.events pids.max
```

## ● cgroup v2 – 启用控制器实例

现在，grp组只有一个控制器：


```
# cat grp/cgroup.controllers  
pids
```

在grp组，启用它的子组的pids控制组：

```
# mkdir grp/sub  
# echo "+pids" > grp/cgroup.subtree_control  
# cat grp/cgroup.subtree_control  
pids
```

但是，不能启用其它控制器：

```
# echo "+cpu" > grp/cgroup.subtree_control  
sh: echo : write error : No such file or directory
```



## ● cgroup v2 – 自上而下的约束

### 自上而下的约束

- 子组始终受父组约束，即子组不能放宽父组的约束。
- 如果父组禁止了某个控制器，那么子组不可能启用该控制器。

# ● cgroup v2 – 入驻/闲置通知

## Populated (入驻) /Unpopulated (闲置)

- 废弃了v1的release\_agent和notify\_on\_release接口。
- 除了根组，每个控制组都有cgroup.events接口，提供populated字段。
  - 1: 当前组及其任何子组存在进程
  - 0: 无任何进程。

```
[root@bogon A]# cat cgroup.events
populated 1
frozen 0
```

- 通过监控cgroup.events文件可以获取入驻/闲置事件：
  - *inotify*: 监控IN\_MODIFY事件
  - *poll/epoll*: 监控POLLPRI/EPOLLPRI事件
- 一个进程可以监控多个cgroup.events文件
- 代价很小
- 通知可以委托给容器

# ● cgroup v2 – 代理

## 术语

- **委托** (Delegation) : 将层级中的子树授权给其他低权限用户来管理
- **委托者** (Delegater) : 拥有父组特权的用户
- **受托者** (Delegatee) : 被指派管理子层级的低权限用户

## 作用

- 获得授权的低权限用户可以管理子层级中的进程
- 对于非root运行的容器很有用



## ● cgroup v2 – 配置代理

### 委托者需授权受托者读写特定文件的权限

- 通常通过将所有权更改为受托者的UID来实现

### 除了委托子树的根目录外，目录中以下文件所有权也会更改：

- cgroups.procs
- cgroup.threads
- cgroup.subtree\_control
- /sys/kernel/cgroup/delegate中列出的其它文件

### 委托者不应该让受托者对资源控制接口文件有写权限

- 委托者使用上述文件来控制受托者的资源分配
- 受托者不应该有修改它们的权限

## ● cgroup v2 – 域(Domain)与线程(Threaded)模式

**v2的控制组最初都处于“域”模式（进程模式）：**

- 一个进程中的所有线程必须在同一个控制组中
- 这是cgroup v2的默认值

**满足条件的子树可以切换到“线程”模式：**

- 其所有子树必须是“线程”模式
- 一个进程的多个线程可以在“线程”子树下的不同控制组中
- 一个MT进程的所有线程必须在同一个“线程”子树中

**可以有多个“线程”子树，每个子树包含多个进程。**

**因此，v2虽然有“线程”线程模式，但比v1约束更严格。**

## ● cgroup v2 – 域与线程控制器

### 控制组被分成两类：

- 线程控制器（既支持线程也支持进程粒度控制）
  - *cpu*
  - *cpuset*
  - *perf\_event*
  - *pids*
- 域控制器（仅支持进程粒度控制）
  - *其它控制器*

# ● cgroup v2 – 线程模式接口文件

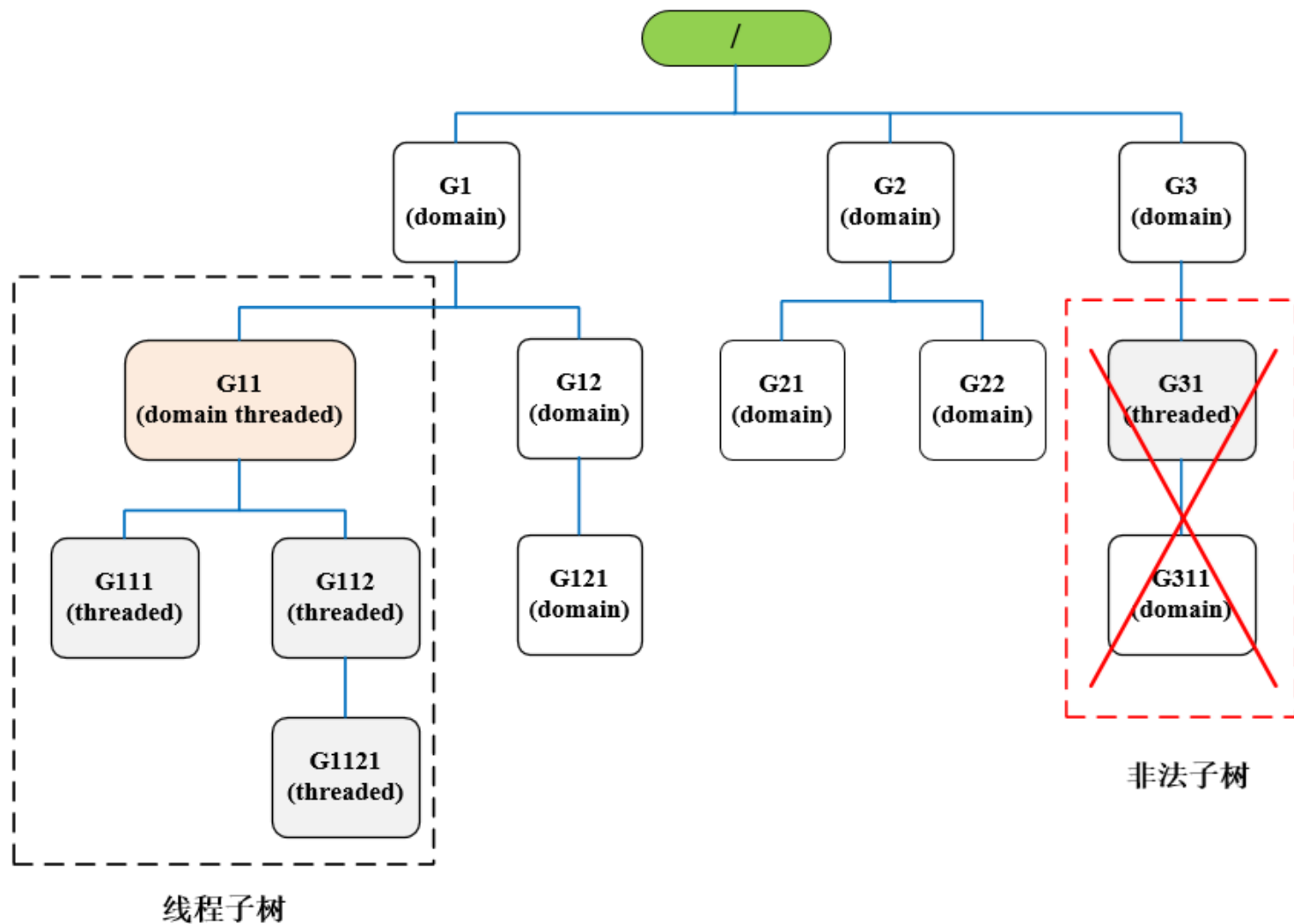
## **cgroup.threads:** cgroup组与线程ID关系接口

- 写入线程ID (LPWID) 则将指定线程迁入cgroup组
- 读取该文件可以获得该cgroup组的全部线程ID

## **cgroup.type:** cgroup组的模式接口

- *domain*: 进程控制粒度的cgroup组
- *threaded*: 线程控制粒度的cgroup组
- *domain threaded*: 作为线程cgroup子树根目录的域组
- *domain invalid*: 表示该cgroup组处于“无效”状态, 此时,
  - 不能入驻任何线程。
  - 不能有任何控制器
  - 无法转换成“threaded”模式组

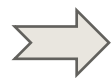
## ● cgroup v2 – 线程模式



## ● cgroup v2 – 演示domain threaded模式

```
root@bogon:/sys/fs/cgroup# pwd
/sys/fs/cgroup
root@bogon:/sys/fs/cgroup# mkdir A
root@bogon:/sys/fs/cgroup# cd A
root@bogon:/sys/fs/cgroup/A# cat cgroup.type
domain
root@bogon:/sys/fs/cgroup/A# mkdir B
root@bogon:/sys/fs/cgroup/A# echo "threaded" > B/cgroup.type
root@bogon:/sys/fs/cgroup/A# cat B/cgroup.type
threaded
root@bogon:/sys/fs/cgroup/A# mkdir C
root@bogon:/sys/fs/cgroup/A# cat C/cgroup.type
domain invalid
root@bogon:/sys/fs/cgroup/A# cat cgroup.type
domain threaded
```

- *概述*
- *cgroup v1的缺陷*
- *cgroup v1 vs v2*
- *cgroup v2*



- **更多.....**

# ● 更多 – WSL启用cgroup v2

## 环境

- 子系统: Windows Subsystem for Linux (WSL2) , Ubuntu 22.04.3 LTS
- 宿主机: Windows 11

## 方法<sup>[文献6]</sup>

1. 以下内容粘贴到资源管理器栏:

`notepad.exe %UserProfile%/.wslconfig`

2. 在.wslconfig文件添加以下几行:

`[wsl2]`

`kernelCommandLine = cgroup_no_v1=all systemd.unified_cgroup_hierarchy=1`

3. 以管理员权限启动PowerShell, 并执行:

`wsl --shutdown`

4. 如果报“远程主机强迫关闭了一个现有的连接”错误, 则执行:

`wsl --update`



## ● 更多 – cgroup v2 OEM

不同Linux发行版，cgroup可能会有差异。

以**openEuler** 20.03(LTS-SP3) 为例，增加了files控制器。

```
# pwd
/sys/fs/cgroup/
# ls
files.no_acct files.usage ...
# cat cgroup.controllers
cpu io memory pids rdma files
# mkdir 1
# echo "+files" > cgroup.subtree_control
# ls 1/
files.limit files.usage ...
```

### 代码提交记录：

- <https://mailweb.openeuler.org/hyperkitty/list/kernel@openeuler.org/thread/RYPM6GI2W6H4XJL4O4GADXATAH6FRKID/>

## ● 更多 – v2的已知缺陷

### Red Hat 8/CentOS 8无法使用CPU控制器

- 有些非内核应用程序（如rtkit-daemon, Oracle RAC, Red Hat Cluster）以实时（Real Time）调度策略运行，实时程序只能挂载在cgroup v1下。
- 详见 [\[参考文献7\]](#)

### iptables的bug

- 内核使用cgroup v2的bug，iptables使用`cgroup --path`参数时，路径识别错误。
- 详见 [\[参考文献8\]](#)

### 尚无由v2导致的CVE报告

## ● 更多 - 参考文献

1. [Cgroup v1: Red Hat Enterprise Linux 7 资源管理指南 \(中文\)](#)
2. [Resource Management Guide - Using cgroups to manage system resources on RHEL 7](#)
3. [Control Groups version 1](#)
4. [Control Group v2](#)
5. [What's new in control groups \(cgroups\) v2 — Open Source Summit Europe 2018](#)
6. [wsl-cgroupsv2](#)
7. [The cpu controller in cgroup v2 can not be used in Red Hat Enterprise Linux 8](#)
8. [Iptables cgroup V2 marking with systemd](#)

A series of white, overlapping geometric lines and polygons on a black background, located on the left side of the slide.

# 谢谢！

- 杨铸
- [www.200yi.com](http://www.200yi.com)
- [fairyfar@msn.com](mailto:fairyfar@msn.com)