



Red Hat Enterprise Linux 6

資源管理指南

在 Red Hat Enterprise Linux 6 上管理系統資源

版 1

Red Hat Enterprise Linux 6 資源管理指南

在 Red Hat Enterprise Linux 6 上管理系統資源
版 1

Martin Prpič

Red Hat 工程部出版中心

mprpic@redhat.com

Rüdiger Landmann

Red Hat 工程部出版中心

r.landmann@redhat.com

Douglas Silas

Red Hat 工程部出版中心

dhensley@redhat.com

法律聲明

Copyright © 2011 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

在 Red Hat Enterprise Linux 6 上管理系統資源

內容目錄

章 1. 控制群組 (CGROUP) 簡介	3
1.1. 如何管理控制群組	3
Linux 程序模型	3
Cgroup 模型	3
1.2. 子系統、階層、控制群組與工作之間的關係	4
1.3. 資源管理隱含式	5
章 2. 使用控制群組	6
2.1. CGCONFIG 服務	6
2.1.1. cgconfig.conf 檔案	6
2.2. 建立階層和連接子系統	8
額外方式	8
2.3. 將子系統連至既有階層，或由既有階層上將子系統中斷連結	9
額外方式	9
2.4. 卸載階層	10
2.5. 建立控制群組	11
額外方式	11
2.6. 移除控制群組	11
2.7. 設定參數	12
額外方式	13
2.8. 將程序移動至控制群組中	13
額外方式	13
2.8.1. cgroup Daemon	14
2.9. 在控制群組中啟動一項程序	14
額外方式	15
2.9.1. 在控制群組中啟動一項服務	15
2.10. 取得有關於控制群組的相關資訊	16
2.10.1. 尋找程序	16
2.10.2. 尋找子系統	16
2.10.3. 尋找階層	16
2.10.4. 尋找控制群組	16
2.10.5. 顯示控制群組的參數	17
2.11. 卸除控制群組	17
2.12. 額外資源	17
章 3. 子系統和可調整的參數	19
3.1. BLKIO	19
3.2. CPU	21
3.3. CPUACCT	22
3.4. CPuset	22
3.5. DEVICES	24
3.6. FREEZER	25
3.7. MEMORY	26
3.8. NET_CLS	28
3.9. NS	28
3.10. 額外資源	28
附錄 A. 修訂紀錄	30

章 1. 控制群組 (CGROUP) 簡介

Red Hat Enterprise Linux 6 提供了一項新的 kernel 功能：*control groups*，在本指南中簡稱為 *cgroup*。*Cgroup* 能讓您分配資源－例如 CPU 時間、系統記憶體、網路頻寬，或這些資源的組合－於系統上，使用者定義的運作中工作群組（程序）。您可監控您所配置的 *cgroup*，拒絕 *cgroup* 對於特定資源的存取，甚至是在一部運作中的系統上，動態式地重新配置您的 *cgroup*。*cgconfig*（“控制群組配置”）服務可被配置成在開機時啟用，並重新建立您預定義的 *cgroup*，如此便能使它們在重新開機時可保有一致性。

透過使用 *cgroup*，系統管理員能取得分配、處理優先順序、拒絕、管理，以及監控系統資源的細部控制。硬體資源可機敏地分配於工作與使用者之間，並提昇整體效率。

1.1. 如何管理控制群組

Cgroup 是透過階層式的方式來管理的，和程序、子群組相同，都會由它們的 *parent* 繼承部份屬性。然而，這兩個模型之間有所不同。

Linux 程序模型

Linux 系統上的所有程序皆為相同 *parent* 的子程序：*init* 程序，由 kernel 在開機時執行，並啟用其它程序（並且可能會相應地啟用它們自己的子程序）。因為所有程序皆源自於單獨的父程序，因此 Linux 的程序模型屬於單獨的階層或樹狀目錄。

此外，所有除了 *init* 以外的程序皆會繼承其父程序的環境（例如 *PATH* 變數）^[1] 與特定屬性（例如開放式的檔案描述元）。

Cgroup 模型

Cgroup 與程序的相似點為：

- 它們皆屬於階層式，並且
- 子 *cgroup* 會繼承其父群組的特定屬性。

基礎差異就是在同一部系統上，能夠同時存在許多不同的 *cgroup* 階層。若 Linux 程序模型是個程序的單樹狀，那麼 *cgroup* 模型便是個各別、未連接的樹狀工作（例如程序）。

多重各別的 *cgroup* 階層是必要的，因為各個階層皆連至了「一個或更多」個「子系統」。子系統^[2]代表單獨的資源，例如 CPU 時間或記憶體。Red Hat Enterprise Linux 6 提供了九個控制群組子系統，以名稱和功能列在下方。

Red Hat Enterprise Linux 中的可用子系統

- **blkio** – 此子系統可設置來自於，以及至區塊裝置（例如像是固態、USB 等等的實體磁碟）的輸入/輸出存取限制。
- **cpu** – 此子系統使用了排程器，以提供 CPU *cgroup* 工作的存取權限。
- **cpuacct** – 此子系統會自動產生 *cgroup* 中的工作所使用的 CPU 資源報告。
- **cpuset** – 此子系統會將個別的 CPU 與記憶體節點分配給 *cgroup* 中的工作。
- **devices** – 此子系統能允許或拒絕控制群組中的任務存取裝置。
- **freezer** – 此子系統可中止或復原控制群組中的工作。
- **memory** – 此子系統會根據使用於控制群組中的工作的記憶體資源，自動產生記憶體報告，然後設定這些工作所能使用的記憶體限制：

- **net_cls** – 此子系統會以一個 **class** 標識符號 (**classid**) 來標記網路封包，這能讓 **Linux** 流量控制器 (**tc**) 辨識源自於特定控制群組的封包。流量控制器能被配置來指定不同的優先順序給來自於不同控制群組的封包。
- **ns** – **namespace** 子系統。



注意

您在控制群組文件（例如 **man page** 或是 **kernel** 文件）中，可能會看見 **資源控制器** (**resource controller**) 或 **控制器** (**controller**) 這些名詞。這兩個名詞皆與“子系統” (**subsystem**) 同義，這是因為子系統一般會排程資源，或套用限制至它所連至之階層中的控制群組。

子系統（資源控制器）的定義非常地一般：它是個會針對於工作群組（例如程序）進行動作的物件。

1.2. 子系統、階層、控制群組與工作之間的關係

請記得，在控制群組術語中，系統程序會被稱為工作。

以下為幾項子系統、**cgroup** 階層與工作之間之關係上的規則，以及這些規則的相關解釋。

規則 1

任何單獨子系統（例如 **cpu**）最多皆可連接至一個階層。

因為如此，**cpu** 子系統無法連至兩個不同的階層。

規則 2

一個階層能夠連接一個或更多個子系統。

因此，**cpu** 和 **memory** 子系統（或任何數量的子系統）能連接至一個單獨的階層，不過它們不可連至任何其它階層。

規則 3

每當系統上建立了新的階層，系統上的所有工作一開始都會是該階層的預設控制群組的成員，該控制群組亦稱為 **root cgroup**。當您建立了任何一個階層時，系統上的各項工作，都只能是該階層中一個 **cgroup** 的成員。一項單獨的工作能包含在多個控制群組中，只要這些控制群組皆位於不同的階層中。當工作成為相同階層中，第二個控制群組的成員時，它便會被由該階層中的第一個控制群組中移除。一項工作不會有任何時候處於相同階層中的兩個控制群組中。

因此，若 **cpu** 和 **memory** 子系統連至了一個名為 **cpu_and_mem** 的階層，並且 **net_cls** 子系統連至了一個名為 **net** 的階層，那麼一項執行中的 **httpd** 程序便能成為 **cpu_and_mem** 中，任何一個控制群組的成員，以及 **net** 中，任何一個控制群組的成員。

cpu_and_mem 裡、屬於 **http** 程序之控制群組，可能會將它的 **CPU** 時間限制為其它程序所配置的一半，並將它的記憶體使用量限制為最大 **1,024 MB**。此外，在 **net** 中，它所屬的控制群組可能會將它的傳輸率限制為每秒 **30 MB**。

當第一個階層被建立時，系統上的所有工作都會屬於至少一個控制群組中的成員：**root cgroup**。因此當使用控制群組時，所有系統工作皆會屬於至少一個 **cgroup**。

規則 4

系統上任何會自行分叉的程序（工作），皆會建立一項子程序（工作）。子工作會自動地成為其父程序所屬之所有控制群組的成員。接著，子工作便可視需求被移至不同的控制群組，不過一開始，它總是會繼承其父工作的控制群組（以程序的術語來講為「環境」）。

因此，請考量 `httpd` 工作，它是個在 `cpu_and_mem` 階層中，名為 `half_cpu_1gb_max` 的控制群組中的成員，並且也是 `net` 階層中，一個名為 `trans_rate_30` 的控制群組中的成員。當該 `httpd` 程序將它自己分支時，它的子程序將會自動地成為 `half_cpu_1gb_max` 控制群組以及 `trans_rate_30` 控制群組的成員。它會繼承其父工作所屬的相同的控制群組。

在這之後，父工作與子工作便會是完全獨立的：更改一項工作所屬的控制群組不會影響其它工作。更改父工作的控制群組亦不會對其子工作有任何影響。概述：所有子工作一開始皆會繼承與其父工作相同的控制群組成員資格，然而這些成員資格可之後更改或移除。

1.3. 資源管理隱含式

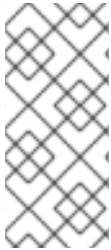
- 因為工作只能屬於任何一個階層中的一個控制群組，因此只有一種方式能使子系統限制或影響工作。這是種邏輯：一項功能，而非限制。
- 您可將數個子系統分組在一起，如此一來它們便會影響單獨階層中的所有工作。因為該階層中的控制群組設置了不同的參數，因此這些工作的影響會有所不同。
- 您可能有時必須重構一個階層。比方說將一個子系統由連接了數個子系統的階層中移除，並將它連至一個新的、獨立的階層。
- 相反地，若是無須將子系統分開在個別的階層上，您可將階層移除並將其子系統連至既有的階層。
- 這項設計能讓您輕易地使用控制群組，例如為單獨階層（例如僅連接了 `cpu` 與記憶體子系統的階層）中的特定工作設定幾個參數。
- 此設計能讓您進行高度指定的配置：系統上各項工作（程序）皆能是各個階層的成員，並且每個階層皆連接了一個單獨的子系統。此類型的配置能讓系統管理員擁有所有單獨工作的所有絕對控制權。

[1] 父程序能在將環境傳送給子程序之前，先進行修改。

[2] 請注意，子系統在 `libcgroup` man page 和其它文件中亦稱為「資源控制器」（resource controller），或僅是「控制器」（controller）。

章 2. 使用控制群組

控制群組最容易的使用方法就是安裝 **libcgroup** 套件，此套件包含了數個與 **cgroup** 相聯的指令列工具程式，以及其相聯的 **man page**。您可「掛載」階層，並使用任何系統上的 **shell** 指令和工具程式，設置 **cgroup** 參數（非一致性）。然而，使用 **libcgroup** 所提供的工具程式可簡化程序與延伸功能。因此，本指南內容專注於 **libcgroup** 指令。在大部分情況下，我們皆包含了相等的 **shell** 指令，以協助描述基礎機制。然而，我們建議您使用 **libcgroup** 指令。



注意

若要使用 **cgroups**，首先請以 **root** 身份，透過下列指令確認 **libcgroup** 套件已安裝在您的系統上：

```
~]# yum install libcgroup
```

2.1. CGCONFIG 服務

透過 **libcgroup** 套件安裝的 **cgconfig** 服務，提供了方便地建立階層、將子系統連接至階層，並在這些階層中管理 **cgroup** 的方式。我們建議您使用 **cgconfig** 來管理您系統上的階層和 **cgroup**。

就預設值，在 Red Hat Enterprise Linux 6 上，**cgconfig** 服務不會被啟用。當您透過 **chkconfig** 啟用該項服務時，它會讀取控制群組配置檔案 **/etc/cgconfig.conf**。如此一來，控制群組會在 **session** 執行時重新建立，並持續運行。根據配置檔案的內容，**cgconfig** 會建立階層、掛載所需的檔案系統、建立控制群組、並為每個群組設定子系統。

與 **libcgroup** 套件一起安裝的預設 **/etc/cgconfig.conf** 檔案，會為各系統建立與掛載個別的階層，並將子系統連接至這些階層。

若您停用了 **cgconfig** 服務（透過 **service cgconfig stop**），它會將它所掛載的所有階層卸載。

2.1.1. cgconfig.conf 檔案

/etc/cgconfig.conf 檔案包含了兩種主要的條目：**mount** 與 **group**。掛載條目會建立、掛載階層，使之成為虛擬檔案系統；並把子系統連結到這些階層上。舉例來說：

```
mount {
    <controller> = <path>;
    ...
}
```

範例用法請見〈[範例 2.1, “建立掛載點”](#)〉。

範例 2.1. 建立掛載點

以下範例會為 **cpuset** 子系統建立一組階層：

```
mount {
    cpuset = /cgroup/cpu;
}
```

這命令列相當於：

```
~]# mkdir /cgroup/cpu
~]# mount -t cgroup -o cpu cpu /cgroup/cpu
```

群組條目會建立控制群組，並設定子系統參數。群組條目會透過以下語法定義：

```
group <name> {
    [<permissions>]
    <controller> {
        <param name> = <param value>;
        ...
    }
    ...
}
```

請注意，**permissions** 一節是選用的。要定義群組條目的允許權限，請使用以下語法：

```
perm {
    task {
        uid = <task user>;
        gid = <task group>;
    }
    admin {
        uid = <admin name>;
        gid = <admin group>;
    }
}
```

範利用法請參見〈[範例 2.2, “建立群組條目”](#)〉：

範例 2.2. 建立群組條目

以下範例會為 **sql daemon** 建立控制群組，賦予 **sqladmin** 群組的使用者把任務加入控制群組的權限，並賦予 **root** 使用者修改子系統參數的權限。

```
group daemons/sql {
    perm {
        task {
            uid = root;
            gid = sqladmin;
        } admin {
            uid = root;
            gid = root;
        }
    } cpu {
        cpu.shares = 100;
    }
}
```

跟〈[範例 2.1, “建立掛載點”](#)〉裡的掛載點範例結合之後，相當於以下指令：

```
~]# mkdir -p /cgroup/cpu/daemons/sql
~]# chown root:root /cgroup/cpu/daemons/sql/*
~]# chown root:sqladmin /cgroup/cpu/daemons/sql/tasks
```

```
~]# echo 100 > /cgroup/cpu/daemons/sql/cpu.shares
```



注意

要讓 **/etc/cgconfig.conf** 裡的改變生效，請重新啟動 **cgconfig** 服務：

```
~]# service cgconfig restart
```

當您安裝 **cgroups** 時，配置範例會寫入 **/etc/cgconfig.conf**。以 **#** 符號開始的每一行表示註解，**cgconfig** 服務不會處理。

2.2. 建立階層和連接子系統



警告

下列指示涵蓋了建立新階層，並將子系統連至此階層的指示，它假設了控制群組尚未配置於您的系統上。在此情況下，這些指示將不會影響系統的作業。然而，更改一個含有工作的控制群組的可調整參數，則會即刻影響這些工作。本指南會在第一次更改可能會影響一或更多項工作的可調整控制群組參數時，進行警告。

在一部已（透過手動式或 **cgconfig** 服務）配置了控制群組的系統上，這些指令將會失敗，除非您先將會影響系統作業的既有階層卸載。請勿在生產系統上試驗這些指示。

若要建立一個階層，並將子系統連至該階層，請以 **root** 身份編輯 **/etc/cgconfig.conf** 檔案的 **mount** 部份。**mount** 部份中的項目格式如下：

```
子系統 = /cgroup/階層；
```

當 **cgconfig** 下次啟用時，它將會建立階層，並將子系統連接至該階層。

下列範例會建立一個名為 **cpu_and_mem** 的階層，並將 **cpu**、**cpuset**、**cpuacct**，以及 **memory** 子系統連接至此階層。

```
mount {
    cpuset = /cgroup/cpu_and_mem;
    cpu    = /cgroup/cpu_and_mem;
    cpuacct = /cgroup/cpu_and_mem;
    memory = /cgroup/cpu_and_mem;
}
```

額外方式

您亦可使用 **shell** 指令和工具程式來建立階層，並將子系統連至它們。

以 **root** 身份為階層建立「掛載點」。請在掛載點中包含控制群組的名稱：

```
~]# mkdir /cgroup/name
```

例如：

```
~]# mkdir /cgroup/cpu_and_mem
```

接下來，請使用 **mount** 指令來掛載階層，並同時連接一個或更多個子系統。例如：

```
~]# mount -t cgroup -o subsystems name /cgroup/name
```

subsystems 是個以逗號區隔開的子系統清單，而 *name* 代表階層的名稱。所有可用子系統的詳細描述列在 [Red Hat Enterprise Linux 中的可用子系統](#) 中，並且 [章 3, 子系統和可調整的參數](#) 提供了詳細的參照。

範例 2.3. 使用掛載指令來連接子系統

在此範例中，有個名為 **/cgroup/cpu_and_mem** 的目錄已存在，它會被作為是我們所建立之階層的掛載點。我們將 **cpu**、**cpuset** 和 **memory** 子系統連至一個我們命名為 **cpu_and_mem** 的階層，並 **mount cpu_and_mem** 階層於 **/cgroup/cpu_and_mem** 上：

```
~]# mount -t cgroup -o cpu,cpuset,memory cpu_and_mem /cgroup/cpu_and_mem
```

您可透過 **lssubsys** 指令列出所有可用子系統，以及其目前的掛載點（比方說，它們所連至的階層的掛載位置）^[3]：

```
~]# lssubsys -am
cpu,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
cpuacct
devices
freezer
blkio
```

此輸出顯示了：

- **cpu**、**cpuset** 和 **memory** 子系統連至了一個掛載於 **/cgroup/cpu_and_mem** 上的階層，並且
- 基於缺少相對映的掛載點，**net_cls**、**ns**、**cpuacct**、**devices**、**freezer** 以及 **blkio** 子系統尚未連至任何階層。

2.3. 將子系統連至既有階層，或由既有階層上將子系統中斷連結

若要將子系統附加至既有的階層，請切斷它與既有階層的連結，或將它移至一個不同的階層中，以 **root** 身份編輯 **/etc/cgconfig.conf** 檔案的 **mount** 部份，並使用與描述於〈[節 2.2, “建立階層和連接子系統”](#)〉中的相同語法。當 **cgconfig** 下次啟動時，它便會根據您所指定的階層辨識子系統。

額外方式

若要將子系統連至既有的階層，請重新掛載該階層。請在 **mount** 指令中包含額外的子系統，以及 **remount** 選項。

範例 2.4. 重新掛載一個階級，以新增子系統

lssubsys 指令會顯示連至 **cpu_and_mem** 階層的 **cpu**、**cpuset** 以及 **memory** 子系統：

```
~]# lssubsys -am
cpu,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
cpuacct
devices
freezer
blkio
```

我們使用了 **remount** 選項來重新掛載 **cpu_and_mem** 階層，並在子系統中包含了 **cpuacct**：

```
~]# mount -t cgroup -o remount,cpu,cpuset,cpuacct,memory cpu_and_mem
/cgroup/cpu_and_mem
```

lssubsys 指令現在顯示了 **cpuacct** 已連至 **cpu_and_mem** 階層：

```
~]# lssubsys -am
cpu,cpuacct,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
devices
freezer
blkio
```

相似地，您可藉由重新掛載既有的階層，並透過 **-o** 指令來省略子系統名稱，以將子系統由該階層切除連結。比方說，若要切斷 **cpuacct** 子系統的連結，請重新掛載並省略它：

```
~]# mount -t cgroup -o remount,cpu,cpuset,memory cpu_and_mem
/cgroup/cpu_and_mem
```

2.4. 卸載階層

您可透過執行 **umount** 指令來卸載控制群組的階層：

```
~]# umount /cgroup/name
```

例如：

```
~]# umount /cgroup/cpu_and_mem
```

若階層目前是空的（也就是說它只包含了 **root** 控制群組），該階層將會在卸載時被停用。若階層包含任何其它控制群組，儘管該階層已卸載，它在 **kernel** 中也會保持為啟用狀態。

若要移除階層，請在卸載階層之前，確認所有子群組皆已移除，或使用 **cgclear** 指令，來停用就算是非空的階層－請參閱〈[節 2.11, “卸除控制群組”](#)〉。

2.5. 建立控制群組

使用 **cgcreate** 指令來建立控制群組。**cgcreate** 的語法為：**cgcreate -t uid:gid -a uid:gid -g subsystems:path**，其中：

- **-t** (選用) – 指定使用者 (透過使用者 ID，也就是 **uid**) 以及群組 (透過群組 ID，亦即 **gid**) 來擁有 **tasks** 偽檔案，給此控制群組使用。該使用者可新增工作至控制群組。



注意

請注意，由控制群組移除工作的唯一方法就是將它移至不同的控制群組。若要移動一項工作，使用者必須擁有 **destination** 控制群組的寫入權限；來源控制群組的寫入權限並不重要。

- **-a** (選用)：讓使用者 (透過使用者 ID，也就是 **uid**) 以及群組 (透過群組 ID，亦即 **gid**) 來擁有所有偽檔案，而不是讓此控制群組的 **tasks** 擁有。這使用者可以修改擁有系統資源的控制群組之任務。
- **-g** – 指定控制群組應建立於哪個階層中，作為以逗號區隔開、與這些階層相聯的子系統。若此清單中的子系統位於不同的階層中，那麼群組便會建立於各個這些階層中。階層之後會包含一個冒號，以及與階層相關之子群組的路徑。請勿在路徑中包含階層的掛載點。

比方說，位於 **/cgroup/cpu_and_mem/lab1/** 目錄的控制群組會稱為 **lab1** – 它的路徑已獨特地指定了，因為一個子系統最多只能屬於一個階層。此外，請注意群組是由控制群組所建立於的所有階層中的子系統所控制的，儘管這些子系統未指定於 **cgcreate** 指令中 – 請參閱 [範例 2.5, “cgcreate 使用方法”](#)。

因為相同階層中的所有控制群組皆擁有相同的控制器，因此子群組會擁有與其父群組相同的控制器。

範例 2.5. cgcreate 使用方法

考量一部系統，其 **cpu** 和 **memory** 子系統一起掛載於 **cpu_and_mem** 階層中，並且 **net_cls** 控制器已掛載於一個獨立、名為 **net** 的階層中。我們現在將執行：

```
~]# cgcreate -g cpu,net_cls:/test-subgroup
```

cgcreate 指令會建立兩個群組，名為 **test-subgroup**，一個位於 **cpu_and_mem** 階層中，而另一個則位於 **net** 階層中。**cpu_and_mem** 階層中的 **test-subgroup** 群組是由 **memory** 子系統所控制的，儘管我們並未在 **cgcreate** 指令中指定它。

額外方式

若要直接建立控制群組的子群組，請使用 **mkdir** 指令：

```
~]# mkdir /cgroup/hierarchy/name/child_name
```

例如：

```
~]# mkdir /cgroup/cpuset/lab1/group1
```

2.6. 移除控制群組

以 **cgdelete** 移除控制群組，其語法與 **cgcreate** 相似。請執行：**cgdelete 子系統 :路徑**，而：

- 子系統代表一個以逗號區隔開的子系統清單。
- 路徑代表與階層之 **root** 相關的控制群組路徑。

例如：

```
~]# cgdelete cpu,net_cls:/test-subgroup
```

cgdelete 亦可遞迴地透過 **-r** 選項，移除所有子群組。

當您刪除控制群組時，它所有的工作皆會移至它的父群組中。

2.7. 設定參數

請以一組含有權限的使用者帳號執行 **cgset** 指令來設置子系統參數，以修改相關的控制群組。比方說，若 **/cgroup/cpuset/group1** 存在的話，請以下列指令來指定此群組擁有其存取權限的 CPU：

```
cpuset]# cgset -r cpuset.cpus=0-1 group1
```

cgset 的語法為：**cgset -r 參數 =值 控制群組路徑**，而：

- 參數代表欲設置的參數，這與控制群組目錄中的檔案相應
- 值為參數的值
- *path_to_cgroup* 代表與階層 **root** 相關的控制群組之路徑。比方說，若要設置 **root** 群組（如果 **/cgroup/cpuacct** 存在的話）的參數，請執行：

```
cpuacct]# cgset -r cpuacct.usage=0 /
```

此外，因為 **.** 與 **root** 群組相關（也就是 **root** 群組本身），因此您亦可執行：

```
cpuacct]# cgset -r cpuacct.usage=0 .
```

然而，請注意 **/** 為建議的語法。



注意

只有一小部份的參數可以用來設定 **root** 群組（例如上述範例所顯示的 **cpuacct.usage**）。這是因為 **root** 群組擁有所有的現有資源，因此透過定義某些參數（例如 **cpuset.cpu** 參數）以限制現有程序，是沒有道理的。

若要設置 **group1** 參數（也就是 **root** 群組的一個子群組），請執行：

```
cpuacct]# cgset -r cpuacct.usage=0 group1
```

群組名稱後的斜線（例如 **cpuacct.usage=0 group1/**）乃非必要的。

您可以 **cgset** 設置的值取決於特定階層中上層所設置的值。比方說，若 **group1** 被限制為在一部系統上只使用 CPU 0，您無法設置 **group1/subgroup1** 來使用 CPUs 0 和 1，或是只使用 CPU 1。

您亦可使用 **cgset** 來將一個控制群組中的參數複製至另一個既有的控制群組中。例如：

```
~]# cgset --copy-from group1/ group2/
```

透過 **cgset** 來複製參數的語法為：**cgset --copy-from *path_to_source_cgroup* *path_to_target_cgroup***，而：

- *path_to_source_cgroup* 代表與階層之 **root** 群組相關，欲複製其參數的控制群組之路徑
- *path_to_target_cgroup* 是目的地控制群組的路徑，與階層之 **root** 群組相關

在您由一個群組複製參數至另一群組前，請確認各子系統的所有必要參數皆已設置，否則指令將會失敗。欲知必要參數的詳細資訊，請參閱〈[重要 – 必要的參數](#)〉。

額外方式

若要直接設置一個控制群組中的子系統的參數，請透過使用 **echo** 指令來將值插入至相聯的子系統偽檔案中。比方說，這項指令會將 **0-1** 這個值插入 **group1** 控制群組的 **cpuset.cpus** 偽檔案中：

```
~]# echo 0-1 > /cgroup/cpuset/group1/cpuset.cpus
```

當使用了這個值時，此控制群組中的工作會被限制僅可使用系統上的 CPU 0 和 1。

2.8. 將程序移動至控制群組中

您可透過執行 **cgclassify** 指令來將程序移至控制群組中：

```
~]# cgclassify -g cpu,memory:group1 1701
```

cgclassify 的語法為：**cgclassify -g *subsystem:path_to_cgroup* *pidlist***，而：

- *subsystem* (子系統) 代表一個逗號區隔開的子系統清單，* 以啟動與所有可用子系統相聯的階層中的程序。請注意，若相同名稱的控制群組存在多重階層中，**-g** 選項便會移動各個這些群組中的程序。請確認控制群組存在您在此所指定的子系統的各個階層中。
- *path_to_cgroup* 代表階層中的控制群組之路徑
- *pid* 清單代表以空格隔開的 *process identifier* (PID) 清單

您亦可在 *pid* 之前附加一個 **--sticky** 選項，以保留相同控制群組中的任何子程序。若您不設置此選項，並且 **cgred daemon** 正在執行的話，子程序將會被根據 **/etc/cgrules.conf** 中的設定，分配給控制群組。然而，程序本身將會保留在您所啟動該程序的控制群組中。

透過使用 **cgclassify**，您能夠同時地移動多項程序。比方說，這項指令可將 PID 為 **1701** 以及 **1138** 的程序移至控制群組 **group1/** 中：

```
~]# cgclassify -g cpu,memory:group1 1701 1138
```

請注意，欲移動的 PID 將以空格區隔開來，並且已指定的群組應位於不同的階層中。

額外方式

若要直接將一項程序移至控制群組中，請將它的 PID 寫入控制群組的 **tasks** 檔案中。比方說，若要將 PID 為 **1701** 的程序移至位於 **/cgroup/lab1/group1/** 的控制群組中：

```
~]# echo 1701 > /cgroup/lab1/group1/tasks
```

2.8.1. cgred Daemon

Cgred 是個會根據設定於 **/etc/cgrules.conf** 檔案中的參數，來將工作移至控制群組中的 **daemon**。**/etc/cgrules.conf** 檔案中的項目能夠是以下兩種格式之一：

- 使用者 階層 控制群組
- 使用者:指令 階層 控制群組

例如：

```
maria    devices    /usergroup/staff
```

此項目指定了任何屬於使用者 **maria** 的程序將根據指定於 **/usergroup/staff** 控制群組中的參數，存取裝置子系統。若要相聯特定指令與特定控制群組，請如下附加 **command** 參數：

```
maria:ftp devices    /usergroup/staff/ftp
```

項目現在指定了當名稱為 **maria** 的使用者使用了 **ftp** 指令時，程序會自動地移至包含了 **devices** 子系統的階層中的 **/usergroup/staff/ftp** 控制群組。然而，請注意，**daemon** 只會在適當的條件滿足後，才會將程序移至控制群組。因此，**ftp** 程序可能會在錯誤的群組中，短暫執行。此外，若該程序在錯誤的群組中，快速地衍生了子程序，這些子程序將可能不會被移動。

/etc/cgrules.conf 檔案中的項目可包含下列額外標記法：

- @ – 當放置在 **使用者** 之前，便代表群組，而非各別的使用者。比方說，**@admins** 代表 **admins** 群組中的所有使用者。
- * – 代表了「全部」。例如，子 系 統 欄位中的 * 代表了所有的子系統。
- % – 代表了一個與上面一行中的項目相同的項目。例如：

```
@adminstaff devices    /admingroup
@labstaff    %    %
```

2.9. 在控制群組中啟動一項程序

重要

有些子系統需要您在將任務移動到使用這些子系統的控制群組之前，先設置必要的參數。比方說，在您使用 **cpuset** 子系統之前，您必須先定義 **cpuset.cpus** 和 **cpuset.mems** 參數。

此部份中的範例描述了指令的正確語法，不過這只在已設置了範例中所使用的控制器之必要參數的系統上有效。若您還未配置相關的控制器，您不能將此部份中的範例指令直接複製，並預期它們能在您的系統上運作。

請參閱 [節 3.10, “額外資源”](#)，以取得特定子系統的必要參數之詳述。

您可透過執行 **cgexec** 指令來在控制群組中啟動程序。比方說，這項指令會在 **group1** 控制群組中啟動 **lynx** 網站瀏覽器，並且遵照 **cpu** 子系統在該群組上所強制使用的限制：

```
~]# cgexec -g cpu:group1 lynx http://www.redhat.com
```

cgexec 的語法為：**cgexec -g subsystems:path_to_cgroup command arguments**，而：

- **subsystem** (子系統) 代表一個逗號區隔開的子系統清單，* 以啟動與所有可用子系統相聯的階層中的程序。請注意，如描述於〈[節 2.7, “設定參數”](#)〉中的 **cgset**，若相同名稱的控制群組存在多重階層中，**-g** 選項便會在各個這些群組中建立程序。請確認控制群組存在您在此所指定的子系統的多個階層中。
- **path_to_cgroup** 代表與階層相關的控制群組之路徑。
- **command** 代表欲執行的指令
- **arguments** 為指令的任何引數

您亦可在 **command** 之前附加 **--sticky** 選項，以保留相同控制群組中的任何子程序。若您不設置此選項，而 **cgred daemon** 正在執行的話，子程序將會被根據 **/etc/cgrules.conf** 中的設定，分配給控制群組。然而，程序本身將會保留在您所啟動其的控制群組中。

額外方式

當您啟動一項新程序時，它會繼承其父程序的群組。因此，另一項在特定控制群組中啟動程序的方式，就是將您的 **shell** 程序移至該群組（請參閱〈[節 2.8, “將程序移動至控制群組中”](#)〉），並由該 **shell** 啟動程序。例如：

```
~]# echo $$ > /cgroup/lab1/group1/tasks
lynx
```

請注意，在退出了 **lynx** 之後，您既有的 **shell** 還會存在 **group1** 控制群組中。因此，有個更佳的方式就是：

```
~]# sh -c "echo \$$ > /cgroup/lab1/group1/tasks && lynx"
```

2.9.1. 在控制群組中啟動一項服務

您可在控制群組中啟用一些服務。可啟用於控制群組中的服務必須：

- 使用一個 **/etc/sysconfig/servicename** 檔案
- 使用來自於 **/etc/init.d/functions** 的 **daemon()** 功能來啟用服務

若要在控制群組中啟動一項服務，請編輯它位於 `/etc/sysconfig` 目錄中的檔案，以包含一項格式為 **`CGROUP_DAEMON="subsystem:control_group"`** 的項目，*subsystem* 為與特定階層相聯的子系统，*control_group* 則為該階層中的控制群組。例如：

```
CGROUP_DAEMON="cpuset:daemons/sql"
```

2.10. 取得有關於控制群組的相關資訊

2.10.1. 尋找程序

若要找出某項程序屬於哪個控制群組，請執行：

```
~]$ ps -o cgroup
```

或是，若您知道程序的 PID，請執行：

```
~]$ cat /proc/PID/cgroup
```

2.10.2. 尋找子系统

若要尋找在您 **kernel** 中可用的子系统，以及它們是如何一起掛載至階層，請執行：

```
~]$ cat /proc/cgroups
```

或是，若要尋找特定子系統的掛載點，請執行：

```
~]$ lssubsys -m subsystems
```

其中 *subsystem* 代表您有興趣的子系统之清單。請注意 **lssubsys -m** 指令只會傳回每個階層的最高層掛載點。

2.10.3. 尋找階層

建議您將階層掛載在 **/cgroup** 下。假設您系統上已這麼做了，請列出或瀏覽該目錄的內容，以取得一系列階層的清單。若 **tree** 已安裝在您的系統上，請執行它以取得所有階層與其中的控制群組的總覽：

```
~]$ tree /cgroup/
```

2.10.4. 尋找控制群組

若要列出系統上的控制群組，請執行：

```
~]$ lscgroup
```

您可藉由指定控制器與路徑（格式為 **控制器:路徑**），來限制至特定階層的輸出。例如：

```
~]$ lscgroup cpuset:adminusers
```

只列出 **cpuset** 子系统連至的階層中的 **adminusers** 控制群組的子群組。

2.10.5. 顯示控制群組的參數

若要顯示特定控制群組的參數，請執行：

```
~]$ cgget -r parameter list_of_cgroups
```

parameter 代表一個包含了子系統的值之偽檔案，並且 *list_of_cgroups* 則代表以空格區隔開的控制群組之清單。例如：

```
~]$ cgget -r cpuset.cpus -r memory.limit_in_bytes lab1 lab2
```

顯示了控制群組 **lab1** 和 **lab2** 的 **cpuset.cpus** 與 **memory.limit_in_bytes** 的值。

若您不曉得參數本身的名稱為何，請使用一項類似下列的指令：

```
~]$ cgget -g cpuset /
```

2.11. 卸除控制群組



警告

cgclear 指令會將所有階層中全部的控制群組刪除掉。若您沒有將這些階層儲存在一個配置檔案中的話，您將無法重新建立它。

若要清除整個控制群組檔案系統，請使用 **cgclear** 指令。

控制群組中的所有工作皆會被重新分配至階層的 **root** 節點，所有的控制群組皆會被移除，並且檔案系統本身將會由系統上卸載，並同時刪除所有先前掛載的階層。最後，控制群組檔案系統所掛載於的目錄，也會被刪除掉。



注意

使用 **mount** 指令來建立控制群組（相對於透過 **cgconfig** 服務來建立）會在 **/etc/mtab** 檔案（檔案系統掛載表）裡建立條目。這項改變也會反映在 **/proc/mounts** 檔案裡。然而，使用 **cgclear** 指令卸載控制群組，加上其它 **cgconfig** 指令，會使用直接的 **kernel** 介面，不會改變 **/etc/mtab** 檔案中的內容，而且只會將新的指令寫入 **/proc/mounts** 檔案裡。因此，使用 **cgclear** 指令上傳控制群組之後，卸載的控制群組可能還是會在 **/etc/mtab** 檔案裡，而且也因此，執行 **mount** 指令還是會顯示出來。我們建議您參考 **/proc/mounts** 檔案，以得知正確的已掛載的控制群組清單。

2.12. 額外資源

控制群組指令的定義文件為 **libcgroup** 套件所提供的指南。部份號碼指定於下列的 **man page** 清單中。

libcgroup Man Page

- **man 1 cgclassify – cgclassify** 指令可使用來將執行中的工作，移至一個或更多個

cgroups 中。

man 1 cgclear – cgclear 指令可使用來刪除階層中的所有 cgroup。

man 5 cgconfig.conf – cgroup 定義於 **cgconfig.conf** 檔案中。

man 8 cgconfigparser – cgconfigparser 指令會剖析 **cgconfig.conf** 檔案，並掛載階層。

man 1 cgcreate – cgcreate 指令會在階層中建立新的 cgroup。

man 1 cgdelete – cgdelete 指令會將指定的 cgroup 移除。

man 1 cgexec – cgexec 指令會在指定的 cgroup 中執行工作。

man 1 cgget – cgget 指令會顯示 cgroup 參數。

man 5 cgreg.conf – cgreg.conf 為 **cgreg** 服務的配置檔案。

man 5 cgrules.conf – cgrules.conf 包含了使用來決定工作何時屬於特定 cgroup 的規則。

man 8 cgrulesengd – cgrulesengd 服務會將工作分配給 cgroup。

man 1 cgset – cgset 指令會為 cgroup 設置參數。

man 1 lscgroup – lscgroup 指令會列出階層中的 cgroup。

man 1 lssubsys – lssubsys 指令會列出包含了指定的子系統的階層。

[3] **lssubsys** 指令為 **libcgroup** 套件所提供的工具程式之一。您必須安裝 **libcgroup** 才可使用它：若您無法執行 **lssubsys**，請參閱〈[章 2, 使用控制群組](#)〉。

章 3. 子系統和可調整的參數

「子系統」(Subsystem) 是可偵測到控制群組的數個 kernel 模組。一般來講，它們是能將不同層級的系統資源，分配給不同控制群組的資源控制器。然而，子系統亦可針對不同程序群組而定，被設計為與 kernel 進行任何的其它互動。用來開發新子系統的「應用程式介面」(API) 記載於 kernel 文件中的 `cgroups.txt` 中，存放在您系統的 `/usr/share/doc/kernel-doc-kernel-version/Documentation/cgroups/` 上 (由 kernel-doc 套件提供)。最新版本的 cgroups 文件亦可藉由 <http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt> 取得。不過請注意，最新版本文件中的功能，可能會與安裝在您系統上的 kernel 中的功能不相符。

包含了控制群組的子系統參數的「狀態物件」會被顯示為控制群組的虛擬檔案系統中的「偽檔案」(pseudofiles)。這些偽檔案可透過 shell 指令，或是它們的相等系統調用來操作。比方說，`cpuset.cpus` 是個可指定控制群組允許存取哪個 CPU 的偽檔案。若是 `/cgroup/cpuset/webserver` 是個在一部系統上執行的網站伺服器的控制群組，而我們執行下列指令：

```
~]# echo 0,2 > /cgroup/cpuset/webserver/cpuset.cpus
```

便會將 `0,2` 寫入 `cpuset.cpus` 偽檔案中，並從而限制任何 PID 列在了 `/cgroup/cpuset/webserver/tasks` 中的工作，使它們在系統上僅可使用 CPU 0 以及 CPU 2。

3.1. BLKIO

區塊 I/O (`blkio`) 子系統負責控制和監控控制群組中的工作，對於區塊裝置上 I/O 的存取權限。將數值寫入某些偽檔案中會限制存取功能或頻寬，從這些偽檔案中讀取資訊能提供關於 I/O 運作的資訊。

blkio.weight

這指定了控制群組就預設值，可使用的區塊 I/O 存取權限的相應比例 (權重)，範圍為 **100** 到 **1000**。特定裝置會以 `blkio.weight_device` 參數置換這個值。比方說，若要指定預設權重 **500** 給某個控制群組，使其能夠存取區塊裝置，請執行：

```
echo 500 > blkio.weight
```

blkio.weight_device

這指定了控制群組可使用的特定裝置的 I/O 存取權限的相關比例 (權重)，範圍為 **100** 到 **1000**。此參數的值會置換所指定的裝置的 `blkio.weight` 值。這些值的格式為 `major:minor weight`，其中 `major` 與 `minor` 和裝置類型與節點編號表示於 *Linux 已分配的裝置* 中，亦稱為 *Linux 裝置清單*，並可藉由 <http://www.kernel.org/doc/Documentation/devices.txt> 取得。比方說，若要指定權重 **500** 給某個控制群組使其能夠存取 `/dev/sda`，請執行：

```
echo 8:0 500 > blkio.weight_device
```

在 *Linux 已分配的裝置* 標記法中，`8:0` 代表 `/dev/sda`。

blkio.time

回報控制群組擁有特定裝置的 I/O 存取權限的時間。項目含有三個欄位：`major`、`minor` 以及 `time`。`Major` 和 `minor` 為指定於 *Linux 已分配的裝置* 中的裝置類型與節點編號，而 `time` 為時間長度，單位為毫秒 (ms)。

blkio.sectors

回報控制群組的特定裝置的輸入與輸出磁區的數量。項目包含四個欄位：*major*、*minor*以及 *sectors*。*major* 和 *minor* 為指定於 *Linux* 已分配的裝置中的裝置類型和節點編號，*sectors* 則為磁碟磁區的數量。

blkio.io_service_bytes

回報了控制群組由特定裝置傳出或傳入的位元組數量。項目含有四個欄位：*major*、*minor*、*operation*，以及 *bytes*。*major* 與 *minor* 為指定於 *Linux* 已分配的裝置中的裝置類型與節點編號，*operation* 代表作業的類型（*read*、*write*、*sync*，或是 *async*），並且 *bytes* 代表已傳輸的位元組數量。

blkio.io_serviced

回報控制群組在特定裝置上執行的 I/O 作業數量。項目含有四個欄位：*major*、*minor*、*operation* 以及 *bytes*。*major* 和 *minor* 為指定於 *Linux* 已分配的裝置中的裝置類型與節點編號，而 *operation* 代表作業的類型（*read*、*write*、*sync* 或 *async*），並且 *number* 代表作業的數量。

blkio.io_service_time

回報控制群組在特定裝置上，I/O 作業的請求送出與完成之間所花的時間。項目含有四個欄位：*major*、*minor*、*operation* 以及 *bytes*。*major* 和 *minor* 為指定於 *Linux* 已分配的裝置中的裝置類型和節點編號，*operation* 代表作業的類型（*read*、*write*、*sync* 或 *async*），並且 *time* 為時間的長度，單位為十億分之一秒（*ns*）。時間會以十億分之一秒為單位回報，而非較大的單位，如此一來此報告會較有意義，就算是針對於固態裝置亦然。

blkio.io_wait_time

回報控制群組在特定裝置上的 I/O 作業，等待排程器佇列中的服務所耗費的時間總長。當您詮釋此報告時，請注意：

- 所回報的時間可能會比花掉的時間長，因為回報的時間為控制群組的所有 I/O 作業的時間總長，而非控制群組本身花在等待 I/O 作業的時間。若要找出群組整體所花費的等待時間，請使用 **blkio.group_wait_time**。
- 若裝置有個 **queue_depth > 1**，所回報的時間便只包含直到請求被送出至裝置的時間，而非在裝置重新排序請求時，等待服務所花費的時間。

項目含有四個欄位：*major*、*minor*、*operation* 以及 *bytes*。*Major* 和 *minor* 為指定於 *Linux* 已分配的裝置中的裝置類型和節點編號，*operation* 代表作業類型（*read*、*write*、*sync* 或是 *async*），並且 *time* 為時間長度，單位為十億分之一秒（*ns*）。時間會以十億分之一秒為單位回報，而非較大的單位，如此一來此報告會較有意義，就算是針對於固態裝置。

blkio.io_merged

回報控制群組請求合併入 I/O 作業請求的 BIOS 數量。項目含有兩個欄位：*number* 和 *operation*。*number* 為請求的數量，並且 *operation* 代表作業類型（*read*、*write*、*sync* 或 *async*）。

blkio.io_queued

回報控制群組的 I/O 作業請求排程數量。項目含有兩個欄位：*number* 和 *operation*。*number* 為回報請求的數量，並且 *operation* 代表作業類型（*read*、*write*、*sync* 或 *async*）。

blkio.avg_queue_size

回報控制群組的平均 I/O 作業佇列大小（在整個群組存在的期間）。每當此控制群組的佇列取得一個 *timeslice* 時，佇列大小就會被取樣。請注意，此報告只有在 **CONFIG_DEBUG_BLK_CGROUP=y** 設置於系統上的情況下，才可使用。

blkio.group_wait_time

回報控制群組在等待它其中一個佇列時，所耗費的時間總長（十億分之一秒 – ns）。每當此控制群組的佇列取得了一個 **timeslice** 時，報告便會被更新，因此若您在控制群組等待 **timeslice** 時讀取此偽檔案，報告將不會包含等待目前排程的作業所耗費的時間。請注意，此報告只有在 **CONFIG_DEBUG_BLK_CGROUP=y** 設置於系統上的情況下，才可使用。

blkio.empty_time

回報控制群組在無任何等待處理的請求的情況下，所耗費的時間總長（十億分之一秒 – ns）。每當此控制群組的佇列含有等待處理的請求時，報告便會被更新，因此若您在控制群組沒有等待處理的請求時讀取此偽檔案，報告將不會包含在目前空白狀態下所耗費的時間。請注意，此報告只有在 **CONFIG_DEBUG_BLK_CGROUP=y** 設置於系統上的情況下，才可使用。

blkio.idle_time

回報控制群組的排程器，預期比已在佇列中或來自於其它群組更佳的請求，所耗費的閒置時間總長（單位為十億分之一秒 – ns）。每當群組已不再閒置時，報告便會被更新，因此若您在控制群組閒置時讀取此偽檔案，報告將不會包含在目前閒置狀態下所耗費的時間。請注意，此報告只有在 **CONFIG_DEBUG_BLK_CGROUP=y** 設置於系統上的情況下，才可使用。

blkio.dequeue

回報控制群組的 I/O 作業請求，被特定裝置清除佇列的次數。項目含有三個欄位：*major*、*minor*，以及 *number*。*major* 和 *minor* 代表指定於 *Linux Allocated Devices* 中的裝置類型與節點編號，而 *number* 則代表群組請求被清除佇列的次數。請注意，此報告只有在 **CONFIG_DEBUG_BLK_CGROUP=y** 設置於系統上的情況下，才可使用。

blkio.reset_stats

重設紀錄在其它偽檔案中的數據。將一個整數寫入此檔案中，以重設此 **cgroup** 的數據。

3.2. CPU

cpu 子系統會排程 CPU，以存取控制群組。存取 CPU 資源會根據以下參數來排程，每個參數都位於控制群組虛擬檔案系統的獨立 **pseudofile**（偽檔案）裡：

cpu.shares

這整數值表示控制群組裡，任務可以使用的相對 CPU 時間。舉例來說，在兩個控制群組中的任務之 **cpu.shares** 都設為 **1** 的話，這兩個任務會收到均等的 CPU 時間；但其中一個任務的 **cpu.shares** 設為 **2** 的話，那它收到的 CPU 時間會是 **cpu.shares** 設為 **1** 的兩倍。

cpu.rt_runtime_us

這表示一段時間，單位為微秒（microsecond，簡寫為 **μs**，在此以「us」表示），這是控制群組中的任務存取 CPU 資源的最長連續時間。建立這個限制可以防止控制群組中的任務獨占 CPU 時間。如果控制群組中的任務要每五秒存取 CPU 資源四秒，請將 **cpu.rt_runtime_us** 設為 **4000000**，並把 **cpu.rt_period_us** 設為 **5000000**。

cpu.rt_period_us

這表示一段時間，單位為微秒（microsecond，簡寫為 **μs**，在此以「us」表示），這是控制群組存取 CPU 資源所該分配的頻率。如果控制群組中的任務要每五秒存取 CPU 資源四秒，請將 **cpu.rt_runtime_us** 設為 **4000000**，並把 **cpu.rt_period_us** 設為 **5000000**。

3.3. CPUACCT

CPU 計算 (**cpuacct**) 子系統會自動產生關於控制群組中的任務，CPU 資源使用上的報表。報表類型有三種：

cpuacct.stat

回報此控制群組以及其子群組在使用者模式與系統 (**kernel**) 模式中的工作，所耗費的 CPU 週期數目 (單位為系統上的 **USER_HZ** 所定義)。

cpuacct.usage

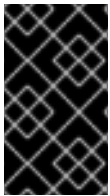
這會回報此控制群組的所有任務 (包括階層較低的任務) 所消耗的總 CPU 時間 (單位為毫秒)。

cpuacct.usage_percpu

這會回報此控制群組的所有任務 (包括階層較低的任務) 對每顆 CPU 消耗的 CPU 時間 (單位為毫秒)。

3.4. CPuset

cpuset 子系統會指派每個 CPU 與記憶體節點至控制群組。每個 **cpuset** 都可以根據以下參數來指定，在控制群組虛擬檔案系統裡，每個參數都有獨立的「*pseudofile*」(偽檔案)：



重要

在您將任務移到使用了任何子系統的 **cgroup** 之前，您必須設定這些子系統的一些必要參數。舉例來說，在將任務移到使用 **cpuset** 子系統的 **cgroup** 之前，**cpuset.cpus** 與 **cpuset.mems** 參數必須為此 **cgroup** 定義。

cpuset.cpus (必要)

這會指定此控制群組的任務所允許存取的 CPU。這是 ASCII 格式、以逗號隔開的清單，減號 (-) 用來區分範圍。例如：

```
0-2,16
```

代表第 0、1、2、與 16 個 CPU

cpuset.mems (必要)

指定此控制群組的任務所允許存取的記憶體節點。這是 ASCII 格式、以逗號隔開的清單，減號 (-) 用來區分範圍。例如：

```
0-2,16
```

表示第 0、1、2 與 16 的記憶體節點。

cpuset.memory_migrate

這包含了一個旗標值 (0 或 1)，在 **cpuset.mems** 改變時，是否要把記憶體中的分頁轉移到新的節點上。預設上，記憶體轉移是停用的 (0)，分頁會留在一開始分配的節點上，即使這個節點已經不再是 **cpuset.mems** 所指定的節點之一。如果啟用的話 (1)，那麼系統就會把分頁轉移至

cpuset.mems 所指定的參數之記憶體分頁上，如果可能的話，會保留其相對位置：舉例來說，清單上第二個節點的分頁最開始是由 **cpuset.mems** 所指定，它會被分配到 **cpuset.mems** 所指定的清單之第二個節點上（如果空間允許的話）。

cpuset.cpu_exclusive

這包含了旗標值（**0** 或 **1**），指定其它 **cpuset** 與其父子是否能分享指定給此 **cpuset** 的處理器。就預設值（**0**），CPU 不會特別只分配給一個 **cpuset**。

cpuset.mem_exclusive

這包含了旗標值（**0** 或 **1**），指定其它 **cpuset** 是否可以分享指定給這個 **cpuset** 的記憶體節點。預設上（**0**）記憶體節點是不會專門分配給一個 **cpuset**。專為 **cpuset** 保留記憶體節點（**1**）的功用，與使用 **cpuset.mem_hardwall** 啟用記憶體 **hardwall** 是一樣的。

cpuset.mem_hardwall

這包含了旗標值（**0** 或 **1**），指定 **kernel** 分配記憶體分頁與緩衝區資料時，是否要限制在指定給此 **cpuset** 的記憶體節點。預設上（**0**），分頁與緩衝區資料會由屬於多個使用者的程序所共享。啟用 **hardwall** 之後（**1**），每個任務的使用者分配都會被區分開來。

cpuset.memory_pressure

這是一個唯讀檔案，包含此 **cpuset** 的程序所建立的 *memory pressure*（記憶體壓力）的執行平均值。這個偽檔案的值會在 **cpuset.memory_pressure_enabled** 啟用時自動更新；若沒啟用的話，這個偽檔案的值會是 **0**。

cpuset.memory_pressure_enabled

這包含了旗標值（**0** 或 **1**），指定系統是否要運算這個控制群組的程序所建立的 *memory pressure*（記憶體壓力）。計算後的值會寫至 **cpuset.memory_pressure**，並且呈現程序試圖釋放記憶體的速率，計算方式是每秒試圖重新取得記憶體的數目，乘以 1,000。

cpuset.memory_spread_page

這包含了一個旗標值（**0** 或 **1**），指定檔案系統緩衝區是不是要散佈於分配給這個 **cpuset** 的記憶體節點。預設值（**0**）表示不把這些記憶體分頁平均分配給緩衝區，同時緩衝區都會放在建立緩衝區的程序所處的同一個節點上。

cpuset.memory_spread_slab

這包含了旗標值（**0** 或 **1**），指定 **kernel slab** 是不是要把快取檔案 I/O 的運作，平均分配到 **cpuset** 去。預設值（**0**）表示不平均分配 **kernel slab** 快取，而且 **slab** 快取會放在建立快取的程序所在的同樣節點上。

cpuset.sched_load_balance

包含了旗標值（**0** 或 **1**），指定 **kernel** 是不是要對此 **cpuset** 中的 CPU 進行負載平衡。預設值（**1**）會進行負載平衡，把負載過重的 CPU 的程序移到負載較輕的 CPU 上。

然而請注意，若在任何父控制群組中啟用了負載平衡，在控制群組中設定此旗標將不會造成影響，因為負載平衡已在更高的層級中進行了。因此，若要停用控制群組中的負載平衡，您也必須停用它在層級中，其所有 **parent** 的負載平衡。在此情況下，您亦必須考量是否要啟用該控制群組同層級的負載平衡。

cpuset.sched_relax_domain_level

這包含了介於 **-1** 到一個小的正整數之間，表示 **kernel** 進行負載平衡的 **CPU** 範圍。如果 **cpuset.sched_load_balance** 停用的話，這個值就沒有任何意義。

這個值的精確意義與系統架構有關；但以下是幾個典型的值：

cpuset.sched_relax_domain_level 的值

值	效用
-1	系統預設值，用於負載平衡
0	不要立即進行負載平衡；只有在定期的時候進行
1	立即對同樣核心的所有執行續進行負載平衡
2	立即對同樣套件的所有核心進行負載平衡
3	立即對同樣節點或刀鋒伺服器的所有 CPU 進行負載平衡
4	立即對 NUMA 架構的多個 CPU 進行負載平衡
5	立即對 NUMA 架構的所有 CPU 進行負載平衡

3.5. DEVICES

devices 子系統能允許或拒絕控制群組中的任務存取裝置。



重要

裝置的白名單 (**devices**) 子系統被視為 **Red Hat Enterprise Linux 6** 中的技術預覽。

技術預覽功能目前在 **Red Hat Enterprise Linux 6** 訂閱服務中不受支援、可能無法完整運作，並且一般並不適合使用於生產環境。然而，為了方便使用者，**Red Hat** 在作業系統中包含了這些功能，並為這項功能提供了更多的曝光率。您也許會在非生產的環境下發現這些功能非常有幫助，您亦可在技術預覽功能受到完整支援之前，提供意見和功能上的建議。

devices.allow

這指定了控制群組中的任務可以存取哪些裝置？每個條目都包含四個欄位：**type**、**major**、**minor**、以及 **access**。**type**、**major**、與 **minor** 欄位的值對應 **Linux** 已分配的裝置，又稱 **Linux** 裝置清單，可以在 <http://www.kernel.org/doc/Documentation/devices.txt> 找到。

type

type (類型) 可以是以下三者之一：

- **a**：套用到所有裝置，包括「字元裝置」與「區塊裝置」
- **b**：使用區塊裝置

- **c**：使用字元裝置

major, minor

major (主要的) 與 *minor* (次要的) 是 *Linux* 已分配的裝置所指定的裝置節點編號。主要與次要編號會以冒號隔開。舉例來說，如果 **8** 是某個 **SCSI** 硬碟的主要編號，而 **1** 是第一組 **SCSI** 硬碟的第一個分割區，那麼 **8:1** 就完全表明了這個分割區，對應到 **/dev/sda1** 裡的一個檔案系統位置。

***** 表示所有的主要與次要裝置節點，舉例來說，**9:*** (所有 **RAID** 裝置) 或 ***:*** (所有裝置)。

access

access (存取) 是以下一或多的字母：

- **r**：允許任務讀取特定裝置
- **w**：允許任務寫入特定裝置
- **m**：允許任務建立不存在的裝置

舉例來說，**access** 指定為 **r** 的時候，任務就只能讀取裝置；但當 **access** 指定為 **rw** 時，任務就可以對裝置進行讀、寫。

devices.deny

指定控制群組中的任務不能存取的裝置。這裡的語法跟 **devices.allow** 完全一樣。

devices.list

回報在此控制群組中，已針對於工作設置了存取控制的裝置。

3.6. FREEZER

freezer 子系統可中止或復原控制群組中的工作。

freezer.state

freezer.state 能擁有三個不同的值：

- **FROZEN** – 控制群組中的工作將被中止。
- **FREEZING** – 系統正在中止控制群組中的工作。
- **THAWED** – 控制群組中的工作已復原。

要暫停某個特定的程序：

1. 將該程序移動到擁有 **freezer** 子系統所連接之層級的 **cgroup** 裡。
2. 凍結該特定的 **cgroup**，以暫停其中的程序。

把程序移動到以暫停（凍結）的 **cgroup** 是不可能的。

請注意，**FROZEN** 和 **THAWED** 這兩個值可寫入 **freezer.state**，**FREEZING** 則是唯讀，無法寫入。

3.7. MEMORY

memory（記憶體）子系統會根據使用於控制群組中的工作的記憶體資源，自動產生記憶體報告，然後設定這些工作所能使用的記憶體限制：

memory.stat

回報廣泛的記憶體數據，如以下表格所述：

表格 3.1. **memory.stat** 所回報的值

數據	描述
cache	分頁快取，包括 tmpfs (shmem)，單位為位元組
rss	匿名與 swap 快取，不包含 tmpfs (shmem)，單位為位元組
mapped_file	記憶體映射的已映射檔案之大小，包括 tmpfs (shmem) 單位為位元組
pgpgin	分至記憶體中的分頁數量
pgpgout	由記憶體中分出的分頁數量
swap	swap 使用量，單位為位元組
active_anon	啟用中、最近最少使用的 (LRU) 清單上的匿名，以及 swap 快取記憶體，包括 tmpfs (shmem)，單位為位元組
inactive_anon	未啟用的 LRU 清單上的匿名與 swap 快取，包括 tmpfs (shmem)，單位為位元組
active_file	啟用中的 LRU 清單上的檔案支援記憶體，單位為位元組
inactive_file	非啟用中 LRU 清單上的檔案支援記憶體，單位為位元組
unevictable	無法重新收回的記憶體，單位為位元組
hierarchical_memory_limit	包含了 memory cgroup 的階層記憶體限制，單位為位元組
hierarchical_memsw_limit	記憶體加上包含了 memory cgroup 的階層的 swap 限制，單位為位元組

此外，除了 **hierarchical_memory_limit** 和 **hierarchical_memsw_limit** 之外，這些檔案皆有對應的前綴 **total_**，它不僅會回報控制群組，還會回報它的所有子群組。比方說，**swap** 會回報控制群組的 **swap** 使用量，而 **total_swap** 則會回報控制群組和其所有子群組的 **swap** 使用量。

當您詮釋 **memory.stat** 所回報的值時，請注意各數據如何相互關聯：

- **active_anon + inactive_anon = anonymous memory + tmpfs 的檔案快取 + swap 快取**

因此，`active_anon + inactive_anon ≠ rss`，因為 `rss` 不包含 `tmpfs`。

- `active_file + inactive_file = cache - tmpfs` 的大小

memory.usage_in_bytes

回報控制群組中，程序目前所使用的記憶體使用量總數（位元組）。

memory.memsw.usage_in_bytes

回報控制群組中，程序目前所使用的記憶體數量與 `swap` 空間的總和（單位為位元組）。

memory.max_usage_in_bytes

回報控制群組中，程序所使用的最大記憶體數量（單位為位元組）。

memory.memsw.max_usage_in_bytes

回報控制群組中，程序所使用的最大記憶體數量與 `swap` 空間（單位為位元組）。

memory.limit_in_bytes

設置最大的使用者記憶體數量（包括檔案快取）。若沒有指定單位的話，數值將會被解譯為位元組。然而，您可透過使用字尾來指定較大的單位 – `k` 或 `K` 代表 KB，`m` 或 `M` 代表 MB，以及 `g` 或 `G` 則代表 GB。

您無法使用 `memory.limit_in_bytes` 來限制 `root` 控制群組；您只可將值套用至階層中，處於較低階層的群組中。

將 `-1` 寫入 `memory.limit_in_bytes`，以移除任何既有的限制。

memory.memsw.limit_in_bytes

設置最大的記憶體總和與 `swap` 使用量。若沒有指定單位的話，數值將會被解譯為位元組。然而，您可透過使用字尾來指定較大的單位 – `k` 或 `K` 代表 KB，`m` 或 `M` 代表 MB，以及 `g` 或 `G` 則代表 GB。

您無法使用 `memory.memsw.limit_in_bytes` 來限制 `root` 控制群組；您只可將值套用至階層中，處於較低階層的群組中。

將 `-1` 寫入 `memory.memsw.limit_in_bytes`，以移除任何既有的限制。

memory.failcnt

回報記憶體限制超過 `memory.limit_in_bytes` 中所設置的值的次數。

memory.memsw.failcnt

回報記憶體加上 `swap` 空間限制超過 `memory.memsw.limit_in_bytes` 中所設置的值的次數。

memory.force_empty

當設為 `0` 時，會將此控制群組中的工作所使用的所有分頁記憶體清空。此介面只能在控制群組沒有工作時才可使用。若記憶體無法釋放的話，它便會視情況被移至父控制群組中。請在移除控制群組之前，使用 `memory.force_empty` 來避免將未使用的分頁快取移至其父控制群組中。

memory.swappiness

設置 `kernel` 的傾向特性，以將使用於此控制群組中的程序的記憶體換出，而不是由分頁快取收回分頁。這和為系統設置於 `/proc/sys/vm/swappiness` 中的特性相同，計算方式也相同。預設值為 `60`。

若使用的值比 **60** 還要低的話，將會降低 **kernel** 換出程序記憶體的分頁，大於 **60** 的值會提高 **kernel** 換出程序記憶體的分頁，並且當值大於 **100** 時，**kernel** 將會開始換出此控制群組中，屬於程序的位址空間一部分的分頁。

請注意，**0** 這個值無法避免程序記憶體被切換出；當系統記憶體不足時，換出這個動作還是有可能會發生，因為全域虛擬記憶體管理邏輯不會讀取 **cgroup** 的值。若要完全地封鎖分頁，請使用 **mlock()** 來代替 **cgroup**。

您不可更改下列群組的 **swappiness**：

- **root** 控制群組，使用了設置於 **/proc/sys/vm/swappiness** 中的 **swappiness**。
- 一個底下含有子群組的控制群組。

memory.use_hierarchy

包含一個指定記憶體使用量是否應被記入控制群組階層中的旗標 (**0** 或 **1**)。若啟用的話 (**1**)，記憶體控制器將會由超過了它記憶體限制的程序和子程序收回記憶體。就預設值 (**0**) 來說，控制器不會由一項工作的子程序收回記憶體。

3.8. NET_CLS

net_cls 子系統會以一個 **class** 標識符號 (**classid**) 來標記網路封包，這能讓 **Linux** 流量控制器 (**tc**) 辨識源自於特定控制群組的封包。流量控制器能被配置來指定不同的優先順序給來自於不同控制群組的封包。

net_cls.classid

net_cls.classid 包含了一個格式為十六進位的單獨數值，它顯示了流量控制控點 (*handle*)。比方說，**0x1001** 代表了照慣例寫成 **10:1** 的控點。

這些控點的格式為：**0xAAAABBBB**，**AAAA** 為十六進位的 **major** 數字，而 **BBBB** 則為十六進位的 **minor** 數字。您可忽略掉前綴的零；**0x10001** 和 **0x00010001** 是相同的，並且代表 **1:1**。

欲學習如何配置流量控制器，以使用 **net_cls** 附加至網路封包的 **handle**，請參閱 **tc** 的 **man page**。

3.9. NS

ns 子系統提供了將程序分組為個別命名空間 (*namespace*)。在特定命名空間中，程序可互相進行互動，不過卻會與在其它命名空間中執行的程序隔離。當使用於作業系統層級的虛擬化時，這些個別的命名空間有時亦稱為容器 (*container*)。

3.10. 額外資源

系統特屬的 kernel 文件

下列所有檔案皆位於 **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups/** 目錄中。

- **blkio** 子系統 – **blkio-controller.txt**
- **cpuacct** 子系統 – **cpuacct.txt**

- **cpuset** 子系統 – **cpuset.txt**
- **devices** 子系統 – **devices.txt**
- **freezer** 子系統 – **freezer-subsystem.txt**
- **memory** 子系統 – **memory.txt**

附錄 A. 修訂紀錄

修訂 1-5.400 Rebuild with publican 4.0.0	2013-10-31	Rüdiger Landmann
修訂 1-5 Rebuild for Publican 3.0	2012-07-18	Anthony Towns
修訂 1.0-5 《資源管理指南》的 Red Hat Enterprise Linux 6.1 GA 版。	Thu May 19 2011	Martin Prpič
修訂 1.0-4 修正多個範例 – BZ#667623 、 BZ#667676 、 BZ#667699 。 闡明 <code>cgclear</code> 指令 – BZ#577101 。 闡明 <code>lssubsystem</code> 指令 – BZ#678517 。 凍結一個進程 – BZ#677548 。	Tue Mar 1 2011	Martin Prpič
修訂 1.0-3 修正了 <code>remount</code> 的範例 – BZ#612805	Wed Nov 17 2010	Rüdiger Landmann
修訂 1.0-2 移除預發行意見的指示	Thu Nov 11 2010	Rüdiger Landmann
修訂 1.0-1 來自於 QE 的修正 – BZ#581702 and BZ#612805	Wed Nov 10 2010	Rüdiger Landmann
修訂 1.0-0 GA 的完整功能版本	Tue Nov 9 2010	Rüdiger Landmann