Red Hat Enterprise Linux 7

存储管理指南

在 RHEL 7 中部署和配置单节点存储

Last Updated: 2024-04-26

Red Hat Enterprise Linux 7 存储管理指南

在 RHEL 7 中部署和配置单节点存储

Milan Navrátil
Red Hat Customer Content Services

Jacquelynn East
Red Hat Customer Content Services

Don Domingo
Red Hat Customer Content Services

编辑

Marek Suchánek
Red Hat Customer Content Services
msuchane@redhat.com

Apurva Bhide
Red Hat Customer Content Services
abhide@redhat.com

法律通告

摘要

本指南提供了如何有效管理 Red Hat Enterprise Linux 7 上的存储设备和文件系统的说明。它供具有基本了解 Red Hat Enterprise Linux 或 Fedora 的系统管理员使用。

目录

第 1 章 概述

存储管理指南包含有关 Red Hat Enterprise Linux 7 中支持的文件系统和数据存储功能的大量信息。本书旨在为管理单节点（即非群集）存储解决方案的管理员提供快速参考。

存储管理指南被分成以下部分： 文件系统、存储管理和数据重复数据删除以及使用 VDO 进行数据压缩。

文件系统部分详细介绍了 Red Hat Enterprise Linux 7 支持的各种文件系统。它描述了它们，并解释了如何最好地利用它们。

存储管理部分详细介绍了 Red Hat Enterprise Linux 7 支持的各种工具和存储管理任务。它描述了它们，并解释了如何最好地利用它们。

VDO 的数据重复数据删除和压缩部分描述了 Virtual Data Optimizer (VDO)。它解释了如何使用 VDO 来减少您的存储要求。

1.1. RED HAT ENTERPRISE LINUX 7 中的新功能和增强功能

Red Hat Enterprise Linux 7 具有以下文件系统改进：

不包括 eCryptfs

从 Red Hat Enterprise Linux 7 开始，不包括 eCryptfs。有关加密文件系统的更多信息，请参阅红帽的安全指南。

系统存储管理器

Red Hat Enterprise Linux 7 包括一个名为 System Storage Manager 的新应用程序，它提供了一个命令行界面来管理各种存储技术。如需更多信息，请参阅 第 16 章 *系统存储管理器(SSM)*。

XFS 是默认文件系统

从 Red Hat Enterprise Linux 7 开始，XFS 是默认的文件系统。有关 XFS 文件系统的详情，请参考 第 3 章 *XFS 文件系统*。

文件系统重组

Red Hat Enterprise Linux 7 引入了一个新的文件系统结构。目录 **/bin**、**/sbin**、**/lib** 和 **/lib64** 现在嵌套在 **/usr** 下。

snapper

Red Hat Enterprise Linux 7 引入了一个名为 Snapper 的新工具，它可以方便地创建和管理 LVM 和 Btrfs 的快照。如需更多信息，请参阅 第 14 章 *使用 Snapper 创建和维护快照*。

Btrfs（技术预览）

注意

Btrfs 在 Red Hat Enterprise Linux 7 中作为技术预览功能提供，但自 Red Hat Enterprise Linux 7.4 发行版本起已被弃用。它将在以后的 Red Hat Enterprise Linux 主发行版本中删除。

如需更多信息，请参阅 Red Hat Enterprise Linux 7.4 发行注记中的 已弃用的功能。

Btrfs 是一个本地文件系统，旨在提供更好的性能和可扩展性，包括集成的 LVM 操作。红帽不完全支持这个文件系统，因此是一个技术预览。有关 Btrfs 的详情，请参考 第 6 章 *Btrfs（技术预览）*。

NFSv2 没有支持长

从 Red Hat Enterprise Linux 7 开始，不再支持 NFSv2。

部分 I. 文件系统

文件系统部分提供有关红帽完全支持的文件系统结构和维护、Btrfs 技术预览和文件系统的信息：ext3、ext4、GFS2、XFS、NFS 和 FS-Cache。

注意

Btrfs 在 Red Hat Enterprise Linux 7 中作为技术预览功能提供，但自 Red Hat Enterprise Linux 7.4 发行版本起已被弃用。它将在以后的 Red Hat Enterprise Linux 主发行版本中删除。

如需更多信息，请参阅 Red Hat Enterprise Linux 7.4 发行注记中的 已弃用的功能。

有关 Red Hat Enterprise Linux 文件系统和存储限制的概述，请参阅红帽知识库中的 Red Hat Enterprise Linux 技术功能和限制。

XFS 是 Red Hat Enterprise Linux 7 和 Red Hat 中的默认文件系统，红帽建议您使用 XFS，除非您有强大的原因来使用另一个文件系统。有关常见文件系统及其属性的常规信息，请参阅以下红帽知识库文章： 如何选择您的 Red Hat Enterprise Linux 文件系统。

第 2 章 文件系统结构和维护

文件系统结构是操作系统中最基本的组织级别。操作系统与其用户、应用程序和安全模型交互的方式几乎始终取决于操作系统如何在存储设备上组织文件。提供通用文件系统结构可确保用户和程序能够访问和写文件。

文件系统将文件划分为两个逻辑类别：

可共享和不可取的文件

可以在本地和远程主机访问 *可共享* 文件。*Unsharable* 文件仅在本地可用。

变量和静态文件

变量文件 （如文档）可以随时更改。*静态文件* （如二进制文件）不会在没有系统管理员的操作的情况下改变。

以这种方式分类文件有助于将每个文件的功能与分配给保存它们的目录的权限相关联。操作系统及其用户与文件交互的方式决定了放置文件的目录、该目录是以只读还是读写权限挂载的，以及每个用户对该文件的访问级别。该组织的最高层至关重要；对底层目录的访问可能会受到限制，否则，如果从最高层向下的访问规则不遵循严格的结构，就会出现安全问题。

2.1. 文件系统层次结构标准(FHS)概述.

Red Hat Enterprise Linux 使用 *文件系统层次结构标准* (*FHS*)文件系统结构，它定义了许多文件类型和目录的名称、位置和权限。

FHS 文档是任何符合 FHS 的文件系统的权威参考，，但该标准留下了许多未定义或可扩展的领域。本节概述了该标准，并描述了该标准未涵盖的文件系统部分。

FHS 合规的两个最重要的元素是：

- 与其他符合 FHS 的系统兼容
- 能够以只读形式挂载 **/usr/** 分区。这很重要，因为 **/usr/** 包含常见的可执行文件，用户不应更改。此外，由于 **/usr/** 以只读形式挂载，它应该可以从 CD-ROM 驱动器或通过只读 NFS 挂载从另一台机器挂载。

2.1.1. FHS 组织

此处记下的目录和文件是 FHS 文档指定的目录和文件的一个子集。有关最完整的信息，请查看最新的 FHS 文档 http://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf; file-hierarchy(7) man page 还提供了概述。

注意

可用的目录取决于任意给定系统上安装的内容。以下列表只能是可以找到的内容的示例。

2.1.1.1. 收集文件系统信息

df 命令

df 命令报告系统的磁盘空间使用情况。其输出类似如下：

例 2.1. df 命令输出

```
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                      11675568   6272120   4810348  57% / /dev/sda1
                        100691      9281     86211  10% /boot
none                    322856         0    322856   0% /dev/shm
```

默认情况下，**df** 以 1 KB 块为单位显示分区大小，以及以 KB 为单位的已用和可用磁盘空间。要以 MB 和 GB 为单位查看信息，请使用命令 **df -h**。**h** 参数代表"人类可读的"格式。**df -h** 的输出类似如下：

例 2.2. df -h 命令输出

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                       12G  6.0G  4.6G  57% / /dev/sda1
  99M  9.1M   85M  10% /boot
none    316M     0  316M   0% /dev/shm
```

注意

在给定示例中，挂载的分区 **/dev/shm** 代表系统的虚拟内存文件系统。

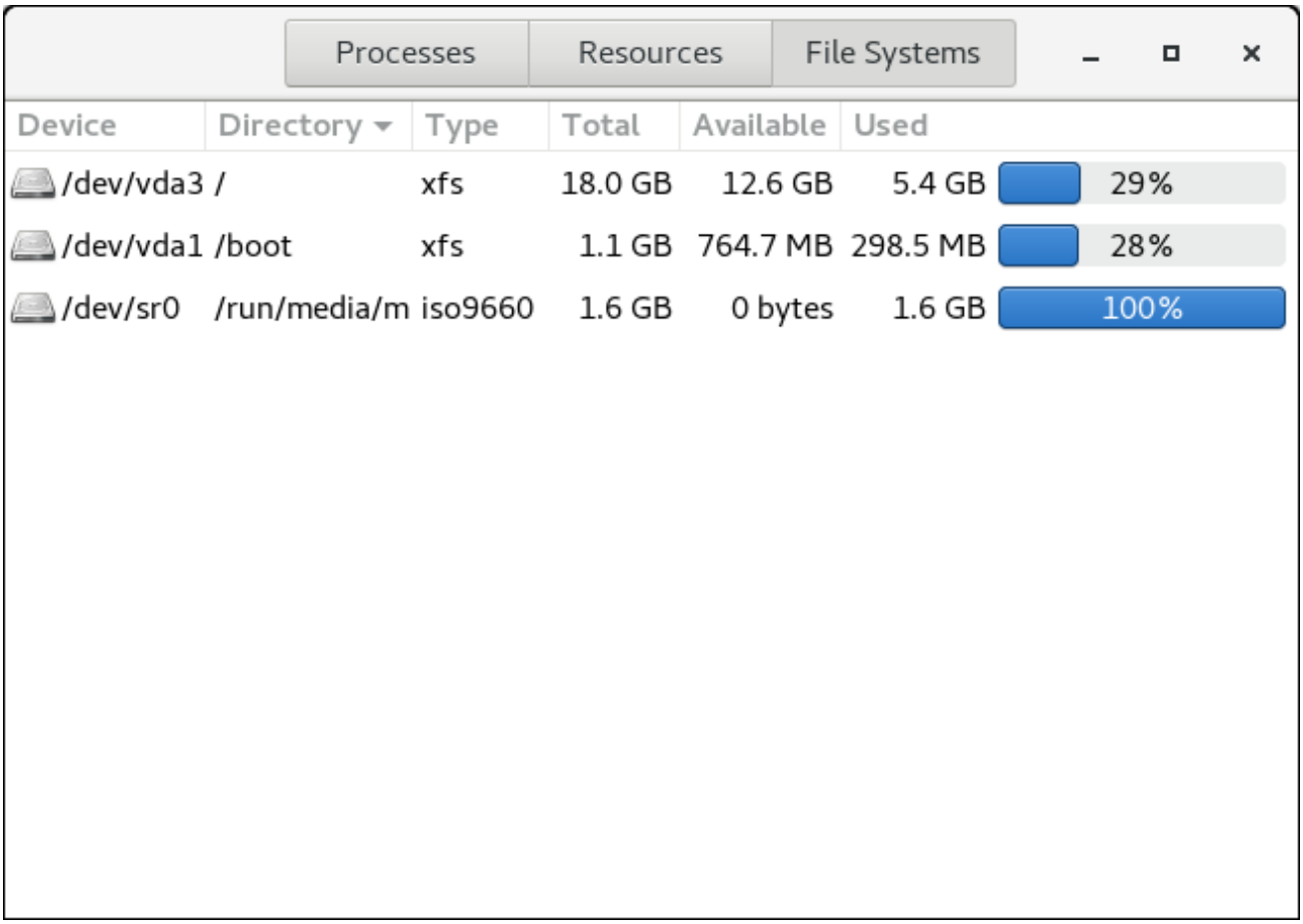
DU 命令

du 命令显示目录中文件使用的总空间量，显示每个子目录的磁盘使用情况。**du** 输出中的最后一行显示目录的总磁盘使用情况。要仅查看目录的总磁盘使用情况（以人类可读的格式），请使用 **du -hs**。有关更多选项，请参阅 **man du**。

GNOME 系统监控器

要以图形格式查看系统的分区和磁盘空间使用情况，请点击 **Applications → Tools System** Monitor 或使用 **gnome-system-monitor** 命令，使用 Gnome **System Monitor**。选择 **File Systems** 选项卡来查看系统的分区。下图展示了 **File Systems** 选项卡。

图 2.1. GNOME 系统监控器中的文件系统选项卡



[D]

2.1.1.2. /boot/ 目录

/boot/ 目录包含引导系统所需的静态文件，例如 Linux 内核。这些文件对于系统正常启动至关重要。

> **警告**
>
> 不要删除 **/boot/** 目录。这样做会使系统无法启动。

2.1.1.3. /dev/ 目录

/dev/ 目录包含代表以下设备类型的设备节点：

- 连接到系统的设备；
- 内核提供的虚拟设备.

这些设备节点对于系统正常工作至关重要。**udevd** 守护进程根据需要在 **/dev/** 中创建和删除设备节点。

/dev/ 目录和子目录中的设备定义为 *字符*（仅提供输入和输出的串行流，如鼠标或键盘）或 *块*（随机访问，如硬盘驱动器或软盘驱动器）。如果安装了 GNOME 或 KDE，则连接（如使用 USB）或插入（如 CD 或者 DVD 驱动器）时会自动检测一些存储设备，并显示其内容的弹出窗口。

表 2.1. /dev 目录中通用文件的示例

File	描述
/dev/hda	主 IDE 通道上的主设备。
/dev/hdb	主 IDE 通道上的从设备.
/dev/tty0	第一个虚拟控制台.
/dev/tty1	第二个虚拟控制台.
/dev/sda	主 SCSI 或 SATA 通道中的第一个设备。
/dev/lp0	第一个并行端口。

有效的块设备可以是两种类型的条目之一：

映射设备

卷组中的逻辑卷，例如 **/dev/mapper/VolGroup00-LogVol02**。

静态设备

传统存储卷，例如 **/dev/*sdbX***，其中 *sdb* 是存储设备名称，*X* 是分区号。**/dev/sdb*X*** 也可以是 **/dev/disk/by-id/*WWID***，或 **/dev/disk/by-uuid/*UUID***。如需更多信息，请参阅 第 25.8 节 “持久性命名”。

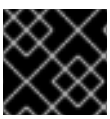
2.1.1.4. /etc/ 目录

/etc/ 目录保留机器本地的配置文件。它不应包含任何二进制文件；如果存在二进制文件，请将其移到 **/usr/bin/** 或 **/usr/sbin/**。

例如，**/etc/skel/** 目录存储"框架"用户文件，这些文件用于在首次创建用户时填充主目录。应用也将其配置文件存储在此目录中，并且在执行时可能会引用这些文件。**/etc/exports** 文件控制哪些文件系统导出到远程主机。

2.1.1.5. /mnt/ 目录

为临时挂载的文件系统保留 **/mnt/** 目录，如 NFS 文件系统挂载。对于所有可移动存储介质，请使用 **/media/** 目录。自动检测到的可移动介质挂载在 **/media** 目录中。



重要

安装程序不能使用 **/mnt** 目录。

2.1.1.6. /opt/ 目录

/opt/ 目录通常为不属于默认安装的软件和附加软件包保留。安装到 **/opt/** 的软件包会创建一个目录，例如，**/opt/*packagename*/**。在大多数情况下，此类软件包遵循可预测的子目录结构；大多数软件包将其二进制文件存储在 **/opt/*packagename*/bin/** 中，其 **man** page 存储在 **/opt/*packagename*/man/** 中。

2.1.1.7. /proc/ 目录

/proc/ 目录包含从内核中提取信息或向其发送信息的特殊文件。此类信息的示例包括系统内存、CPU 信息和硬件配置。有关 **/proc/** 的详情，请参考 第 2.3 节 "/proc 虚拟文件系统"。

2.1.1.8. /srv/ 目录

/srv/ 目录包含 Red Hat Enterprise Linux 系统提供的特定于站点的数据。此目录为用户提供特定服务(如 FTP、WWW 或 CVS)的数据文件的位置。仅与特定用户相关的数据应位于 **/home/** 目录中。

2.1.1.9. /sys/ 目录

/sys/ 目录利用特定于内核的新 **sysfs** 虚拟文件系统。随着对内核中热插硬件设备的支持，**/sys/** 目录包含与 **/proc/** 持有的信息相似，但显示特定于热插拔设备的设备信息的层次结构视图。

2.1.1.10. /usr/ 目录

/usr/ 目录用于可在多台机器之间共享的文件。**/usr/** 目录通常位于其自己的分区上，并以只读形式挂载。至少，**/usr/** 应该包含以下子目录：

/usr/bin

此目录用于二进制文件。

/usr/etc

此目录用于系统范围的配置文件。

/usr/games

此目录存储游戏。

/usr/include

此目录用于 C 头文件。

/usr/kerberos

此目录用于与 Kerberos 相关的二进制文件和文件。

/usr/lib

此目录用于设计不是由 shell 脚本或用户直接使用的对象文件和库。

从 Red Hat Enterprise Linux 7.0 开始，**/lib/** 目录已与 **/usr/lib** 合并。现在，它还包含在 **/usr/bin/** 和 /usr/ **sbin/** **中执行二进制文件所需的**库。这些共享库镜像用于引导系统或执行根文件系统中的命令。

/usr/libexec

此目录包含其他程序调用的小型帮助程序。

/usr/sbin

从 Red Hat Enterprise Linux 7.0 开始，**/sbin** 已移到 **/usr/sbin** 中。这意味着它包含所有系统管理二进制文件，包括启动、恢复、恢复或修复系统所必需的。**/usr/sbin/** 中的二进制文件需要 root 特权才能使用。

/usr/share

此目录存储不特定于架构的文件。

/usr/src

此目录存储源代码。

/usr/tmp 链接到 /var/tmp

此目录存储临时文件。

/usr/ 目录还应包含 **/local/** 子目录。根据 FHS，系统管理员在本地安装软件时会使用这个子目录，在系统更新时应该不会被覆盖。**/usr/local** 目录的结构与 **/usr/** 类似，包含以下子目录：

- **/usr/local/bin**
- **/usr/local/etc**
- **/usr/local/games**
- **/usr/local/include**
- **/usr/local/lib**
- **/usr/local/libexec**
- **/usr/local/sbin**
- **/usr/local/share**
- **/usr/local/src**

Red Hat Enterprise Linux 对 **/usr/local/** 的使用与 FHS 稍有不同。FHS 指出 **/usr/local/** 应该用于存储系统软件升级安全的软件。由于 **RPM Package Manager** 可以安全地执行软件升级，因此不需要通过将文件存储在 **/usr/local/** 中来保护文件。

相反，Red Hat Enterprise Linux 将 **/usr/local/** 用于机器本地的软件。例如，如果 **/usr/** 目录作为只读 NFS 共享从远程主机挂载，那么仍然可以将软件包或程序安装到 **/usr/local/** 目录下。

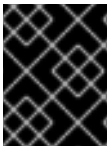
2.1.1.11. /var/ 目录

由于 FHS 要求 Linux 将 **/usr/** 挂载为只读，因此任何写入日志文件或需要 **spool/** 或 **lock/** 目录的程序都应将它们写入 **/var/** 目录。FHS 指出 **/var/** 用于变量数据，其中包括 spool 目录和文件、日志记录数据、临时和临时文件。

以下是在 **/var/** 目录中找到的一些目录：

- **/var/account/**
- **/var/arpwatch/**
- **/var/cache/**
- **/var/crash/**
- **/var/db/**
- **/var/empty/**

- **/var/ftp/**
- **/var/gdm/**
- **/var/kerberos/**
- **/var/lib/**
- **/var/local/**
- **/var/lock/**
- **/var/log/**
- 链接到 **/var/spool/mail/**的 **/var/mail**
- **/var/mailman/**
- **/var/named/**
- **/var/nis/**
- **/var/opt/**
- **/var/preserve/**
- **/var/run/**
- **/var/spool/**
- **/var/tmp/**
- **/var/tux/**
- **/var/www/**
- **/var/yp/**



重要

/var/run/media/*user* 目录包含用作可移动介质挂载点的子目录，如 USB 存储介质、DVD、CD-ROM 和 Zip 磁盘。请注意，**/media/** 目录用于此目的。

系统日志文件（如 **messages** 和 **lastlog** ）位于 **/var/log/** 目录中。**/var/lib/rpm/** 目录包含 RPM 系统数据库。锁定文件位于 **/var/lock/** 目录中，通常位于使用该文件的程序的目录中。**/var/spool/** 目录有存储某些程序数据文件的子目录。这些子目录包括：

- **/var/spool/at/**
- **/var/spool/clientmqueue/**
- **/var/spool/cron/**
- **/var/spool/cups/**
- **/var/spool/exim/**

- **/var/spool/lpd/**
- **/var/spool/mail/**
- **/var/spool/mailman/**
- **/var/spool/mqueue/**
- **/var/spool/news/**
- **/var/spool/postfix/**
- **/var/spool/repackage/**
- **/var/spool/rwho/**
- **/var/spool/samba/**
- **/var/spool/squid/**
- **/var/spool/squirrelmail/**
- **/var/spool/up2date/**
- **/var/spool/uucp/**
- **/var/spool/uucppublic/**
- **/var/spool/vbox/**

2.2. 特殊 RED HAT ENTERPRISE LINUX 文件位置

Red Hat Enterprise Linux 稍微扩展了 FHS 结构以适应特殊文件。

大多数与 RPM 相关的文件保存在 **/var/lib/rpm/** 目录中。有关 RPM 的更多信息，请参阅 **man rpm**。

/var/cache/yum/ 目录包含 **Package Updater** 使用的文件，包括系统的 RPM 标头信息。此位置还可用于在更新系统时临时存储下载的 RPM。有关红帽网络的详情请参考 https://rhn.redhat.com/。

特定于 Red Hat Enterprise Linux 的另一个位置是 **/etc/sysconfig/** 目录。该目录存储各种配置信息。许多在启动时运行的脚本都使用此目录中的文件。

2.3. /PROC 虚拟文件系统

与大多数文件系统不同，**/proc** 既不包含文本文件，也不包含二进制文件。由于它托管了 *虚拟文件*，**/proc** 被称为虚拟文件系统。这些虚拟文件的大小通常为零字节，即使它们包含大量信息。

/proc 文件系统不用于存储。它的主要用途是为硬件、内存、运行进程和其他系统组件提供基于文件的接口。可以通过查看相应的 **/proc** 文件，在多个系统组件上检索实时信息。还可以操作 **/proc** 中的某些文件（用户和应用程序）来配置内核。

以下 **/proc** 文件与管理和监控系统存储相关：

/proc/devices

显示当前配置的各种字符和块设备。

/proc/filesystems

列出内核当前支持的所有文件系统类型。

/proc/mdstat

包含系统上多个磁盘或 RAID 配置（如果存在）的当前信息。

/proc/mounts

列出系统当前使用的所有挂载。

/proc/partitions

包含分区块分配信息.

有关 **/proc** 文件系统的更多信息，请参阅 Red Hat Enterprise Linux 7 *部署指南*。

2.4. 丢弃未使用的块

批量丢弃和在线丢弃操作是挂载的文件系统的功能，它们丢弃文件系统未使用的块。它们可用于固态硬盘和精简配置的存储。

- *批量丢弃操作* 由用户使用 **fstrim** 命令显式运行。此命令丢弃文件系统中符合用户条件的所有未使用的块。
- *在线丢弃操作* 在挂载时指定，可以使用 **-o discard** 选项作为 **mount** 命令的一部分，也可以使用 **/etc/fstab** 文件中的 **discard** 选项。它们在没有用户干预的情况下实时运行。在线丢弃操作只丢弃从已使用转变为空闲的块。

从 Red Hat Enterprise Linux 6.4 开始，ext4 文件系统都支持这两种操作类型，以及 XFS 文件系统。另外，文件系统底层的块设备必须支持物理丢弃操作。如果 **/sys/block/*设备*/queue/discard_max_bytes** 文件中存储的值不为零，则支持物理丢弃操作。

如果您要对以下执行 **fstrim** 命令：

- 不支持丢弃操作的设备，或者
- 一个由多个设备组成的逻辑设备(LVM 或者 MD)，其中任何一个设备都不支持丢弃操作

将显示以下信息：

```
fstrim -v /mnt/non_discard
fstrim: /mnt/non_discard: the discard operation is not supported
```

注意

mount 命令允许您使用 **-o discard** 选项挂载不支持丢弃操作的设备。

红帽建议批量丢弃操作，除非系统的工作负载不允许批量丢弃，或者需要在线丢弃操作来保持性能。

详情请查看 fstrim(8) 和 mount(8) man page。

第 3 章 XFS 文件系统

XFS 是一个高度可扩展的、高性能的文件系统，它最初是由 Silicon Graphics, Inc 设计的。XFS 是 Red Hat Enterprise Linux 7 的默认文件系统。

XFS 的主要功能

- XFS 支持 *元数据日志*，这有助于更快地恢复崩溃。
- XFS 文件系统可以在挂载和激活时进行碎片整理和放大。
- 另外，Red Hat Enterprise Linux 7 支持特定于 XFS 的备份和恢复工具。

分配功能

XFS 具有以下分配方案：

- 基于数据块的分配
- 条带化分配策略
- 延迟分配
- 空间预分配

延迟分配和其他性能优化对 XFS 的影响与对 ext4 的影响一样。即，程序对 XFS 文件系统的写操作无法保证为磁盘上，除非程序随后发出 **fsync** （） 调用。

有关延迟分配对文件系统(ext4 和 XFS)的影响的更多信息，请参阅 分配功能 中的 第 5 章 *ext4 文件系统*。

注意

创建或扩展文件偶尔会失败，并显示意外的 ENOSPC 写入失败，即使磁盘空间似乎足够了。这是因为 XFS 的性能导向型设计。实际上，它不会成为问题，因为它只有在剩余空间只是几个块时才会发生。

其他 XFS 功能

XFS 文件系统还支持以下内容：

***扩展的属性* (xattr)**

这允许系统能够按文件关联多个额外的名称/值对。它会被默认启用。

配额日志

这可避免在崩溃后需要进行冗长的配额一致性检查。

项目/目录配额

这允许对目录树的配额限制。

小于秒的时间戳

这允许时间戳进入亚秒。

默认 atime 行为是 relatime

对于 **XFS**，relatime 默认为 on。与 **noatime** 相比，它几乎没有开销，同时仍维护 sane **atime** 值。

3.1. 创建 XFS 文件系统

- 要创建 XFS 文件系统，请使用以下命令：

```
# mkfs.xfs block_device
```

  - 使用到块设备的路径替换 *block_device*。例如：**/dev/sdb1**,**/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**, 或 **/dev/my-volgroup/my-lv**。
  - 一般情况下，默认选项是常见用途的最佳选择。
  - 在包含现有文件系统的块设备上使用 **mkfs.xfs** 时，添加 **-f** 选项来覆盖该文件系统。

例 3.1. mkfs.xfs 命令输出

以下是 **mkfs.xfs** 命令的输出示例：

```
meta-data=/dev/device            isize=256    agcount=4, agsize=3277258 blks
         =                       sectsz=512   attr=2
data     =                       bsize=4096   blocks=13109032, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=6400, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

注意

XFS 文件系统创建后，其大小不能缩小。但是，仍然可以使用 **xfs_growfs** 命令放大。如需更多信息，请参阅 第 3.4 节 “增加 XFS 文件系统的大小”。

条带块设备

对于条带块设备（如 RAID5 阵列），可以在创建文件系统时指定条带几何结构。使用合适的条带几何形状可显著提高 XFS 文件系统的性能。

在 LVM 或 MD 卷上创建文件系统时，**mkfs.xfs** 选择最佳的几何结构。对于将几何结构信息导出到操作系统的某些硬件 RAID，这可能也是如此。

如果设备导出条带几何结构信息，则 **mkfs** 工具（用于 ext3、ext4 和 xfs）将自动使用此 geometry。如果 **mkfs** 工具没有检测到条带几何结构，即使存储确实有条带几何结构，则可以在使用以下选项创建文件系统时手动指定它：

su=*value*

指定条带单位或 RAID 块大小。***该值必须*** 以字节为单位，使用可选的 **k**、**m** 或 **g** 后缀。

sw=*value*

指定 RAID 设备中数据磁盘的数量，或者条带中条带单位的数量。

以下示例对包含 4 个条带单位的 RAID 设备指定了 64k 的块大小：

```
# mkfs.xfs -d su=64k,sw=4 /dev/block_device
```

其它资源

有关创建 XFS 文件系统的更多信息，请参阅：

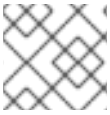
- mkfs.xfs(8) man page
- Red Hat Enterprise Linux 性能调优指南 , 章节 https://access.redhat.com/documentation/zh-cn/red_hat_enterprise_linux/7/html-single/performance_tuning_guide/index#sect-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Configuring_file_systems_for_performance-Tuning_XFS

3.2. 挂载 XFS 文件系统

XFS 文件系统可以在没有额外选项的情况下挂载，例如：

```
# mount /dev/device /mount/point
```

Red Hat Enterprise Linux 7 的默认值为 inode64。



注意

与 **mke2fs** 不同，**mkfs.xfs** 不使用配置文件；它们都是在命令行中指定的。

写屏障

默认情况下，XFS 使用写屏障来确保文件系统的完整性，即使启用了写缓存的设备掉电了。对于没有写缓存的设备，或者有电池支持的写缓存的设备，请使用 **nobarrier** 选项禁用障碍：

```
# mount -o nobarrier /dev/device /mount/point
```

有关写屏障的详情，请参考 第 22 章 *写屏障*。

直接访问技术预览

从 Red Hat Enterprise Linux 7.3 开始，在 ext4 和 XFS 文件系统中作为技术预览提供 **直接访问** (DAX)。应用是一种将持久内存直接映射到其地址空间的方法。要使用 DAX，系统必须有某种可用的持久性内存，通常使用一个或多个非易失性双内存模块(NVDIMM)，必须在 NVDIMM 上创建支持 DAX 的文件系统。另外，该文件系统必须使用 **dax** 挂载选项挂载。然后，在 dax 挂载的文件系统中的一个文件 **mmap** 会导致存储直接映射到应用程序的地址空间中。

3.3. XFS 配额管理

XFS 配额子系统管理磁盘空间（块）和文件(inode)使用情况的限制。XFS 配额控制或报告在用户、组、目录或项目级别使用这些项目的使用情况。此外，请注意，尽管用户、组、目录或项目配额是独立启用的，但组和项目配额是互斥的。

在按目录或按项目管理时，XFS 管理与特定项目关联的目录层次结构的磁盘使用情况。为此，XFS 可以识别项目之间的跨组织的"组"边界。这提供了比管理用户或组配额更广泛的控制级别。

XFS 配额在挂载时通过特定的挂载选项启用。每个挂载选项也可以指定为 **noenforce**; 这允许在不强制实施任何限制的情况下报告使用情况。有效的配额挂载选项有：

- **uquota**/**uqnoenforce**: 用户配额
- **gquota**/**gqnoenforce**: 组配额
- **pquota**/**pqnoenforce**: 项目配额

启用配额后，**xfs_quota** 工具可用于设置限制并报告磁盘使用情况。默认情况下，**xfs_quota** 以交互方式运行，并以 *基本模式*。基本模式子命令只是报告使用情况，适用于所有用户。基本 **xfs_quota** 子命令包括：

quota *username/userID*

显示给定 ***username*** 或数字 ***userID***的使用情况和限制

df

显示块和 inode 的空闲和已使用数。

相反，**xfs_quota** 也具有 *专家模式*。此模式的子命令允许实际限制的配置，并且仅可提供给具有升级特权的用户使用。要以交互方式使用专家模式子命令，请使用以下命令：

```
# xfs_quota -x
```

专家模式子命令包括：

report *path*

报告特定文件系统的配额信息。

limit

修改配额限制。

如需基本或专家模式的子命令的完整列表，请使用子命令 **help**。

所有子命令也可以使用 **-c** 选项直接从命令行运行，而 **-x** 用于专家子命令。

例 3.2. 显示示例配额报告

例如，要显示 **/home** 的配额报告示例（在 **/dev/blockdevice**上），请使用命令 **xfs_quota -x -c 'report -h' /home**。此时会显示类似如下的输出：

```
User quota on /home (/dev/blockdevice)
Blocks
User ID      Used   Soft   Hard Warn/Grace
---------- ---------------------------------
root            0      0      0  00 [------]
testuser   103.4G      0      0  00 [------]
...
```

要为用户 **john** 设置软和硬内节点计数限制为 500 和 700，其主目录为 **/home/john**，请使用以下命令：

```
# xfs_quota -x -c 'limit isoft=500 ihard=700 john' /home/
```

在这种情况下，传递 *mount_point*，这是挂载的 xfs 文件系统。

默认情况下，**limit** 子命令将目标识别为用户。在为组配置限制时，请使用 **-g** 选项（如上例中所示）。同样，将 **-p** 用于项目。

也可以使用 **bsoft** 或 **bhard** 而不是 **isoft** 或 **ihard** 来配置软和硬块限制。

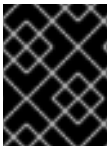
例 3.3. 设置 Soft 和 Hard Block Limit

例如，要将软和硬块限制分别设置为 1000m 和 1200m，来对 **/target/path** 文件系统上的 **accounting** 进行分组，请使用以下命令：

```
# xfs_quota -x -c 'limit -g bsoft=1000m bhard=1200m accounting' /target/path
```

注意

命令 **bsoft** 和 **bhard** 计数为字节。



重要

虽然 **man xfs_quota** 中描述了实时块(**rtbhard**/**rtbsoft**)，但在设置配额时，不会启用实时子卷。因此，**rtbhard** 和 **rtbsoft** 选项不适用。

设置项目限制

使用 XFS 文件系统，您可以在称为受管树的文件系统中单独目录层次结构上设置配额。每个受管树都通过 *项目 ID* 和可选项目名称 唯一标识。

1. *将项目控制的目录添加到 **/etc/projects**。例如，以下命令将唯一 ID 为 11 的 **/var/log** 路径添加到 **/etc/projects**。您的项目 ID 可以是任何映射到项目的数字值。*

   ```
   # echo 11:/var/log >> /etc/projects
   ```

2. *将项目名称添加到 **/etc/projid**，将项目 ID 映射到项目名称。例如，以下命令将名为 logfiles 的项目与上一步中定义的项目 ID 11 关联：*

   ```
   # echo logfiles:11 >> /etc/projid
   ```

3. *初始化项目目录。例如，以下命令初始化项目目录 **/var** ：*

   ```
   # xfs_quota -x -c 'project -s logfiles' /var
   ```

4. *为使用初始化目录的项目配置配额：*

   ```
   # xfs_quota -x -c 'limit -p bhard=lg logfiles' /var
   ```

通用配额配置工具（例如**quota**、picquota 和 **edquota** ）也可用于操作 XFS 配额。**但是这些工具不能用于 XFS 项目配额。**

重要

红帽建议在所有其他可用工具中使用 **xfs_quota**。

有关设置 XFS 配额的更多信息，请参阅 **man xfs_quota**、**man projid (5)** 和 **man projects (5)**。

3.4. 增加 XFS 文件系统的大小

使用 **xfs_growfs** 命令在挂载时可能会增大 XFS 文件系统：

```
# xfs_growfs /mount/point -D size
```

-D size 选项将文件系统增大到指定的大小（以文件系统块表示）。如果没有 **-D size** 选项，**xfs_growfs** 会将文件系统增大到该设备支持的最大大小。

在使用 **-D** 大小 增大 XFS 文件系统之前，请确保底层块设备具有适当的大小，以便以后存放文件系统。为受影响的块设备使用合适的调整大小的方法。

注意

尽管挂载时可以增大 XFS 文件系统，但不能缩小它们的大小。

有关增大文件系统的更多信息，请参阅 **man xfs_growfs**。

3.5. 修复 XFS 文件系统

要修复 XFS 文件系统，请使用 **xfs_repair** ：

```
# xfs_repair /dev/device
```

xfs_repair 工具高度可扩展，旨在高效地修复具有许多 inode 的超大文件系统。与其他 Linux 文件系统不同，**xfs_repair** 不会在引导时运行，即使 XFS 文件系统没有被完全卸载。如果未完全卸载，**xfs_repair** 会在挂载时简单地重新执行日志，确保文件系统一致。

警告

xfs_repair 工具无法修复带有脏日志的 XFS 文件系统。要清除日志，请挂载并卸载 XFS 文件系统。如果日志损坏且无法重新执行，请使用 **-L** 选项("强制日志归零")清除日志，即 **xfs_repair -L /dev/device**。请注意，这可能导致进一步损坏或数据丢失。

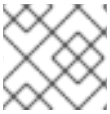
有关修复 XFS 文件系统的更多信息，请参阅 **man xfs_repair**。

3.6. 暂停 XFS 文件系统

要挂起或恢复对文件系统的写活动，请使用以下命令：

```
# xfs_freeze mount-point
```

暂停写活动允许使用基于硬件的设备快照捕获处于一致状态的文件系统。



注意

xfs_freeze 工具由 **xfsprogs** 软件包提供，该软件包仅适用于 x86_64。

要暂停（也就是冻结）XFS 文件系统，请使用：

```
# xfs_freeze -f /mount/point
```

要解冻 XFS 文件系统，请使用：

```
# xfs_freeze -u /mount/point
```

在进行 LVM 快照时，不需要先使用 **xfs_freeze** 来暂停文件系统。相反，LVM 管理工具将在拍摄快照之前自动暂停 XFS 文件系统。

有关冻结和释放 XFS 文件系统的更多信息，请参阅 **man xfs_freeze**。

3.7. 备份和恢复 XFS 文件系统

XFS 文件系统备份和恢复涉及以下工具：

- 用于创建备份的 xfsdump
- xfsrestore 用于从备份中恢复

3.7.1. XFS 备份和恢复的功能

Backup

您可以使用 **xfsdump** 工具来：

- 对常规文件镜像执行备份。

  只能将一个备份写入常规文件。

- 在磁带驱动器中执行备份。

  xfsdump 工具还允许您将多个备份写入同一磁带。备份可跨越多个标题。

  要将多个文件系统备份到单个磁带设备，只需将备份写入已包含 XFS 备份的磁带。这会将新备份附加到上一个备份。默认情况下，**xfsdump** 不会覆盖现有的备份。

- 创建增量备份。

  xfsdump 工具使用转储级别来决定其他备份相对的基本备份。从 0 到 9 的数字指的是增加的转储级别。增量备份只备份自上一次较低级别转储以来发生变化的文件：

- 要执行全备份，请在文件系统中执行 0 级转储。
- 1 级转储是全备份后的第一个增量备份。下一个增量备份为 2 级，它仅备份自上 1 级转储后更改的文件，以此类推，最多为 9 级。

- 使用大小、子树或 inode 标志从备份中排除文件，以过滤它们。

恢复

xfsrestore 工具从 xfsdump 生成的备份中恢复文件系统。xfsrestore 工具有两种模式：

- 简单 模式允许用户从 0 级转储恢复整个文件系统。这是默认的模式。
- 累积 模式启用从增量备份恢复文件系统：即，1 级到 9 级。

唯一的 会话 ID 或会话标签 标识每个备份。从包含多个备份的磁带恢复备份需要相应的会话 ID 或标签。

要从备份中提取、添加或删除特定文件，请输入 **xfsrestore** 交互模式。交互模式提供了一组命令来操作备份文件。

3.7.2. 备份 XFS 文件系统

这个步骤描述了如何将 XFS 文件系统的内容备份到文件或者磁带中。

过程 3.1. 备份 XFS 文件系统

- 使用以下命令备份 XFS 文件系统：

```
# xfsdump -l level [-L label] -f backup-destination path-to-xfs-filesystem
```

  - 使用备份的转储级别替换 level。使用 **0** 执行完整备份，或 **1** 到 **9** 执行后续增量备份。
  - 使用您要存储备份的路径替换 backup-destination。目的地可以是常规文件、磁带驱动器或远程磁带设备。例如：用于文件的 **/backup-files/Data.xfsdump**，对于磁带驱动器，**/dev/st0**。
  - 使用您要备份的 XFS 文件系统的挂载点替换 path-to-xfs-filesystem。例如：**/mnt/data/**。文件系统必须挂载。
  - 当备份多个文件系统并将其保存到单个磁带设备中时，使用 **-L label** 选项为每个备份添加一个会话标签，以便在恢复时更容易识别它们。使用备份的任何名称替换 label ：例如 **backup_data**。

例 3.4. 备份多个 XFS 文件系统

- 要备份挂载在 **/boot/** 和 **/data/** 目录中的 XFS 文件系统的内容，并将其保存为 **/backup-files/** 目录中的文件：

```
# xfsdump -l 0 -f /backup-files/boot.xfsdump /boot
# xfsdump -l 0 -f /backup-files/data.xfsdump /data
```

- 要备份单个磁带设备中的多个文件系统，请使用 **-L label** 选项为每个备份添加一个会话标签：

```
# xfsdump -l 0 -L "backup_boot" -f /dev/st0 /boot
# xfsdump -l 0 -L "backup_data" -f /dev/st0 /data
```

其它资源

- **有关备份 XFS 文件系统的详情请参考 xfsdump(8) man page。**

3.7.3. 从备份中恢复 XFS 文件系统

这个步骤描述了如何从文件或者磁带备份中恢复 XFS 文件系统的内容。

先决条件

- **您需要 XFS 文件系统的文件或者磁带备份，如 第 3.7.2 节 "备份 XFS 文件系统" 所述。**

过程 3.2. 从备份中恢复 XFS 文件系统

- **恢复备份的命令因您是从全备份或增量备份中恢复，还是从单个磁带设备恢复多个备份而有所不同：**

```
# xfsrestore [-r] [-S session-id] [-L session-label] [-i]
           -f backup-location restoration-path
```

  - **使用备份位置替换 backup-location。这可以是常规文件、磁带驱动器或远程磁带设备。例如：用于文件的 /backup-files/Data.xfsdump，对于磁带驱动器，/dev/st0。**
  - **使用您要恢复文件系统的目录的路径替换 restore-path。例如：/mnt/data/。**
  - **要从增量(1 级到 9 级)备份恢复文件系统，请添加 -r 选项。**
  - **要从包含多个备份的磁带设备恢复备份，请使用 -S 或 -L 选项指定备份。**

    -S 允许您通过其会话 ID 选择备份，而 -L 则允许您按会话标签选择。要获取会话 ID 和会话标签，请使用 xfsrestore -I 命令。

    使用备份的会话 ID 替换 session-id。例如，b74a3586-e52e-4a4a-8775-c3334fa8ea2c。使用备份的会话标签替换 session-label。例如，my_backup_session_label。
  - **要以交互方式使用 xfsrestore，请使用 -i 选项。**

    xfsrestore 完成读取指定设备后，交互式对话框开始。交互式 xfsrestore shell 中的可用命令包括 cd, ls, add, delete, 和 extract; 如需命令的完整列表，请使用 help 命令。

例 3.5. 恢复多个 XFS 文件系统

要恢复 XFS 备份文件，并将其内容保存到 /mnt/ 下的目录中：

```
# xfsrestore -f /backup-files/boot.xfsdump /mnt/boot/
# xfsrestore -f /backup-files/data.xfsdump /mnt/data/
```

要从包含多个备份的磁带设备恢复，请使用会话标签或会话 ID 指定每个备份：

```
# xfsrestore -f /dev/st0 -L "backup_boot" /mnt/boot/
# xfsrestore -f /dev/st0 -S "45e9af35-efd2-4244-87bc-4762e476cbab" /mnt/data/
```

从 Tape 恢复备份时的信息性消息
当从带有来自多个文件系统的备份的磁带恢复备份时，xfsrestore 工具可能会发出信息。当 xfsrestore 按顺序检查磁带上的每个备份时，信息会告知您是否找到了与请求的备份相匹配。例如：

```
xfsrestore: preparing drive
xfsrestore: examining media file 0
xfsrestore: inventory session uuid (8590224e-3c93-469c-a311-fc8f23029b2a) does not match
the media header's session uuid (7eda9f86-f1e9-4dfd-b1d4-c50467912408)
xfsrestore: examining media file 1
xfsrestore: inventory session uuid (8590224e-3c93-469c-a311-fc8f23029b2a) does not match
the media header's session uuid (7eda9f86-f1e9-4dfd-b1d4-c50467912408)
[...]
```

说明性消息会一直显示，直到找到匹配的备份。

其它资源

- **有关恢复 XFS 文件系统的详情请参考 xfsrestore(8) man page。**

3.8. 配置错误行为

当 I/O 操作过程中发生错误时，XFS 驱动程序以以下两种方式之一响应：

- **继续重试，直到出现以下任一操作：**
  - **I/O 操作成功，或者**
  - **超过 I/O 操作重试计数或时间限制。**
- **考虑永久错误并停止系统。**

XFS 目前可识别您可以配置所需行为的以下错误条件：

- **EIO: 在尝试写入设备时出现错误**
- **ENOSPC: 该设备中没有剩余空间**
- **ENODEV: Device cannot found**

所有其他可能的错误条件（没有定义特定的处理程序）共享一个全局配置。

您可以设置 XFS 在最大重试次数和最长时间（以秒为单位）中导致错误持久性的条件。当满足任何一个条件时，XFS 会停止重试。

在卸载文件系统时，还有一个选项可立即取消重试，而不考虑任何其他配置。这允许卸载操作在出现持久错误时也可以成功。

3.8.1. 特定和未定义的条件的配置文件

控制错误行为的配置文件位于 /sys/fs/xfs/device/error/ 目录中。

/sys/fs/xfs/设备/error/metadata/ 目录包含每个特定错误条件的子目录：

- **用于 EIO 错误条件的 /sys/fs/xfs/device/error/metadata / EIO /**
- **/sys/fs/xfs/device/error/metadata/ENODEV/ 用于 ENODEV 错误条件**

- */sys/fs/xfs/device/error/metadata/ENOSPC/ 用于 ENOSPC 错误条件*

然后，每个配置文件包含以下配置文件：

- */sys/fs/xfs/device/error/metadata/condition/max_retries: 控制 XFS 重新尝试操作的最大次数。*
- */sys/fs/xfs/device/error/metadata/condition/retry_timeout_seconds: XFS 将停止重试操作的时间限制（以秒为单位）*

除了上一节中描述的其它错误条件外，所有其他可能的错误条件在这些文件中共享一个通用配置：

- */sys/fs/xfs/device/error/metadata/default/max_retries: 控制重试的最大次数*
- */sys/fs/xfs/device/error/metadata/default/retry_timeout_seconds: 控制重试的时间限制*

3.8.2. 为特定和未定义条件设置文件系统行为

要设置重试的最大数量，请将所需的数量写入 max_retries 文件。

- *对于特定条件：*

```
# echo value > /sys/fs/xfs/device/error/metadata/condition/max_retries
```

- *对于未定义的条件：*

```
# echo value > /sys/fs/xfs/device/error/metadata/default/max_retries
```

value 是 -1 和最大可能值为 int 的值(C 签名的整数类型)之间的数字。64 位 Linux 中是 2147483647。

要设置时间限制，请将所需的秒数写入 retry_timeout_seconds 文件。

- *对于特定条件：*

```
# echo value > /sys/fs/xfs/device/error/metadata/condition/retry_timeout_seconds
```

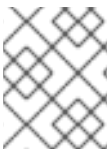
- *对于未定义的条件：*

```
# echo value > /sys/fs/xfs/device/error/metadata/default/retry_timeout_seconds
```

value 是 -1 和 86400 之间的数字，这是一天的秒数。

在 max_retries 和 retry_timeout_seconds 选项中，-1 表示重试 forever 和 0 以立即停止。

device 是设备的名称，如 /dev/ 目录中找到，例如：sda。



注意

每个错误条件的默认行为取决于错误上下文。一些错误（如 ENODEV）被视为致命且不可恢复的，无论重试计数如何，其默认值为 0。

3.8.3. 设置卸载行为

如果设置了 fail_at_unmount 选项，文件系统会在卸载过程中覆盖所有其他错误配置，并在不重试 I/O 操作的情况下立即修改文件系统。这允许卸载操作在出现持久错误时也可以成功。

设置卸载行为：

```
# echo value > /sys/fs/xfs/device/error/fail_at_unmount
```

值可以是 1 或 0 ：

- 1 表示如果找到错误，则立即取消重试。
- 0 表示遵守 max_retries 和 retry_timeout_seconds 选项。

device 是设备的名称，如 /dev/ 目录中找到，例如： sda。

重要

在尝试卸载文件系统前，需要根据需要设置 fail_at_unmount 选项。启动卸载操作后，配置文件和目录可能无法使用。

3.9. 其他 XFS 文件系统工具

Red Hat Enterprise Linux 7 还具有管理 XFS 文件系统的其他工具：

xfs_fsr

用于对已挂载的 XFS 文件系统进行碎片整理。当不使用任何参数调用时，xfs_fsr 会对所有挂载的 XFS 文件系统中的所有常规文件进行碎片整理。此工具还允许用户在指定的时间暂停碎片整理，并在以后从其停止的地方恢复。

另外，xfs_fsr 还允许只对一个文件进行碎片整理，如 xfs_fsr /path/to/file 中所述。红帽建议不要定期取消整个文件系统碎片，因为 XFS 默认避免碎片。系统范围的碎片整理可能会导致空闲空间出现碎片的影响。

xfs_bmap

打印 XFS 文件系统中文件所使用的磁盘块图。此图列出了指定文件所使用的每一个块，以及文件中没有相应块（即漏洞）的区域。

xfs_info

打印 XFS 文件系统信息.

xfs_admin

更改 XFS 文件系统的参数.xfs_admin 工具只能修改卸载的设备或文件系统的参数。

xfs_copy

将整个 XFS 文件系统的内容并行复制到一个或多个目的地。

以下工具在调试和分析 XFS 文件系统时也很有用：

xfs_metadump

将 XFS 文件系统元数据复制到文件中。红帽只支持使用 xfs_metadump 工具复制卸载的文件系统或只读挂载的文件系统；否则，生成的转储可能会损坏或不一致。

xfs_mdrestore

将 XFS metadump 镜像（使用 **xfs_metadump** 生成）恢复到文件系统镜像。

xfs_db

调试 XFS 文件系统。

有关这些工具的详情，请查看其各自的 **man** page。

3.10. 从 EXT4 迁移到 XFS

从 Red Hat Enterprise Linux 7.0 开始，XFS 是默认文件系统，而不是 ext4。本节重点介绍使用或管理 XFS 文件系统时的区别。

Red Hat Enterprise Linux 7 中仍完全支持 ext4 文件系统，并在安装时可以选择。虽然可以从 ext4 迁移到 XFS，但这不是必须的。

3.10.1. Ext3/4 和 XFS 之间的区别

文件系统修复

Ext3/4 在引导时在用户空间中运行 **e2fsck**，以便根据需要恢复日志。相比之下，XFS 在挂载时在内核空间中执行日志恢复。提供了 **fsck.xfs** shell 脚本，但不执行任何有用的操作，因为它只能满足 initscript 要求。

当请求 XFS 文件系统修复或检查时，请使用 **xfs_repair** 命令。使用 **-n** 选项进行只读检查。

xfs_repair 命令不会在带有脏日志的文件系统上运行。要修复此类文件系统 挂载和卸载，必须首先执行重新执行来回放日志。如果日志损坏且无法重新执行，可以在日志中使用 **-L** 选项为零。

有关 XFS 文件系统修复文件系统的更多信息，请参阅 第 12.2.2 节 “XFS”

元数据错误行为

当遇到元数据错误时，ext3/4 文件系统具有可配置的行为，其默认设置只是继续。当 XFS 遇到不可恢复的元数据错误时，它将关闭文件系统并返回 **EFSCORRUPTED** 错误。系统日志将包含遇到的错误的详情，并在需要时推荐运行 **xfs_repair**。

配额

XFS 配额不是一个可重新挂载的选项。必须在初始挂载上指定 **-o quota** 选项，才能使配额生效。

虽然 quota 软件包中的标准工具可以执行基本的配额管理任务（如 setquota 和 repquota），xfs_quota 工具可用于特定于 XFS 的功能，如项目配额管理。

quotacheck 命令对 XFS 文件系统没有影响。第一次在 XFS 上打开配额核算时，会在内部执行自动 **quotacheck**。由于 XFS 配额元数据是第一类，因此日志的元数据对象将始终一致，直到手动关闭配额为止。

文件系统重新定义大小

XFS 文件系统没有工具来缩小文件系统。XFS 文件系统可以通过 **xfs_growfs** 命令在线增长。

内节点（inode）号

对于大于 1 TB 且具有 256 字节索引节点的文件系统，或者大于 2 TB，XFS 索引节点编号可能超过 2^32。这种大型索引节点编号会导致 32 位 stat 调用失败，并显示 EOVERFLOW 返回值。使用默认的

Red Hat Enterprise Linux 7 配置时，上面描述的问题可能会发生：使用四个分配组非条带化。自定义配置（如文件系统扩展或更改 XFS 文件系统参数）可能会导致不同的行为。

应用程序通常正确处理此类更大的索引节点编号。如果需要，使用 **-o inode32** 参数挂载 XFS 文件系统，以强制低于 2^32 的 inode 号。请注意，使用 **inode32** 不会影响已分配了 64 位数的 inode。

重要

除非特定环境需要，否则不要使用 **inode32** 选项。**inode32** 选项可改变分配行为。因此，如果没有可用空间在较低磁盘块中分配 inode，则可能会出现 ENOSPC 错误。

定性预分配

XFS 使用 规范预分配 来分配块过去 EOF，因为文件被写入。这可避免因为 NFS 服务器上的并发流写工作负载造成文件碎片。默认情况下，此预分配会增加文件大小，并将在"du"输出中明显显示。如果没有在五分钟内没有指定指定预分配的文件，则会丢弃预分配。如果内节点在这个时间前从缓存中循环，则当内节点被回收时，将丢弃预分配。

如果因为定性预分配而看到了预规划 ENOSPC 问题，可以使用 **-o allocsize=amount** 挂载选项指定固定的预分配量。

与碎片相关的工具

由于 Heuristics 和行为（如延迟分配和规范预分配）导致 XFS 文件系统来说，碎片很少发生。但是，存在用于衡量文件系统碎片的工具，以及对文件系统进行碎片整理。不鼓励使用它们。

xfs_db frag 命令尝试将所有文件系统分配成单个碎片编号，以百分比表示。命令的输出需要大量的专业知识才能理解其含义。例如，一个碎片因数为 75% 意味着每个文件平均有 4 个扩展。因此，xfs_db 的 frag 的输出不被视为有用，并建议更小心地分析任何碎片问题。

警告

xfs_fsr 命令可用于对单个文件或文件系统上的所有文件进行碎片整理。强烈建议不要这样做，因为它可能会破坏文件的本地性，并可能出现碎片释放的空间。

使用 ext3 和 ext4 的命令与 XFS 的比较

下表将用于 ext3 和 ext4 的常见命令与特定于 XFS 的对应部分进行比较。

表 3.1. ext3 和 ext4 的通用命令与 XFS 的比较

任务	ext3/4	XFS
创建文件系统	**mkfs.ext4** or **mkfs.ext3**	**mkfs.xfs**
文件系统检查	**e2fsck**	**xfs_repair**
重新定义文件系统大小	**resize2fs**	**xfs_growfs**

任务	ext3/4	XFS
保存文件系统的镜像	**e2image**	**xfs_metadump** 和 **xfs_mdrestore**
标签或者调整文件系统	**tune2fs**	**xfs_admin**
备份文件系统	**dump** 和 **restore**	**xfsdump** 和 **xfsrestore**

下表列出了 XFS 文件系统功能的通用工具，但 XFS 版本具有更具体的功能，因此推荐这样做。

表 3.2. ext4 和 XFS 的通用工具

任务	ext4	XFS
Quota	**quota**	**xfs_quota**
文件映射	**filefrag**	**xfs_bmap**

有关许多列出的 XFS 命令的详情请参考 第 3 章 XFS 文件系统。如需更多信息，您还可以查阅列出的 XFS 管理工具的手册页。

第 4 章 EXT3 文件系统

ext3 文件系统基本上是一个 ext2 文件系统的改进版本。这些改进提供以下优点：

可用性

在意外的电源故障或系统崩溃（也称为 未清理的系统关闭）后，机器上每个挂载的 ext2 文件系统都必须通过 e2fsck 程序检查一致性。这是一个耗时的过程，可能会显著延迟系统引导时间，特别是对于包含大量文件的大量卷而言。在这段时间中，卷上的任何数据都不可访问。

可以在实时文件系统上运行 fsck -n。但是，如果遇到写入了部分元数据，它不会进行任何更改，并且可能会产生误导。

如果在堆栈中使用 LVM，另一个选项是生成文件系统的 LVM 快照，并在其上运行 fsck。

最后，您可以选择以只读形式重新挂载文件系统。然后，在重新挂载前，所有待处理的元数据更新（及写入）都会强制到磁盘中。这样可确保文件系统处于一致状态，前提是之前没有损坏。现在，可以运行 fsck -n。

ext3 文件系统提供的日志意味着，在未清理的系统关闭后，不再需要此类文件系统检查。使用 ext3 进行一致性检查的唯一时间是在某些罕见的硬件故障情形中，如硬盘驱动器故障。在未清理的系统关闭后恢复 ext3 文件系统的时间不取决于文件系统的大小或文件的数量，而是取决于用于保持一致性的 日志 的大小。默认日志大小大约需要一秒钟就能恢复，具体取决于硬件的速度。

注意

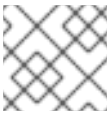
红帽支持的 ext3 中的唯一日志模式是 data=ordered （默认）。

数据完整性

ext3 文件系统防止发生未清理系统关闭时数据完整性的丢失。ext3 文件系统允许您选择数据接受的保护类型和级别。对于文件系统的状态，ext3 卷配置为默认保持高级别的数据一致性。

速度

尽管多次写入一些数据，但大多数情况下， ext3 的吞吐量高于 ext2，因为 ext3 的日志优化了硬盘驱动器磁头的移动。您可以从三种日志记录模式中进行选择来优化速度，但这样做意味着在系统出现故障时要权衡数据的完整性。



注意

红帽支持的 ext3 中的唯一日志模式是 data=ordered （默认）。

轻松迁移

可轻松从 ext2 迁移到 ext3，并获得强大的日志记录文件系统的优势，而无需重新格式化。有关执行此任务的详情，请参考 第 4.2 节 “转换为 ext3 文件系统”。

注意

Red Hat Enterprise Linux 7 提供统一的 extN 驱动程序。它通过禁用 ext2 和 ext3 配置，而是对这些磁盘上的格式使用 ext4.ko。这意味着，无论使用的 ext 文件系统是什么，内核信息总是指 ext4。

4.1. 创建 EXT3 文件系统

安装后，有时需要创建新的 ext3 文件系统。例如，如果系统中添加了新的磁盘驱动器，您可能需要对该驱动器进行分区，并使用 ext3 文件系统。

1. 使用 **mkfs.ext3** 工具使用 ext3 文件系统格式化分区或 LVM 卷：

```
# mkfs.ext3 block_device
```

   - 使用到块设备的路径替换 block_device。例如：**/dev/sdb1**,**/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**, 或 **/dev/my-volgroup/my-lv**。

2. 使用 **e2label** 工具标记文件系统：

```
# e2label block_device volume_label
```

配置 UUID

也可以为文件系统设置特定的 UUID。要在创建文件系统时指定 UUID，请使用 **-U** 选项：

```
# mkfs.ext3 -U UUID device
```

- 使用您要设置的 UUID 替换 UUID：例如 **7cd65de3-e0be-41d9-b66d-96d749c02da7**。
- 使用 ext3 文件系统的路径替换 device，以将 UUID 添加到其中：例如 **/dev/sda8**。

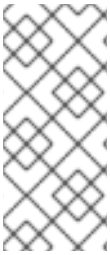
要更改现有文件系统的 UUID，请参阅 第 25.8.3.2 节 “修改持久性命名属性”

其它资源

- mkfs.ext3(8) man page
- e2label(8) man page

4.2. 转换为 EXT3 文件系统

tune2fs 命令将 **ext2** 文件系统转换为 **ext3**。



注意

要将 ext2 转换为 ext3，请始终使用 **e2fsck** 工具，在使用 **tune2fs** 之前和之后检查您的文件系统。在尝试将 ext2 转换为 ext3 之前，请备份所有文件系统，以防出现错误。

另外，红帽建议创建新的 ext3 文件系统并将数据迁移到其中，而不是尽可能从 ext2 转换到 ext3。

要将 **ext2** 文件系统转换为 **ext3**，以 root 身份登录并在终端中输入以下命令：

```
# tune2fs -j block_device
```

block_device 包含要转换的 ext2 文件系统。

发出 **df** 命令来显示挂载的文件系统。

4.3. 恢复回 EXT2 文件系统

要恢复回 ext2 文件系统，请使用以下流程：

为简单起见，本节中的示例命令对块设备使用以下值：

/dev/mapper/VolGroup00-LogVol02

过程 4.1. 从 ext3 恢复回 ext2

1. **以 root 身份登录，并输入以下内容来卸载分区：**

```
# umount /dev/mapper/VolGroup00-LogVol02
```

2. **输入以下命令将文件系统类型改为 ext2：**

```
# tune2fs -O ^has_journal /dev/mapper/VolGroup00-LogVol02
```

3. **输入以下命令检查分区是否有错误：**

```
# e2fsck -y /dev/mapper/VolGroup00-LogVol02
```

4. **然后，通过输入以下内容将该分区再次挂载为 ext2 文件系统：**

```
# mount -t ext2 /dev/mapper/VolGroup00-LogVol02 /mount/point
```

使用分区的挂载点替换 /mount/point。

注意

如果分区的根级别存在 .journal 文件，请将其删除。

要将分区永久更改为 ext2，请记住要更新 /etc/fstab 文件，否则它将在引导后恢复回来。

第 5 章 EXT4 文件系统

ext4 文件系统是 ext3 文件系统的可扩展扩展。对于 Red Hat Enterprise Linux 7，它可以支持最大大小为 16 TB 的单个文件大小，文件系统可以最多支持 50 TB，而 Red Hat Enterprise Linux 6 只支持最多 16 TB 的文件系统。它还支持无限数量的子目录（ext3 文件系统最多支持 32,000 个），但是一旦链接数超过 65,000 个，它将重置回 1，并且不再增加。目前不支持 bigalloc 功能。

注意

与 ext3 一样，必须卸载 ext4 卷才能执行 fsck。如需更多信息，请参阅 第 4 章 ext3 文件系统。

主要功能

ext4 文件系统使用扩展（与 ext2 和 ext3 使用的传统块映射方案相反），这提高了使用大型文件时的性能，并减少大型文件的元数据开销。此外，ext4 还相应地标记未分配的块组和 inode 表部分，这允许在文件系统检查期间跳过它们。这样可加快文件系统检查的速度，随着文件系统大小的增加，这将变得更加有益。

分配功能

ext4 文件系统具有以下分配方案：

- 持久性预分配
- 延迟分配
- 多块分配
- 条带感知分配

由于延迟分配和其他性能优化，ext4 向磁盘写文件的行为与 ext3 不同。在 ext4 中，当程序写入文件系统时，无法保证为磁盘上，除非程序随后发出 fsync （） 调用。

默认情况下，ext3 会自动立即将新创建的文件强制到磁盘，即使没有 fsync （）。此行为隐藏了程序中的错误，这些程序不使用 fsync （） 来确保写入的数据在磁盘上。另一方面，ext4 文件系统通常会等待几秒钟才能将更改写入磁盘，从而允许它合并并重新排序写操作，以获得比 ext3 更好的磁盘性能。

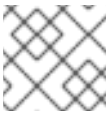
警告

与 ext3 不同，ext4 文件系统不会在事务提交时强制数据写到磁盘。因此，将缓冲的写入刷新到磁盘需要更长的时间。与任何文件系统一样，请使用 fsync （） 等数据完整性调用来确保数据被写入持久性存储。

其他 ext4 功能

ext4 文件系统还支持以下内容：

- 扩展属性 (xattr) - 这允许系统为每个文件关联几个额外的名称和值对。
- 配额日志 - 这避免了崩溃后需要冗长的配额一致性检查。



注意

ext4 中唯一支持的日志模式是 data=ordered （默认）。

- **亚秒时间戳 - 这向亚秒提供时间戳。**

5.1. 创建 EXT4 文件系统

- **要创建 ext4 文件系统，请使用以下命令：**

```
# mkfs.ext4 block_device
```

  - **使用到块设备的路径替换 block_device。例如：/dev/sdb1,/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a, 或 /dev/my-volgroup/my-lv。**
  - **一般说来，默认选项适用于大多数使用场景。**

例 5.1. mkfs.ext4 命令输出

以下是这个命令的输出示例，它显示了生成的文件系统的几何结构和特性：

```
~]# mkfs.ext4 /dev/sdb1
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
245280 inodes, 979456 blocks
48972 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1006632960
30 block groups
32768 blocks per group, 32768 fragments per group
8176 inodes per group
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376, 294912, 819200, 884736

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

重要

可以使用 tune2fs 在 ext3 文件系统上启用特定的 ext4 功能。但是，以这种方式使用 tune2fs 尚未经过充分测试，因此 Red Hat Enterprise Linux 7不支持。因此，红帽不能保证使用 tune2fs 转换或挂载的 ext3 文件系统的一致性性能和可预测的行为。

条带块设备

对于条带块设备（如 RAID5 阵列），可以在创建文件系统时指定条带几何结构。使用正确的条带几何结构可大大提高 ext4 文件系统的性能。

在 LVM 或 MD 卷上创建文件系统时，mkfs.ext4 选择最佳的几何结构。在将几何结构信息导出到操作系统的某些硬件 RAID 中也是如此。

要指定条带几何结构，请使用 mkfs.ext4 的 -E 选项（即扩展的文件系统选项），包含以下子选项：

stride=value
: 指定 RAID 块大小。

stripe-width=value
: 指定 RAID 设备中数据磁盘的数量，或者条带中条带单位的数量。

对于这两个子选项，必须在文件系统块单元中指定 value。例如，要在 4k 块文件系统上创建带有 64k（即 16 x 4096）的文件系统，请使用以下命令：

```
# mkfs.ext4 -E stride=16,stripe-width=64 /dev/block_device
```

配置 UUID

也可以为文件系统设置特定的 UUID。要在创建文件系统时指定 UUID，请使用 -U 选项：

```
# mkfs.ext4 -U UUID device
```

- 使用您要设置的 UUID 替换 UUID：例如 7cd65de3-e0be-41d9-b66d-96d749c02da7。
- 使用 ext4 文件系统的路径替换 device，以将 UUID 添加到其中：例如 /dev/sda8。

要更改现有文件系统的 UUID，请参阅 第 25.8.3.2 节 “修改持久性命名属性”

其它资源

有关创建 ext4 文件系统的详情，请参考：

- mkfs.ext4(8) man page

5.2. 挂载 EXT4 文件系统

无需额外选项就可挂载 ext4 文件系统。例如：

```
# mount /dev/device /mount/point
```

ext4 文件系统也支持几个挂载选项来影响行为。例如，acl 参数启用访问控制列表，而 user_xattr 参数则启用用户扩展属性。要启用这两个选项，请使用它们对应的参数与 -o，如下所示：

```
# mount -o acl,user_xattr /dev/device /mount/point
```

与 ext3 一样，如果文件数据出现错误，则可以使用 data_err=abort 选项来中止日志。

```
# mount -o data_err=abort /dev/device /mount/point
```

tune2fs 工具还允许管理员在文件系统超级块中设置默认挂载选项。有关这方面的更多信息，请参阅 man tune2fs。

写屏障

默认情况下，ext4 使用写屏障来确保文件系统的完整性，即使启用了写缓存的设备断电了也是如此。对于没有写缓存的设备，或使用电池支持的写缓存的设备，使用 **nobarrier** 选项禁用障碍，如下所示：

```
# mount -o nobarrier /dev/device /mount/point
```

有关写屏障的详情，请参考 第 22 章 写屏障。

直接访问技术预览

从 Red Hat Enterprise Linux 7.3 开始，**Direct Access** (DAX)提供，作为 ext4 和 XFS 文件系统上的一个技术预览，一个应用程序可以将持久内存直接映射到其地址空间中。要使用 DAX，系统必须有某种可用的持久内存，通常使用一个或多个非线内存模块(NVDIMM)，且必须在 NVDIMM 上创建支持 DAX 的文件系统。另外，该文件系统必须使用 **dax** 挂载选项挂载。然后，在 dax 挂载的文件系统中的一个文件**mmap** 会导致存储直接映射到应用程序的地址空间中。

5.3. 重新定义 EXT4 文件系统大小

在增大 ext4 文件系统之前，请确保底层块设备的大小合适，以便能够存放之后的文件系统。为受影响的块设备使用合适的调整大小的方法。

使用 **resize2fs** 命令挂载时可能会增大 ext4 文件系统：

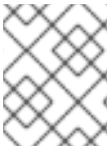
```
# resize2fs /mount/device size
```

resize2fs 命令还可缩小卸载的 ext4 文件系统的大小：

```
# resize2fs /dev/device size
```

在重新定义 ext4 文件系统大小时，size **2fs** 工具以文件系统块大小为单位读取大小，除非使用了表示特定单元的后缀。以下后缀表示特定的单位：

- ***s*** - 512 字节扇区
- ***K*** – kilobytes
- ***M*** - MB
- ***G*** – gigabytes



注意

扩展时 size 参数是可选的（通常是多余的）。**resize2fs** 会自动扩展以填充容器的所有可用空间，通常是逻辑卷或分区。

有关调整 ext4 文件系统大小的更多信息，请参阅 **man resize2fs**。

5.4. 备份 EXT2、EXT3 或 EXT4 文件系统

这个步骤描述了如何将 ext4、ext3 或 ext2 文件系统的内容备份到文件中。

先决条件

- 如果系统运行了很长时间，请在备份前在分区上运行 **e2fsck** 工具：

```
# e2fsck /dev/device
```

过程 5.1. 备份 ext2、ext3 或 ext4 文件系统

1. 备份配置信息，包括 /etc/fstab 文件的内容和 fdisk -l 命令的输出。这对恢复分区非常有用。

   要捕获这些信息，请运行 sosreport 或 sysreport 工具。有关 sosreport 的更多信息，请参阅 sosreport 是什么以及如何在 Red Hat Enterprise Linux 4.6 及之后的版本中创建？Kdowledgebase 文章。

2. 根据分区的角色：

   - 如果您要备份的分区是一个操作系统分区，请将您的系统引导至救援模式。请参阅 系统管理员 指南中的引导到救援模式 部分。
   - 备份常规的数据分区时，将其卸载。

     虽然可以在挂载数据时备份数据分区，但备份挂载数据分区的结果可能会无法预测。

     如果您需要使用 dump 实用程序备份挂载的文件系统，当文件系统没有负载过重时，这样做。备份时，文件系统上会出现更多的活动，备份损坏的风险越高。

3. 使用 dump 程序备份分区的内容：

   ```
   # dump -0uf backup-file /dev/device
   ```

   使用您要存储备份的文件的路径替换 backup-file。使用您要备份的 ext4 分区的名称替换 device。确保您将备份保存到挂载在与您要备份分区不同的分区的目录中。

   例 5.2. 备份多个 ext4 分区

   要将 /dev/ sda1、/dev /sda2 和 /dev/sda3 分区的内容备份到存储在 /backup-files/ 目录中的备份文件，请使用以下命令：

   ```
   # dump -0uf /backup-files/sda1.dump /dev/sda1
   # dump -0uf /backup-files/sda2.dump /dev/sda2
   # dump -0uf /backup-files/sda3.dump /dev/sda3
   ```

   要执行远程备份，请使用 ssh 实用程序或配置免密码 ssh 登录。有关 ssh 和无密码登录的更多信息，请参阅 系统管理员指南中的使用 ssh 实用程序 和使用基于密钥的身份验证部分。https://access.redhat.com/documentation/zh-cn/red_hat_enterprise_linux/7/html-single/system_administrators_guide/index#s2-ssh-configuration-keypairs

   例如，在使用 ssh 时：

   例 5.3. 使用 ssh执行远程备份

   ```
   # dump -0u -f - /dev/device | ssh root@remoteserver.example.com dd of=backup-file
   ```

   请注意，如果使用标准重定向，则必须单独传递 -f 选项。

其它资源

- *如需更多信息，请参阅 dump(8) man page。*

5.5. 恢复 EXT2、EXT3 或 EXT4 文件系统

这个步骤描述了如何从文件备份中恢复 ext4、ext3 或 ext2 文件系统。

先决条件

- *您需要备份分区及其元数据，如 第 5.4 节 "备份 ext2、ext3 或 ext4 文件系统" 所述。*

过程 5.2. 恢复 ext2、ext3 或 ext4 文件系统

1. *如果您要恢复操作系统分区，请将您的系统引导至救援模式。请参阅 系统管理员 指南中的引导到救援模式 部分。*

   普通数据分区不需要这一步。

2. *使用 fdisk 或 parted utilites 重建您要恢复的分区。*

   如果分区不再存在，重新创建它们。新分区必须足够大以包含恢复的数据。正确获取开始和结束号非常重要；它们是在备份时从 fdisk 实用程序获取的分区的开始和结束扇区号。

   有关修改分区的详情，请参考 第 13 章 分区

3. *使用 mkfs 工具格式化目标分区：*

   ```
   # mkfs.ext4 /dev/device
   ```

   重要

   不要 格式化存储您的备份文件的分区。

4. *如果您创建了新分区，请重新标记所有分区，以便它们与 /etc/fstab 文件中的条目匹配：*

   ```
   # e2label /dev/device label
   ```

5. *创建临时挂载点并在其上挂载分区：*

   ```
   # mkdir /mnt/device
   # mount -t ext4 /dev/device /mnt/device
   ```

6. *从挂载的分区上的备份中恢复数据：*

   ```
   # cd /mnt/device
   # restore -rf device-backup-file
   ```

   如果要在远程机器上恢复或从保存在远程主机上的备份文件恢复，您可以使用 ssh 工具。有关 ssh 的更多信息，请参阅 系统管理员指南中的使用 ssh 实用程序 章节。

   请注意，您需要为以下命令配置免密码登录。有关设置免密码 ssh 登录的更多信息，请参阅 系统管理员 指南中的使用基于密钥的身份验证部分。

- **从存储在同一台机器上的备份文件恢复远程机器上的分区：**

```
# ssh remote-address "cd /mnt/device && cat backup-file | /usr/sbin/restore -r -f -"
```

- **从存储在不同远程机器上的备份文件恢复远程机器上的分区：**

```
# ssh remote-machine-1 "cd /mnt/device && RSH=/usr/bin/ssh /usr/sbin/restore -rf
remote-machine-2:backup-file"
```

7. **reboot:**

```
# systemctl reboot
```

例 5.4. 恢复多个 ext4 分区

要从 例 5.2 "备份多个 ext4 分区" 恢复 /dev/sda1、 /dev/sda2 和 /dev/sda3 分区：

1. **使用 fdisk 命令重建您要恢复的分区。**
2. **格式化目标分区：**

```
# mkfs.ext4 /dev/sda1
# mkfs.ext4 /dev/sda2
# mkfs.ext4 /dev/sda3
```

3. **重新标记所有分区，以便它们与 /etc/fstab 文件匹配：**

```
# e2label /dev/sda1 Boot1
# e2label /dev/sda2 Root
# e2label /dev/sda3 Data
```

4. **准备工作目录。**

   挂载新分区：

```
# mkdir /mnt/sda1
# mount -t ext4 /dev/sda1 /mnt/sda1
# mkdir /mnt/sda2
# mount -t ext4 /dev/sda2 /mnt/sda2
# mkdir /mnt/sda3
# mount -t ext4 /dev/sda3 /mnt/sda3
```

   挂载包含备份文件的分区：

```
# mkdir /backup-files
# mount -t ext4 /dev/sda6 /backup-files
```

5. **将数据从备份恢复到挂载的分区：**

```
# cd /mnt/sda1
# restore -rf /backup-files/sda1.dump
# cd /mnt/sda2
```

```
# restore -rf /backup-files/sda2.dump
# cd /mnt/sda3
# restore -rf /backup-files/sda3.dump
```

6. reboot:

```
# systemctl reboot
```

其它资源

- 如需更多信息，请参阅 restore(8) man page。

5.6. 其他 EXT4 文件系统实用程序

Red Hat Enterprise Linux 7 还具有管理 ext4 文件系统的其他工具：

e2fsck

用于修复 ext4 文件系统.由于 ext4 磁盘结构的更新，此工具对 ext4 文件系统的检查和修复比对 ext3 更高效。

e2label

更改 ext4 文件系统上的标签。此工具也适用于 ext2 和 ext3 文件系统。

quota

控制并报告 ext4 文件系统上用户和组的磁盘空间（块）和文件（inode）的使用情况。有关使用 配额的详情，请参考 **man quota** 和 第 17.1 节 "配置磁盘配额"。

fsfreeze

要暂停对文件系统的访问，请使用命令 **# fsfreeze -f mount-point** 来冻结它，而 **# fsfreeze -u mount-point** to unfreeze it。这会停止对文件系统的访问，并在磁盘上创建稳定的镜像。

注意

对于设备映射器驱动器，不需要使用 **fsfreeze**。

如需更多信息，请参阅 **fsfreeze (8)** 手册页。

如 第 5.2 节 "挂载 ext4 文件系统" 中所示，**tuned2fs** 工具也可以调整 ext2、ext3 和 ext4 文件系统的可配置文件系统参数。另外，以下工具在调试和分析 ext4 文件系统时也很有用：

debugfs

调试 ext2、ext3 或 ext4 文件系统。

e2image

将重要的 ext2、ext3 或 ext4 文件系统元数据保存到文件中。

有关这些工具的详情，请参考其各自的 **man** page。

第 6 章 BTRFS（技术预览）

注意

Btrfs 在 Red Hat Enterprise Linux 7 中作为技术预览功能提供，但自 Red Hat Enterprise Linux 7.4 发行版本起已被弃用。它将在以后的 Red Hat Enterprise Linux 主发行版本中删除。

如需更多信息，请参阅 Red Hat Enterprise Linux 7.4 发行注记中的 已弃用的功能。

btrfs 是下一代 Linux 文件系统，它提供了高级管理功能，具有可靠性和可扩展性功能。提供快照、压缩和集成设备管理方面是唯一的。

6.1. 创建 BTRFS 文件系统

要制作基本的 btrfs 文件系统，请使用以下命令：

```
# mkfs.btrfs /dev/device
```

有关创建带有添加设备的 btrfs 文件系统以及为元数据和数据指定多设备配置集的详情，请参考 第 6.4 节 “多个设备的集成卷管理”。

6.2. 挂载 BTRFS 文件系统

要挂载 btrfs 文件系统中的任何设备，请使用以下命令：

```
# mount /dev/device /mount-point
```

其他有用的挂载选项包括：

device=/dev/name

在 mount 命令中附加这个选项会告知 btrfs 扫描指定设备的 btrfs 卷。这用于确保挂载成功，因为尝试挂载不是 btrfs 的设备将导致挂载失败。

注意

这并不意味着所有设备都会添加到文件系统中，它只扫描它们。

max_inline=number

使用此选项设置可用于元数据 B-tree leaf 中的内联数据的最大空间量（以字节为单位）。默认值为 8192 字节。对于 4k 页，它会被限制为 3900 字节，因为额外的标头需要容纳到 leaf。

alloc_start=number

使用这个选项设置磁盘分配开始的位置。

thread_pool=number

使用这个选项分配分配的 worker 线程数量。

discard

使用这个选项在空闲块中启用 discard/TRIM。

noacl

使用这个选项禁用 ACL 的使用。

space_cache

使用这个选项将可用空间数据存储在磁盘上，以便更快地缓存块组。这是一个持久的更改，可以安全地引导到旧内核。

nospace_cache

使用这个选项禁用上述 space_cache。

clear_cache

使用这个选项清除挂载期间的所有可用空间缓存。这是一个安全的选项，但将触发要重新构建的空间缓存。因此，让文件系统保持挂载，以便重建过程完成。这个挂载选项旨在被使用一次，且仅在问题出现可用空间后才被使用。

enospc_debug

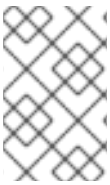
这个选项用于调试"无空格"的问题。

recovery

使用这个选项在挂载时启用自动恢复。

6.3. 调整 BTRFS 文件系统的大小

无法调整 btrfs 文件系统的大小，但可以调整其使用的每个设备的大小。如果只使用一个设备，则它的工作方式与调整文件系统的大小相同。如果使用多个设备，则必须手动调整大小以达到所需结果。



注意

单元大小不具体，它接受 G 或 g 用于 GiB。

该命令不接受 t 代表 TB，或 p 代表 PB。它仅接受 k、m 和 g。

放大 btrfs 文件系统

要在单个设备中增大文件系统，请使用以下命令：

```
# btrfs filesystem resize amount /mount-point
```

例如：

```
# btrfs filesystem resize +200M /btrfssingle
Resize '/btrfssingle' of '+200M'
```

要放大多设备文件系统，必须指定要放大的设备。首先，在指定挂载点显示具有 btrfs 文件系统的所有设备：

```
# btrfs filesystem show /mount-point
```

例如：

```
# btrfs filesystem show /btrfstest
Label: none  uuid: 755b41b7-7a20-4a24-abb3-45fdbed1ab39
 Total devices 4 FS bytes used 192.00KiB
 devid    1 size 1.00GiB used 224.75MiB path /dev/vdc
 devid    2 size 524.00MiB used 204.75MiB path /dev/vdd
 devid    3 size 1.00GiB used 8.00MiB path /dev/vde
 devid    4 size 1.00GiB used 8.00MiB path /dev/vdf

Btrfs v3.16.2
```

然后，在识别要放大设备的 devid 后，使用以下命令：

```
# btrfs filesystem resize devid:amount /mount-point
```

例如：

```
# btrfs filesystem resize 2:+200M /btrfstest
Resize '/btrfstest/' of '2:+200M'
```

注意

数量 也可以是 max 而不是指定的数量。这将使用该设备上的所有剩余可用空间。

缩小 btrfs 文件系统

要缩小单个设备上的文件系统，请使用以下命令：

```
# btrfs filesystem resize amount /mount-point
```

例如：

```
# btrfs filesystem resize -200M /btrfssingle
Resize '/btrfssingle' of '-200M'
```

要缩小多设备文件系统，必须指定 shrunk 设备。首先，在指定挂载点显示具有 btrfs 文件系统的所有设备：

```
# btrfs filesystem show /mount-point
```

例如：

```
# btrfs filesystem show /btrfstest
Label: none  uuid: 755b41b7-7a20-4a24-abb3-45fdbed1ab39
 Total devices 4 FS bytes used 192.00KiB
 devid    1 size 1.00GiB used 224.75MiB path /dev/vdc
 devid    2 size 524.00MiB used 204.75MiB path /dev/vdd
 devid    3 size 1.00GiB used 8.00MiB path /dev/vde
 devid    4 size 1.00GiB used 8.00MiB path /dev/vdf

Btrfs v3.16.2
```

然后，在识别要缩小的设备的 devid 后，使用以下命令：

```
# btrfs filesystem resize devid:amount /mount-point
```

例如：

```
# btrfs filesystem resize 2:-200M /btrfstest
Resize '/btrfstest' of '2:-200M'
```

设置文件系统大小

要将文件系统设置为单一设备上的特定大小，请使用以下命令：

```
# btrfs filesystem resize amount /mount-point
```

例如：

```
# btrfs filesystem resize 700M /btrfssingle
Resize '/btrfssingle' of '700M'
```

要设置多设备文件系统的文件系统大小，必须指定更改的设备。首先，在指定挂载点显示具有 btrfs 文件系统的所有设备：

```
# btrfs filesystem show /mount-point
```

例如：

```
# btrfs filesystem show /btrfstest
Label: none  uuid: 755b41b7-7a20-4a24-abb3-45fdbed1ab39
 Total devices 4 FS bytes used 192.00KiB
 devid    1 size 1.00GiB used 224.75MiB path /dev/vdc
 devid    2 size 724.00MiB used 204.75MiB path /dev/vdd
 devid    3 size 1.00GiB used 8.00MiB path /dev/vde
 devid    4 size 1.00GiB used 8.00MiB path /dev/vdf

Btrfs v3.16.2
```

然后，在识别要更改的设备的 devid 后，使用以下命令：

```
# btrfs filesystem resize devid:amount /mount-point
```

例如：

```
# btrfs filesystem resize 2:300M /btrfstest
Resize '/btrfstest' of '2:300M'
```

6.4. 多个设备的集成卷管理

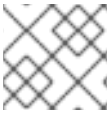
可以在很多设备上创建 btrfs 文件系统，并在文件系统创建后添加更多设备。默认情况下，元数据将跨两个设备进行镜像，数据将在所有存在的设备之间进行条带化，但如果只有一个设备，则元数据将在该设备上重复。

6.4.1. 创建使用多个设备的文件系统

mkfs.btrfs 命令（如 第 6.1 节 "创建 btrfs 文件系统" 所述）接受 -d 用于数据的选项，以及用于元数据的 -m。有效规格包括：

- **raid0**
- **raid1**
- **raid10**
- **dup**
- **单个**

m single 选项指示没有重复元数据。使用硬件 raid 时可能需要这样做。



注意

RAID 10 需要至少四个设备才能正确运行。

> **例 6.1. 创建 RAID 10 btrfs 文件系统**
>
> **在四个设备（元数据镜像、数据条带化）中创建文件系统。**
>
> ```
> # mkfs.btrfs /dev/device1 /dev/device2 /dev/device3 /dev/device4
> ```
>
> **在没有镜像的情况下条带化元数据。**
>
> ```
> # mkfs.btrfs -m raid0 /dev/device1 /dev/device2
> ```
>
> raid10 用于数据和元数据。
>
> ```
> # mkfs.btrfs -m raid10 -d raid10 /dev/device1 /dev/device2 /dev/device3 /dev/device4
> ```
>
> **不要在单一驱动器中重复元数据。**
>
> ```
> # mkfs.btrfs -m single /dev/device
> ```

当驱动器大小不同时，使用单个 选项来使用每个驱动器的完整容量。

```
# mkfs.btrfs -d single /dev/device1 /dev/device2 /dev/device3
```

要在已创建的多设备文件系统中添加新设备，请使用以下命令：

```
# btrfs device add /dev/device1 /mount-point
```

重新引导或重新加载 btrfs 模块后，使用 btrfs 设备扫描 命令来发现所有多设备文件系统。请参阅 第 6.4.2 节 "扫描 btrfs 设备" 了解更多信息。

6.4.2. 扫描 btrfs 设备

使用 btrfs 设备扫描来扫描/dev 下的所有块设备，并探测 btrfs 卷。如果在文件系统中有多个设备运行时，则必须在载入 btrfs 模块后执行此操作。

要扫描所有设备，请使用以下命令：

```
# btrfs device scan
```

要扫描单个设备，请使用以下命令：

```
# btrfs device scan /dev/device
```

6.4.3. 在 btrfs 文件系统中添加新设备

使用 btrfs filesystem show 命令列出所有 btrfs 文件系统及其包含的设备。

btrfs device add 命令用于向挂载的文件系统添加新设备。

btrfs filesystem balance 命令在所有现有设备中平衡（条带）分配的扩展。

用于添加新设备的所有这些命令示例如下：

例 6.2. 在 btrfs 文件系统中添加新设备

首先，创建并挂载 btrfs 文件系统。有关如何创建 btrfs 文件系统的详情，请参考 第 6.1 节 “创建 btrfs 文件系统”，以及 第 6.2 节 “挂载 btrfs 文件系统” 以了解有关如何挂载 btrfs 文件系统的更多信息。

```
# mkfs.btrfs /dev/device1
# mount /dev/device1
```

接下来，在挂载的 btrfs 文件系统中添加第二个设备。

```
# btrfs device add /dev/device2 /mount-point
```

这些设备的元数据和数据仍然仅存储在 /dev/device1 中。现在，它必须平衡才能分散到所有设备上。

```
# btrfs filesystem balance /mount-point
```

平衡文件系统将花费一些时间，因为它会读取所有文件系统的数据和元数据，并在新设备中重写。

6.4.4. 转换 btrfs 文件系统

要将非raid 文件系统转换为 raid，请添加设备并运行更改块分配配置文件的均衡过滤器。

例 6.3. 转换 btrfs 文件系统

要将现有的单个设备系统（本例中为 /dev/sdb 1）转换为两个设备，raid1 系统以便防止单个磁盘失败，请使用以下命令：

```
# mount /dev/sdb1 /mnt
# btrfs device add /dev/sdc1 /mnt
# btrfs balance start -dconvert=raid1 -mconvert=raid1 /mnt
```

重要

如果元数据没有从单设备默认转换，它会保留为 DUP。这不能保证块的副本位于单独的设备上。如果没有转换数据，则它根本没有任何冗余副本。

6.4.5. 删除 btrfs 设备

使用 btrfs device delete 命令删除在线设备。它将使用的任何扩展重新分发到文件系统中的其他设备，以便安全地删除。

例 6.4. 删除 btrfs 文件系统上的设备

首先创建并挂载一些 btrfs 文件系统。

```
# mkfs.btrfs /dev/sdb /dev/sdc /dev/sdd /dev/sde
# mount /dev/sdb /mnt
```

向文件系统添加一些数据。

最后，删除所需的设备。

```
# btrfs device delete /dev/sdc /mnt
```

6.4.6. 在 btrfs 文件系统中替换失败的设备

第 6.4.5 节 “删除 btrfs 设备” 可用于删除提供超级块的失败设备仍然可以读取。但是，如果设备丢失或者超级块损坏，则文件系统需要以降级模式挂载：

```
# mkfs.btrfs -m raid1 /dev/sdb /dev/sdc /dev/sdd /dev/sde


 ssd is destroyed or removed, use -o degraded to force the mount
 to ignore missing devices


# mount -o degraded /dev/sdb /mnt


 'missing' is a special device name


# btrfs device delete missing /mnt
```

命令 btrfs 设备删除缺少的设备会删除文件系统元数据所描述的第一个设备，但在挂载文件系统时不存在。

重要

无法低于特定 raid 布局所需的最小设备数量，即使缺少的设备。可能需要添加新设备才能删除失败的设备。

例如，对于带有两个设备的 raid1 布局，如果设备失败，则需要它：

1. 以 degraded 模式挂载，
2. 添加新设备，
3. 和 删除缺少的设备。

6.4.7. 在 /etc/fstab 中注册 btrfs 文件系统

如果您没有 initrd 或它没有执行 btrfs 设备扫描，则可以通过将文件系统中所有设备明确传递给 mount 命令来挂载多卷 btrfs 文件系统。

> 例 6.5. /etc/fstab 条目示例
>
> 一个合适的 /etc/fstab 条目示例为：
>
> ```
> /dev/sdb  /mnt  btrfs  device=/dev/sdb,device=/dev/sdc,device=/dev/sdd,device=/dev/sde  0
> ```
>
> 请注意，使用通用唯一标识符(UUID)也可以正常工作，且比使用设备路径更稳定。

6.5. SSD 优化

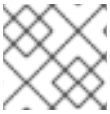
使用 btrfs 文件系统可以优化 SSD。可以通过两种方式完成此操作。

第一种方法是 mkfs.btrfs，在单个设备上为 /sys/block/设备/queue/rotational 关闭元数据重复。这等同于在命令行中指定 -m single。它可以通过提供 -m dup 选项来覆盖和重复元数据。由于 SSD 固件可能会丢失两个副本，因此不需要重复。这种浪费空间是性能成本。

第二种方法是通过一组 SSD 挂载选项： ssd、nossd 和 ssd_spread。

ssd 选项执行几个操作：

- 它允许更大的元数据集群分配。
- 它尽可能顺序分配数据。
- 它禁用 btree leaf rewting 来匹配密钥和块顺序。
- 它在不批量多个进程的情况下提交日志片段。



注意

ssd 挂载选项只启用 ssd 选项。使用nossd 选项禁用它。

在经常重复使用块号时，一些 SSD 会最好地执行，而其他 SSD 则最好地分配未使用的空间的大块。默认情况下，mount -o ssd 将找到块分组，其中有多个可用块可能已混合在其中。命令 mount -o ssd_spread 确保没有混合分配的块。这提高了较低 SSD 的性能。

注意

ssd_spread 选项同时启用 ssd 和 ssd_spread 选项。使用 nossd 禁用这两个选项。

如果提供了任何 ssd 选项且任何设备都不可能，则会自动设置 ssd_spread 选项。

这些选项都需要使用特定的构建进行测试，以查看它们的使用是否提高或降低性能，因为 SSD 固件和应用程序负载的组合都不同。

6.6. BTRFS 参考

man page **btrfs (8)** 涵盖了所有重要的管理命令。特别是，这包括：

- 用于管理快照的所有子卷命令。
- 用于管理 设备 的设备命令。
- 清理、平衡、平衡 和碎片整理 命令。

man page **mkfs.btrfs (8)** 包含有关创建 btrfs 文件系统的信息，包括有关它的所有选项。

man page **btrfsck (8)** 以了解有关 btrfs 系统上 fsck 的信息。

第 7 章 全局文件系统 2

Red Hat Global File System 2 (GFS2)是一个原生文件系统，直接与 Linux 内核文件系统接口(VFS 层)连接。当作为集群文件系统实现时，GFS2 会使用分布式元数据和多个日志。

GFS2 基于 64 位构架，理论上可以容纳 8 EB 的文件系统。但是，当前支持的 GFS2 文件系统的最大值为 100 TB。如果系统需要大于 100 TB 的 GFS2 文件系统，请联络您的红帽服务代表。

在决定文件系统大小时，请考虑其恢复需求。在非常大的文件系统上运行 **fsck** 命令可能需要很长时间，并且消耗大量内存。此外，当磁盘或磁盘子系统发生故障时，恢复时间会受到备份介质速度的限制。

在 Red Hat Cluster Suite 中配置时，可以使用 Red Hat Cluster Suite 配置和管理工具配置和管理 Red Hat GFS2 节点。然后，Red Hat GFS2 在 GFS2 节点间提供数据共享，并在 GFS2 节点上提供文件系统命名空间的单一一致视图。这允许不同节点上的进程共享 GFS2 文件，这与同一节点上的进程可以共享本地文件系统上的文件一样，没有明显差异。有关 Red Hat Cluster Suite 的详情，请参考红帽 的集群管理指南。

GFS2 必须构建在逻辑卷(使用 LVM 创建)上，该逻辑卷是线性卷或镜像卷。在红帽群集套件中使用 LVM 创建的逻辑卷通过 CLVM (LVM 的群集范围实现)进行管理，由 CLVM 守护进程 **clvmd** 启用，并在 Red Hat Cluster Suite 集群中运行。守护进程允许使用 LVM2 管理集群中的逻辑卷，允许集群中的所有节点共享逻辑卷。有关逻辑卷管理器的详情，请参考红帽 逻辑卷管理器管理指南。

gfs2.ko 内核模块实现 GFS2 文件系统，并加载在 GFS2 集群节点上。

有关在集群和非集群存储中创建和配置 GFS2 文件系统的综合信息，请参阅红帽的 全局文件系统 2 指南。

第 8 章 网络文件系统 (NFS)

网络文件系统(NFS)允许远程主机通过网络挂载文件系统，并像它们是本地挂载的文件系统一样与它们进行交互。这使系统管理员能够将资源整合到网络上的集中式服务器上。

本章重点介绍了基本的 NFS 概念和补充信息。

8.1. NFS 简介

目前，Red Hat Enterprise Linux 中包括了两个 NFS 主版本：

- NFS 版本 3 (NFSv3)支持安全异步写入，并在处理错误时比之前的 NFSv2 更强大。它还支持 64 位文件大小和偏移量，允许客户端访问超过 2 GB 的文件数据。
- NFS 版本 4 (NFSv4)通过防火墙，并在 Internet 上工作，不再需要 rpcbind 服务，支持 ACL，并且使用有状态操作。

自 Red Hat Enterprise Linux 7.4 发布以来，Red Hat Enterprise Linux 完全支持 NFS 版本 4.2 (NFSv4.2)。

以下是 Red Hat Enterprise Linux 中的 NFSv4.2 的功能：

- 稀疏文件 ：它验证文件的空间效率，并允许占位符提高存储效率。它是具有一个或多个漏洞的文件；漏洞是未分配的或未初始化的数据块，仅包含零。lseek() NFSv4.2 中的操作支持 seek_hole() 和 seek_data()，它允许应用程序在稀疏文件中映射漏洞的位置。
- 空间保留：它允许存储服务器保留空闲空间，从而禁止服务器耗尽空间。NFSv4.2 支持 allocate() 操作来保留空间、deallocate() 操作，以及 fallocate() 操作来预分配或取消分配文件中的空间。
- 标记的 NFS：它会强制实施数据访问权限，并为 NFS 文件系统上的单个文件在客户端和服务器之间启用 SELinux 标签。
- 布局增强：NFSv4.2 提供新的操作 layoutstats()，客户端可以使用它来通知元数据服务器有关其与布局的通信。

7.4 之前的 Red Hat Enterprise Linux 版本支持到版本 4.1 的 NFS。

以下是 NFSv4.1 的功能：

- 提高了网络的性能和安全性，还包括对并行 NFS (pNFS)的客户端支持。
- 对于回调不再需要单独的 TCP 连接，这允许 NFS 服务器授予委托，即使它无法联系客户端。例如，当 NAT 或防火墙干扰时。
- 它只提供一次语义（除重启操作外），防止之前的问题，当回复丢失并且操作被发送两次时，某些操作可能会返回不准确的结果。

NFS 客户端默认尝试使用 NFSv4.1 挂载，并在服务器不支持 NFSv4.1 时回退到 NFSv4.0。当服务器不支持 NFSv4.0 时，挂载将回退到 NFSv3。

注意

红帽不再支持 NFS 版本 2(NFSv2)。

所有版本的 NFS 都可以使用运行在 IP 网络上的传输控制协议 (TCP)，NFSv4 需要此协议。NFSv3 可以使用在 IP 网络上运行的用户数据报协议 (UDP) 在客户端和服务器之间提供无状态网络连接。

将 NFSv3 与 UDP 一起使用时，无状态 UDP 连接（在正常情况下）协议的开销比 TCP 少。这可在非常干净、无拥塞的网络上转化为更佳的性能。但是，由于 UDP 是无状态的，因此如果服务器意外停机，UDP 客户端将继续向网络发送对服务器的请求。此外，当使用 UDP 丢失帧时，必须重新传输整个 RPC 请求；使用 TCP 时，仅需要重新传输丢失的帧。因此，当连接 NFS 服务器时，TCP 是首选的协议。

挂载和锁定协议已合并到 NFSv4 协议中。该服务器还会监听已知的 TCP 端口 2049。因此，NFSv4 不需要与 rpcbind 交互[1]、lockd 和 rpc.statd 守护进程。NFS 服务器上仍然需要 rpc.mountd 守护进程来设置导出，但不涉及任何在线操作。

注意

TCP 是 Red Hat Enterprise Linux 下 NFS 版本 3 的默认传输协议。UDP 可以根据需要用于兼容性目的，但不建议广泛使用。NFSv4 需要 TCP。

所有 RPC/NFS 守护进程都有一个 '-p' 命令行选项，可以设置端口，使防火墙配置更容易。

在 TCP 包装程序授予对客户端的访问权限后，NFS 服务器指的是 /etc/exports 配置文件，以确定是否允许客户端访问任何导出的文件系统。一旦被验证，所有文件和目录操作都对用户有效。

重要

为了使 NFS 使用启用了防火墙的默认 Red Hat Enterprise Linux 安装，请使用默认 TCP 端口 2049 配置 IPTables。如果没有合适的 IPTables 配置，NFS 无法正常工作。

NFS 初始化脚本和 rpc.nfsd 进程现在允许在系统启动期间绑定到任何指定的端口。但是，如果端口不可用，或者与其他守护进程有冲突，这可能会出错。

8.1.1. 所需的服务

Red Hat Enterprise Linux 使用内核级支持和守护进程的组合来提供 NFS 文件共享。所有 NFS 版本都依赖于客户端和服务器之间的远程过程调用 (RPC)。Red Hat Enterprise Linux 7 中的 RPC 服务由 rpcbind 服务控制。要共享或者挂载 NFS 文件系统，下列服务根据所使用的 NFS 版本而定：

注意

portmap 服务用于将 RPC 程序号映射到早期版本的 Red Hat Enterprise Linux 中的 IP 地址端口号。现在，在 Red Hat Enterprise Linux 7 中，该服务被 rpcbind 替代来启用 IPv6 支持。

nfs

systemctl start nfs 启动 NFS 服务器，以及为共享 NFS 文件系统请求提供服务的适当 RPC 进程。

nfslock

systemctl start nfs-lock 激活一个强制服务，该服务启动适当的 RPC 进程，允许 NFS 客户端锁定服务器上的文件。

rpcbind

rpcbind 接受本地 RPC 服务的端口保留。这些端口随后可用（或发布），以便相应的远程 RPC 服务可以访问它们。rpcbind 响应对 RPC 服务的请求，并设置到请求的 RPC 服务的连接。这不能与 NFSv4 一起使用。

以下 RPC 进程有助于 NFS 服务：

rpc.mountd

NFS 服务器使用这个进程来处理来自 NFSv3 客户端的 MOUNT 请求。它检查所请求的 NFS 共享是否目前由 NFS 服务器导出，并且允许客户端访问它。如果允许挂载请求，rpc.mountd 服务器会以 Success 状态回复，并将这个 NFS 共享的文件处理返回给 NFS 客户端。

rpc.nfsd

rpc.nfsd 允许定义服务器公告的显式 NFS 版本和协议。它与 Linux 内核配合使用，以满足 NFS 客户端的动态需求，例如在每次 NFS 客户端连接时提供服务器线程。这个进程对应于 nfs 服务。

lockd

lockd 是一个在客户端和服务器上运行的内核线程。它实现了 网络锁定管理器 (NLM)协议，它允许 NFSv3 客户端锁定服务器上的文件。每当运行 NFS 服务器以及挂载 NFS 文件系统时，它会自动启动。

rpc.statd

这个进程实现 网络状态监控器 (NSM)RPC 协议，该协议在 NFS 服务器没有正常关闭而重新启动时，通知 NFS 客户端。RPC.statd 由 nfslock 服务自动启动，不需要用户配置。这不能与 NFSv4 一起使用。

rpc.rquotad

这个过程为远程用户提供用户配额信息。RPC.rquotad 由 nfs 服务自动启动，不需要用户配置。

rpc.idmapd

rpc.idmapd 提供 NFSv4 客户端和服务器上调用，这些调用在线 NFSv4 名称（以 user@domain形式的字符串）和本地 UID 和 GID 之间进行映射。要使 idmapd 与 NFSv4 正常工作，必须配置 /etc/idmapd.conf 文件。至少应指定"Domain"参数，该参数定义 NFSv4 映射域。如果 NFSv4 映射域与 DNS 域名相同，可以跳过这个参数。客户端和服务器必须同意 NFSv4 映射域才能使 ID 映射正常工作。

注意

在 Red Hat Enterprise Linux 7 中，只有 NFSv4 服务器使用 rpc.idmapd。NFSv4 客户端使用基于密钥环的 idmapper nfsidmap。nfsidmap 是一个独立程序，由内核按需调用来执行 ID 映射；它不是一个守护进程。如果 nfsidmap 出现问题，客户端会回退到使用 rpc.idmapd。有关 nfsidmap 的更多信息，请参阅 nfsidmap 手册页。

8.2. 配置 NFS 客户端

mount 命令在客户端上挂载 NFS 共享。其格式如下：

```
# mount -t nfs -o options server:/remote/export /local/directory
```

这个命令使用以下变量：

options

以逗号分隔的挂载选项列表；有关有效 NFS 挂载选项的详情，请参考 第 8.4 节 “常用 NFS 挂载选项”。

server

导出您要挂载的文件系统的服务器的主机名、IP 地址或完全限定域名

/remote/export

从 server 导出的文件系统或目录，即您要挂载的目录

/local/directory

挂载 /remote/export 的客户端位置

Red Hat Enterprise Linux 7 中使用的 NFS 协议版本由 挂载选项 nfsvers 或 vers 标识。默认情况下，mount 使用带有 mount -t nfs 的 NFSv4。如果服务器不支持 NFSv4，客户端会自动缩减到服务器支持的版本。如果使用 nfsvers/vers 选项传递服务器不支持的特定版本，挂载会失败。文件系统类型

nfs4 也可用于旧原因；这等同于运行 mount -t nfs -o nfsvers=4 host:/remote/export /local/directory。

如需更多信息，请参阅 man mount。

如果 NFS 共享是手动挂载的，则重新引导后不会自动挂载共享。Red Hat Enterprise Linux 提供了两种方法来在引导时自动挂载远程文件系统：/etc/fstab 文件和 autofs 服务。如需更多信息，请参阅 第 8.2.1 节 “使用 /etc/fstab挂载 NFS 文件系统” 和 第 8.3 节 “autofs”。

8.2.1. 使用 /etc/fstab挂载 NFS 文件系统

从另一台计算机挂载 NFS 共享的另一个方法是向 /etc/fstab 文件中添加一行。行必须指定 NFS 服务器的主机名、要导出的服务器上的目录，以及 NFS 共享要挂载在本地计算机上的目录。您必须是 root 用户才能修改 /etc/fstab 文件。

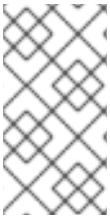
> 例 8.1. 语法示例
>
> /etc/fstab 中行的一般语法如下：
>
> ```
> server:/usr/local/pub    /pub   nfs    defaults 0 0
> ```

在执行此命令之前，客户端计算机上必须存在挂载点 /pub。将这一行添加到客户端系统上的 /etc/fstab 后，使用命令 mount /pub，挂载点 /pub 是从服务器挂载的。

挂载 NFS 导出的有效 /etc/fstab 条目应包含以下信息：

```
server:/remote/export /local/directory nfs options 0 0
```

变量 server 、/remote/export、/local/directory 和 options 与手动挂载 NFS 共享时所用的一样。如需更多信息，请参阅 第 8.2 节 “配置 NFS 客户端”。



注意

在读取 /etc/fstab 之前，客户端上必须存在挂载点 /local/directory。否则，挂载会失败。

编辑 /etc/fstab 后，重新生成挂载单元，以便您的系统注册新配置：

```
# systemctl daemon-reload
```

其它资源

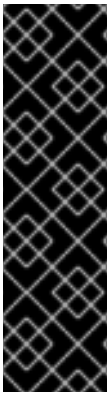
- 有关 /etc/fstab 的详情，请参考 man fstab。

8.3. AUTOFS

使用 /etc/fstab 的一个缺点是，无论用户如何经常访问 NFS 挂载的文件系统，系统都必须指定资源来保持挂载的文件系统。对于一个或两个挂载没有问题，但当系统一次维护多个系统的挂载时，整体的系统性能可能会受到影响。/etc/fstab 的替代方法是使用基于内核的 automount 工具。自动挂载程序由两个组件组成：

- 实现文件系统的内核模块，以及
- 执行所有其他功能的用户空间守护进程。

automount 工具可以自动挂载和卸载 NFS 文件系统（按需挂载），从而节省了系统资源。它可用于挂载其他文件系统，包括 AFS、SMBFS、CIFS、CIFS 和本地文件系统。



重要

nfs-utils 软件包现在是"NFS 文件服务器"和"网络文件系统客户端"组的一部分。因此，默认情况下，它不再与 Base 组一起安装。尝试自动挂载 NFS 共享之前，请先确保系统上已安装 nfs-utils。

autofs 也是"网络文件系统客户端"组的一部分。

autofs 使用 /etc/auto.master （主映射）作为其默认主配置文件。可以使用 autofs 配置（在 /etc/sysconfig/autofs中）与名称服务交换机(NSS)机制结合使用其他支持的网络源和名称。为主映射中配置的每个挂载点运行一个 autofs 版本 4 守护进程的实例，因此可以在命令行中针对任何给定挂载点手动运行。autofs 版本 5 无法实现，因为它使用单个守护进程来管理所有配置的挂载点；因此，必须在主映射中配置所有自动挂载。这符合其他行业标准自动挂载程序的常规要求。挂载点、主机名、导出的目录和选项都可以在一组文件（或其他支持的网络源）中指定，而不必为每个主机手动配置它们。

8.3.1. 相对于版本 4， autofs 版本 5 的改进

autofs 版本 5 与版本 4 相比有以下改进：

直接映射支持

autofs 中的直接映射提供了一种在文件系统层次结构中的任意点自动挂载文件系统的机制。直接映射由主映射中的挂载点 /- 表示。直接映射中的条目包含一个作为键的绝对路径名称（而不是间接映射中使用的相对路径名称）。

Lazy 挂载和卸载支持

多挂载映射条目描述了单个键下的挂载点层次结构。一个很好的例子是 -hosts 映射，通常用于将主机 /net/主机中 的所有导出自动挂载为多挂载映射条目。使用 -hosts 映射时，/net/主机的 ls 将为每个来自 主机 的导出挂载 autofs 触发器挂载。然后，它们将在被访问时进行挂载并使其过期。这可大幅减少访问具有大量导出的服务器时所需的活动挂载数量。

增强的 LDAP 支持

autofs 配置文件(/etc/sysconfig/autofs)提供了一种机制来指定站点实施的 autofs 模式，从而防止在应用程序本身中通过试用和错误来确定这一点。此外，现在支持对 LDAP 服务器进行身份验证的绑定，使用常见 LDAP 服务器实现支持的大多数机制。为此支持添加了一个新的配置文件：/etc/autofs_ldap_auth.conf。默认的配置文件是自文档文件，使用 XML 格式。

正确使用名称服务切换(nsswitch)配置。

名称服务切换配置文件的存在是为了提供一种方法来确定特定的配置数据来自哪里。这种配置的原因是让管理员可以灵活地使用选择的后端数据库，同时维护统一的软件接口来访问数据。尽管在处理 NSS 配置时，版本 4 自动挂载程序变得越来越好，但它仍然不完整。另一方面，autofs 版本 5 是一个完整的实现。

有关此文件支持的语法的更多信息，请参阅 man nsswitch.conf。并非所有 NSS 数据库都是有效的映射源，解析器将拒绝无效的映射源。有效的源是 file, yp,nis,nisplus,ldap, 和 hesiod。

每个 autofs 挂载点都有多个主映射条目

经常使用但还没有提及的一件事是处理直接挂载点 /- 的多个主映射条目。每个条目的映射键被合并，并表现为一个映射的形式。

例 8.2. 每个 autofs 挂载点都有多个主映射条目

以下是直接挂载的 connectathon 测试映射中的示例：

```
/- /tmp/auto_dcthon
/- /tmp/auto_test3_direct
/- /tmp/auto_test4_direct
```

8.3.2. 配置 autofs

自动挂载程序的主要配置文件是 /etc/auto.master，也称为主映射，如 第 8.3.1 节 “相对于版本 4，autofs 版本 5 的改进” 所述。主映射列出了系统上 autofs- 控制的挂载点，以及它们相应的配置文件或网络来源，称为自动挂载映射。master 映射的格式如下：

```
mount-point map-name options
```

使用这种格式的变量有：

mount-point

例如，autofs 挂载点 /home。

map-name

包含挂载点列表的映射源的名称，以及挂载这些挂载点的文件系统的位置。

options

如果提供，它们适用于给定映射中的所有条目，只要它们本身没有指定选项。这个行为与 autofs 版本 4 不同，其中选项是累积的。这已被修改来实现混合环境兼容性。

例 8.3. /etc/auto.master 文件

以下是来自 /etc/auto.master 文件的示例行（使用 cat /etc/auto.master显示）：

```
/home /etc/auto.misc
```

映射的常规格式与主映射类似，但"选项"会出现在挂载点和位置之间，而不是在主映射中的条目末尾：

```
mount-point [options] location
```

使用这种格式的变量有：

mount-point

这指的是 autofs 挂载点。这可以是间接挂载的单个目录名称，也可以是直接挂载的挂载点的完整路径。每个直接和间接映射条目键(挂载点)后面可以跟一个以空格分隔的偏移目录列表（每个以 /开头的子目录名称）使其称为多挂载条目。

options

每当提供时，这些都是未指定其自身选项的映射条目的挂载选项。

位置

这指的是文件系统位置，如本地文件系统路径（对于以 /开头的映射名称，前面带有 Sun 映射格式转义字符":"）、NFS 文件系统或其他有效的文件系统位置。

以下是映射文件（例如 /etc/auto.misc）中的内容示例：

```
payroll -fstype=nfs personnel:/dev/hda3
sales -fstype=ext3 :/dev/hda4
```

映射文件中的第一列指示 autofs 挂载点（来自名为 personnel 的服务器的sales 和 payroll ）。第二列显示 autofs 挂载的选项，第三列则指示挂载的来源。在给定配置后，autofs 挂载点将是 /home/payroll 和 /home/sales。通常省略 -fstype= 选项，通常不需要正确的操作。

如果目录不存在，自动挂载程序会创建它们。如果在自动挂载程序启动之前目录已存在，则自动挂载程序在退出时不会删除它们。

要启动自动挂载守护进程，请使用以下命令：

```
# systemctl start autofs
```

要重启自动挂载守护进程，请使用以下命令：

```
# systemctl restart autofs
```

使用给定配置时，如果进程需要访问 autofs 卸载的目录，如 /home/payroll/2006/July.sxc，则自动挂载守护进程会自动挂载该目录。如果指定了超时，则如果在超时时间内没有访问该目录，则目录会被自动卸载。

要查看自动挂载守护进程的状态，请使用以下命令：

```
# systemctl status autofs
```

8.3.3. 覆盖或增加站点配置文件

覆盖客户端系统上的特定挂载点的站点默认值会很有用。例如，请考虑以下条件：

- 自动挂载程序映射存储在 NIS 中，/etc/nsswitch.conf 文件具有以下指令：

  ```
  automount:  files nis
  ```

- auto.master 文件包含：

  ```
  +auto.master
  ```

- NIS auto.master 映射文件包含：

  ```
  /home auto.home
  ```

- NIS auto.home 映射包含：

```
beth        fileserver.example.com:/export/home/beth
joe         fileserver.example.com:/export/home/joe
*       fileserver.example.com:/export/home/&
```

- 文件映射 /etc/auto.home 不存在。

鉴于这些条件，我们假设客户端系统需要覆盖 NIS 映射 auto.home 并从其他服务器挂载主目录。在这种情况下，客户端需要使用以下 /etc/auto.master 映射：

```
/home /etc/auto.home
+auto.master
```

/etc/auto.home 映射包含条目：

```
*    labserver.example.com:/export/home/&
```

由于自动挂载程序仅处理第一次出现的挂载点，/home 包含 /etc/auto.home 的内容，而不是 NIS auto.home 映射。

另外，要仅使用几个条目来增加站点范围的 auto.home 映射，请创建一个 /etc/auto.home 文件映射，并在其中放置新条目。在结尾处，包含 NIS auto.home 映射。然后 /etc/auto.home 文件映射类似：

```
mydir someserver:/export/mydir
+auto.home
```

使用这些 NIS auto.home 映射条件时，ls /home 命令输出：

```
beth joe mydir
```

最后一个示例按预期工作，因为 autofs 不包含与正在读取的文件映射同名的文件映射的内容。因此，autofs 转到 nsswitch 配置中的下一个映射源。

8.3.4. 使用 LDAP 来存储自动挂载程序映射

必须在所有配置的系统中安装 LDAP 客户端程序库，以便从 LDAP 检索自动挂载程序映射。在 Red

Hat Enterprise Linux 上，openldap 软件包应作为 自动挂载程序 的依赖项自动安装。要配置 LDAP 访问，请修改 /etc/openldap/ldap.conf。确保为您的站点正确设置了 BASE、URI 和 模式。

rfc2307bis 描述了在 LDAP 中存储自动挂载映射的最新建立的模式。要使用此模式，必须通过从架构定义中删除注释字符，在 autofs 配置(/etc/sysconfig/autofs)中设置它。例如：

例 8.4. 设置 autofs 配置

```
DEFAULT_MAP_OBJECT_CLASS="automountMap"
DEFAULT_ENTRY_OBJECT_CLASS="automount"
DEFAULT_MAP_ATTRIBUTE="automountMapName"
DEFAULT_ENTRY_ATTRIBUTE="automountKey"
DEFAULT_VALUE_ATTRIBUTE="automountInformation"
```

确保它们是唯一在配置中未注释的模式条目。automountKey 替换 rfc2307bis 模式中的 cn 属性。以下是 LDAP 数据交换格式(LDIF)配置示例：

例 8.5. LDF 配置

```
# extended LDIF
#
# LDAPv3
# base <> with scope subtree
# filter: (&(objectclass=automountMap)(automountMapName=auto.master))
# requesting: ALL
#

# auto.master, example.com
dn: automountMapName=auto.master,dc=example,dc=com
objectClass: top
objectClass: automountMap
automountMapName: auto.master

# extended LDIF
#
# LDAPv3
# base <automountMapName=auto.master,dc=example,dc=com> with scope subtree
# filter: (objectclass=automount)
# requesting: ALL
#

# /home, auto.master, example.com
dn: automountMapName=auto.master,dc=example,dc=com
objectClass: automount
cn: /home

automountKey: /home
automountInformation: auto.home
```

```
# extended LDIF
#
# LDAPv3
# base <> with scope subtree
# filter: (&(objectclass=automountMap)(automountMapName=auto.home))
# requesting: ALL
#

# auto.home, example.com
dn: automountMapName=auto.home,dc=example,dc=com
objectClass: automountMap
automountMapName: auto.home

# extended LDIF
#
# LDAPv3
# base <automountMapName=auto.home,dc=example,dc=com> with scope subtree
# filter: (objectclass=automount)
# requesting: ALL
#

# foo, auto.home, example.com
dn: automountKey=foo,automountMapName=auto.home,dc=example,dc=com
objectClass: automount
automountKey: foo
automountInformation: filer.example.com:/export/foo

# /, auto.home, example.com
dn: automountKey=/,automountMapName=auto.home,dc=example,dc=com
objectClass: automount
automountKey: /
automountInformation: filer.example.com:/export/&
```

8.4. 常用 NFS 挂载选项

除了在远程主机上使用 NFS 挂载文件系统外，也可以在挂载时指定其他选项，使挂载的共享更易于使用。这些选项可用于 手动挂载 命令、/etc/fstab 设置和 autofs。

以下是 NFS 挂载常用的选项：

lookupcache=mode

指定内核应该如何管理给定挂载点的目录条目缓存。模式 的有效参数为 all、none 或 pos/positive。

nfsvers=version

指定要使用的 NFS 协议版本，其中 version 为 3 或 4。这对于运行多个 NFS 服务器的主机很有用。如果没有指定版本，NFS 将使用内核和 mount 命令支持的最高版本。

选项 vers 等同于 nfsvers，出于兼容性的原因包含在此发行版本中。

noacl

关闭所有 ACL 处理。当与旧版本的 Red Hat Enterprise Linux、Red Hat Linux 或 Solaris 交互时，可能会需要此功能，因为最新的 ACL 技术与较旧的系统不兼容。

nolock

禁用文件锁定。当连接到非常旧的 NFS 服务器时，有时需要这个设置。

noexec

防止在挂载的文件系统中执行二进制文件。这在系统挂载不兼容二进制文件的非 Linux 文件系统时有用。

nosuid

禁用 set-user-identifier 或 set-group-identifier 位。这可防止远程用户通过运行 setuid 程序获得更高的特权。

port=num

指定 NFS 服务器端口的数字值。如果 num 是 0 (默认值)，则 mount 会查询远程主机的 rpcbind 服务，以获取要使用的端口号。如果远程主机的 NFS 守护进程没有注册到其 rpcbind 服务，则会使用标准 NFS 端口号 TCP 2049。

rsize=num 和 wsize=num

这些选项设定在单个 NFS 读取或写入操作中传输的最大字节数。

rsize 和 wsize 没有固定的默认值。默认情况下，NFS 使用服务器和客户端都支持的最大的可能值。在 Red Hat Enterprise Linux 7 中，客户端和服务器的最大值为 1,048,576 字节。详情请查看

rsize 和 wsize 的默认和最大值是什么？Kbase 文章。

sec=flavors

用于访问挂载导出上文件的安全类别。flavors 值是以冒号分隔的一个或多个安全类型列表。

默认情况下，客户端会尝试查找客户端和服务器都支持的安全类别。如果服务器不支持任何选定的类别，挂载操作将失败。

sec=sys 使用本地 UNIX UID 和 GID。它们使用 AUTH_SYS 验证 NFS 操作。

sec=krb5 使用 Kerberos V5 ,而不是本地 UNIX UID 和 GID 来验证用户。

sec=krb5i 使用 Kerberos V5 进行用户身份验证，并使用安全校验和执行 NFS 操作的完整性检查，以防止数据被篡改。

sec=krb5p 使用 Kerberos V5 进行用户身份验证、完整性检查，并加密 NFS 流量以防止流量嗅探。这是最安全的设置，但它也会涉及最大的性能开销。

tcp

指示 NFS 挂载使用 TCP 协议。

udp

指示 NFS 挂载使用 UDP 协议。

如需更多信息，请参阅 man mount 和 man nfs。

8.5. 启动和停止 NFS 服务器

先决条件

- 对于支持 NFSv2 或 NFSv3 连接的服务器，rpcbind[1] 服务必须正在运行。要验证 rpcbind 是否处于活动状态，请使用以下命令：

```
$ systemctl status rpcbind
```

要配置只使用 NFSv4 的服务器，它不需要 rpcbind，请参阅 第 8.6.7 节 “配置只使用 NFSv4 的服务器”。

- 在 Red Hat Enterprise Linux 7.0 中，如果您的 NFS 服务器导出 NFSv3 并在引导时启用，则需要手动启动并启用 nfs-lock 服务：

```
# systemctl start nfs-lock
# systemctl enable nfs-lock
```

在 Red Hat Enterprise Linux 7.1 及更高版本中，如果需要，nfs-lock 会自动启动，并尝试手动启用它。

流程

- 要启动 NFS 服务器，请使用以下命令：

```
# systemctl start nfs
```

- 要使 NFS 在引导时启动，请使用以下命令：

```
# systemctl enable nfs
```

- 要停止服务器，请使用：

```
# systemctl stop nfs
```

- restart 选项是停止然后启动 NFS 的简写方式。这是编辑 NFS 配置文件后使配置更改生效的最有效方式。要重启服务器的类型：

```
# systemctl restart nfs
```

- 编辑 /etc/sysconfig/nfs 文件后，运行以下命令来重启 nfs-config 服务，使新值生效：

```
# systemctl restart nfs-config
```

- try-restart 命令仅在当前正在运行时启动 nfs。此命令等同于 Red Hat init 脚本中的 condrestart (有条件重启)非常有用，因为它不会在 NFS 未运行时启动守护进程。

要有条件地重启服务器，请输入：

```
# systemctl try-restart nfs
```

- 要在不重启服务类型的情况下重新载入 NFS 服务器配置文件：

```
# systemctl reload nfs
```

8.6. 配置 NFS 服务器

在 NFS 服务器中配置导出的方法有两种：

- 手动编辑 NFS 配置文件，即 /etc/exports，以及
- 通过命令行，即使用 exportfs命令

8.6.1. /etc/exports 配置文件

/etc/exports 文件控制哪些文件系统被导出到远程主机，并指定选项。它遵循以下语法规则：

- 空白行将被忽略。
- 要添加注释，以井号(#)开始一行。
- 您可以使用反斜杠(\)换行长行。

- 每个导出的文件系统都应该独立。
- 所有在导出的文件系统后放置的授权主机列表都必须用空格分开。
- 每个主机的选项必须在主机标识符后直接放在括号中，没有空格分离主机和第一个括号。

导出的文件系统的每个条目都有以下结构：

```
export host(options)
```

上述结构使用以下变量：

export

导出的目录

host

导出要共享的主机或网络

options

用于 **host** 的选项

可以指定多个主机，以及每个主机的特定选项。要做到这一点，将它们列在与用空格分隔的列表相同的行中，每个主机名后跟其各自的选项（在括号中），如下所示：

```
export host1(options1) host2(options2) host3(options3)
```

有关指定主机名的不同方法的详情，请参考 第 8.6.5 节 “主机名格式”。

在最简单的形式中，/etc/exports 文件只指定导出的目录和允许访问它的主机，如下例所示：

例 8.6. /etc/exports 文件

```
/exported/directory bob.example.com
```

在这里，bob.example.com 可以从 NFS 服务器挂载 /exported/directory/。因为在这个示例中没有指定选项，所以 NFS 将 使用默认设置。

默认设置为：

ro

导出的文件系统是只读的。远程主机无法更改文件系统中共享的数据。要允许主机更改文件系统（即读写），请指定 rw 选项。

同步

在将之前的请求所做的更改写入磁盘前，NFS 服务器不会回复请求。要启用异步写入，请指定 async 选项。

wdelay

如果 NFS 服务器预期另外一个写入请求即将发生，则 NFS 服务器会延迟写入磁盘。这可以提高性能，因为它可减少不同的写命令访问磁盘的次数，从而减少写开销。要禁用此功能，请指定 no_wdelay。只有指定了默认的 sync 选项时，no_wdelay 才可用。

root_squash

这可防止远程连接的 root 用户（而不是本地）具有 root 权限；相反，NFS 服务器会为他们分配用户 ID nfsnobody。这可以有效地将远程 root 用户的权限"挤压"成最低的本地用户，从而防止在远程服务器上可能的未经授权的写操作。要禁用 root 压缩，请指定 no_root_squash。

要对所有远程用户（包括 root 用户）进行压缩，请使用 all_squash ：要指定 NFS 服务器应该分配给来自特定主机的远程用户的用户 ID 和组 ID，请分别使用 anonuid 和 anongid 选项，如下所示：

```
export host(anonuid=uid,anongid=gid)
```

这里，uid 和 gid 分别是用户 ID 号和组 ID 号。anonuid 和 anongid 选项允许您创建特殊的用户和组帐户，为远程 NFS 用户共享。

默认情况下，Red Hat Enterprise Linux 下的 NFS 支持访问控制列表 (ACL)。要禁用此功能，在导出文件系统时请指定 no_acl 选项。

每个导出的文件系统的默认值都必须被显式覆盖。例如，如果没有指定 rw 选项，则导出的文件系统将以只读形式共享。以下是 /etc/exports 中的示例行，其覆盖两个默认选项：

```
/another/exported/directory 192.168.0.3 (rw,async)
```

在这个示例中，192.168.0.3 可以挂载 /another/exported/directory/ 读写，对磁盘的所有写入都是异步的。有关导出选项的更多信息，请参阅 man exportfs。

其他选项在未指定默认值的情况下可用。这包括禁用子树检查、允许来自不安全端口的访问以及允许不安全的文件锁（对于某些早期 NFS 客户端实现是必需的）。有关这些较少使用的选项的更多信息，请参阅 man 导出。

重要

/etc/exports 文件的格式要求非常精确，特别是在空格字符的使用方面。需要将导出的文件系统与主机、不同主机间使用空格分隔。但是，除了注释行外，文件中不应该包括其他空格。

例如，下面两行并不具有相同的意义：

```
/home bob.example.com(rw)
/home bob.example.com (rw)
```

第一行只允许 bob.example.com 中的用户对 /home 目录进行读写访问。第二行允许来自 bob.example.com 的用户以只读方式挂载目录（默认），而其他用户可以将其挂载为读/写。

8.6.2. exportfs 命令

使用 NFS 将每个文件系统导出到远程用户，以及这些文件系统的访问级别，均列在 /etc/exports 文件中。当 nfs 服务启动时，/usr/sbin/exportfs 命令启动并读取此文件，将控制传递给实际挂载进程的 rpc.mountd （如果 NFSv3），然后传给 rpc.nfsd，然后供远程用户使用。

手动发出时，/usr/sbin/exportfs 命令允许 root 用户有选择地导出或取消导出目录，而无需重启 NFS 服务。给定正确的选项后，/usr/sbin/exportfs 命令将导出的文件系统写入 /var/lib/nfs/xtab。由于 rpc.mountd 在决定对文件系统的访问权限时引用 xtab 文件，因此对导出的文件系统列表的更改会立即生效。

以下是 /usr/sbin/exportfs 常用的选项列表：

-r

通过在 /var/lib/nfs/etab 中构建新的导出列表，将 /etc/exports 中列出的所有目录导出。如果对 /etc/exports 做了任何更改，这个选项可以有效地刷新导出列表。

-a

根据将哪些其他选项传给 /usr/sbin/exportfs，导致所有目录被导出或取消导出。如果没有指定其他选项，/usr/sbin/exportfs 会导出 /etc/exports 中指定的所有文件系统。

-o file-systems

指定没有在 /etc/exports 中列出的要导出的目录。将 file-systems 替换为要导出的其它文件系统。这些文件系统的格式化方式必须与 /etc/exports 中指定的方式相同。此选项通常用于在将导出的文件系统永久添加到导出的文件系统列表之前，对其进行测试。有关 /etc/exports 语法的详情，请参考 第 8.6.1 节 “/etc/exports 配置文件”。

-i

忽略 /etc/exports ；只有命令行上指定的选项才会用于定义导出的文件系统。

-u

不导出所有共享目录。命令 /usr/sbin/exportfs -ua 可暂停 NFS 文件共享，同时保持所有 NFS 守护进程正常运行。要重新启用 NFS 共享，请使用 exportfs -r。

-v

详细操作，当执行 exportfs 命令时，更详细地显示正在导出的或取消导出的文件系统。

如果没有将选项传给 exportfs 命令，它将显示当前导出的文件系统列表。有关 exportfs 命令的详情，请参考 man exportfs。

8.6.2.1. 使用带有 NFSv4 的 exportfs

在 Red Hat Enterprise Linux 7 中，不需要额外的步骤来配置 NFSv4 导出，因为上述任何文件系统都自动提供给使用相同路径的 NFSv3 和 NFSv4 客户端。在之前的版本中并非如此。

要防止客户端使用 NFSv4，请在 /etc/sysconfig/nfs 中设置 RPCNFSDARGS= -N 4 来将其关闭。

8.6.3. 在防火墙后面运行 NFS

NFS 需要 rpcbind，它可为 RPC 服务动态分配端口，并可能导致配置防火墙规则的问题。要允许客户端访问防火墙后面的 NFS 共享，请编辑 /etc/sysconfig/nfs 文件来设置 RPC 服务运行的端口。要允许客户端通过防火墙访问 RPC 配额，请参阅 第 8.6.4 节 “通过防火墙访问 RPC 配额”。

默认情况下，/etc/sysconfig/nfs 文件在所有系统中都不存在。如果 /etc/sysconfig/nfs 不存在，请创建并指定以下内容：

RPCMOUNTDOPTS="-p port"

这会将"-p port"添加到 rpc.mount 命令行： rpc.mount -p port。

要指定 nlockmgr 服务使用的端口，请在 /etc/modprobe.d/lockd.conf 文件中设置 nlm_tcpport 和 nlm_udpport 选项的端口号。

如果 NFS 无法启动，请检查 /var/log/messages。通常，如果指定了已在使用的端口号，NFS 将无法启动。编辑 /etc/sysconfig/nfs 后，您需要重启 nfs-config 服务，以便新值在 Red Hat Enterprise Linux 7.2 中生效，然后再运行以下命令：

```
# systemctl restart nfs-config
```

然后，重启 NFS 服务器：

```
# systemctl restart nfs-server
```

运行 rpcinfo -p 以确认更改生效。

注意

要允许 NFSv4.0 回调通过防火墙设置 /proc/sys/fs/nfs/nfs_callback_tcpport，并允许服务器连接到客户端上的该端口。

NFSv4.1 或更高版本不需要这个过程，纯 NFSv4 环境中不需要 mountd、statd 和 lockd 的其他端口。

8.6.3.1. 发现 NFS 导出

有两种方法可以发现 NFS 服务器导出了哪些文件系统。

- 在支持 NFSv3 的任何服务器上，使用 showmount 命令：

```
$ showmount -e myserver
Export list for mysever
/exports/foo
/exports/bar
```

- 在支持 NFSv4 的任何服务器上，挂载 根目录并查找。

```
# mount myserver:/ /mnt/
# cd /mnt/
exports
# ls exports
foo
bar
```

在同时支持 NFSv4 和 NFSv3 的服务器上，这两种方法都可以工作，并给出同样的结果。

注意

在较旧的 NFS 服务器上的 Red Hat Enterprise Linux 6 之前，具体取决于它们的配置方式，可以通过不同的路径将文件系统导出到 NFSv4 客户端。由于这些服务器默认没有启用 NFSv4，因此这不应有问题。

8.6.4. 通过防火墙访问 RPC 配额

如果您导出使用磁盘配额的文件系统，您可以使用配额远程过程调用(RPC)服务来给 NFS 客户端提供磁盘配额数据。

过程 8.1. 使 RPC 配额访问防火墙不可行

1. 要启用 rpc-rquotad 服务，请使用以下命令：

```
# systemctl enable rpc-rquotad
```

2. 要启动 rpc-rquotad 服务，请使用以下命令：

```
# systemctl start rpc-rquotad
```

请注意，如果启用 rpc-rquotad，则自动启动 nfs-server 服务。

3. 要使配额 RPC 服务在防火墙后面访问，需要打开 UDP 或 TCP 端口 875。默认端口号定义在 /etc/services 文件中。

您可以通过将 -p port-number 附加到 /etc/sysconfig/rpc-rquotad 文件中的 RPCRQUOTADOPTS 变量来覆盖默认端口号。

4. 重启 rpc-rquotad 以使 /etc/sysconfig/rpc-rquotad 文件中的更改生效：

```
# systemctl restart rpc-rquotad
```

从远程主机设置配额

默认情况下，配额只能由远程主机读取。要允许设置配额，请将 -S 选项附加到 /etc/sysconfig/rpc-rquotad 文件中的 RPCRQUOTADOPTS 变量中。

重启 rpc-rquotad 以使 /etc/sysconfig/rpc-rquotad 文件中的更改生效：

```
# systemctl restart rpc-rquotad
```

8.6.5. 主机名格式

主机可以采用以下形式：

单台机器

完全限定域名（可由服务器解析）、主机名（可由服务器解析）或 IP 地址。

使用通配符指定的一系列机器

使用 * 或 ? 字符指定字符串匹配项。通配符不能用于 IP 地址；但是，如果反向 DNS 查找失败，则可能会意外起作用。当在完全限定域名中指定通配符时，点(.)不会包含在通配符中。例如：*.example.com 包含 one.example.com，但不包括 include one.two.example.com。

IP 网络

使用 a.b.c.d/z，其中 a.b.c.d 是网络，z 是子网掩码的位数（如 192.168.0.0/24）。另一种可接受的格式是 a.b.c.d/netmask，其中 a.b.c.d 是网络，netmask 是子网掩码（例如 192.168.100.8/255.255.255.0）。

Netgroups

使用格式 @group-name ，其中 group-name 是 NIS netgroup 名称。

8.6.6. 启用通过 RDMA(NFSoRDMA) 的 NFS

如果存在支持 RDMA 的硬件，则远程直接内存访问(RDMA)服务会在 Red Hat Enterprise Linux 7 中

自动工作。

通过 RDMA 启用 NFS：

1. 安装 rdma 和 rdma-core 软件包。

   /etc/rdma/rdma.conf 文件包含默认设置 XPRTRDMA_LOAD=yes 的行，它请求 rdma 服务加载 NFSoRDMA 客户端 模块。

2. 要启用自动载入 NFSoRDMA 服务器模块，请在 /etc/rdma/rdma.conf 中的新行中添加 SVCRDMA_LOAD=yes。

   RPCNFSDARGS="--rdma=20049" 在 /etc/sysconfig/nfs 文件中指定 NFSoRDMA 服务侦听客户端的端口号。RFC 5667 指定服务器在通过 RDMA 提供 NFSv4 服务时必须侦听端口 20049。

3. 编辑 /etc/rdma/rdma.conf 文件后重启 nfs 服务：

   ```
   # systemctl restart nfs
   ```

   请注意，在较早的内核版本中，编辑 /etc/rdma/rdma.conf 后需要重启系统才能使更改生效。

8.6.7. 配置只使用 NFSv4 的服务器

默认情况下，NFS 服务器在 Red Hat Enterprise Linux 7 中支持 NFSv2、NFSv3 和 NFSv4 连接。但是，您还可以将 NFS 配置为只支持 NFS 版本 4.0 及更新的版本。这可最小化系统上开放端口的数量和运行的服务，因为 NFSv4 不需要 rpcbind 服务来侦听网络。

当您的 NFS 服务器配置为只使用 NFSv4 时，尝试使用 NFSv2 或 NFSv3 挂载共享的客户端会失败，并显示类似如下的错误：

```
Requested NFS version or transport protocol is not supported.
```

过程 8.2. 配置只使用 NFSv4 的服务器

将您的 NFS 服务器配置为只支持 NFS 版本 4.0 及更新的版本：

1. 通过在 /etc/sysconfig/nfs 配置文件中添加以下行来禁用 NFSv2、NFSv3 和 UDP：

```
RPCNFSDARGS="-N 2 -N 3 -U"
```

2. （可选）禁用对 RPCBIND、MOUNT 和 NSM 协议调用的监听，这在仅使用 NFSv4 的情况下不需要。

禁用这些选项的影响有：

- 尝试使用 NFSv2 或 NFSv3 从服务器挂载共享的客户端变得无响应。
- NFS 服务器本身无法挂载 NFSv2 和 NFSv3 文件系统。

禁用这些选项：

- 在 /etc/sysconfig/nfs 文件中添加以下内容：

```
RPCMOUNTDOPTS="-N 2 -N 3"
```

- 禁用相关服务：

```
# systemctl mask --now rpc-statd.service rpcbind.service rpcbind.socket
```

3. 重启 NFS 服务器：

```
# systemctl restart nfs
```

一旦启动或重启 NFS 服务器，这些改变就会生效。

验证仅 NFSv4 配置

您可以使用 netstat 实用程序验证您的 NFS 服务器是否在 NFSv4 模式中配置。

- *以下是仅使用 NFSv4 服务器上的 netstat 输出示例；也禁用了对 RPCBIND、MOUNT 和 NSM 的监听。在这里，nfs 是唯一侦听 NFS 服务：*

```
# netstat -ltu

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:nfs             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 localhost:smtp          0.0.0.0:*               LISTEN
tcp6       0      0 [::]:nfs                [::]:*                  LISTEN
tcp6       0      0 [::]:12432              [::]:*                  LISTEN
tcp6       0      0 [::]:12434              [::]:*                  LISTEN
tcp6       0      0 localhost:7092          [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
udp        0      0 localhost:323           0.0.0.0:*
udp        0      0 0.0.0.0:bootpc          0.0.0.0:*
udp6       0      0 localhost:323           [::]:*
```

- *相比之下，在配置只使用 NFSv4 的服务器前 netstat 输出会包括 sunrpc 和 mountd 服务：*

```
# netstat -ltu

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:nfs             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:36069           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:52364           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:sunrpc          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:mountd          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 localhost:smtp          0.0.0.0:*               LISTEN
tcp6       0      0 [::]:34941              [::]:*                  LISTEN
tcp6       0      0 [::]:nfs                [::]:*                  LISTEN
tcp6       0      0 [::]:sunrpc             [::]:*                  LISTEN
tcp6       0      0 [::]:mountd             [::]:*                  LISTEN
tcp6       0      0 [::]:12432              [::]:*                  LISTEN
tcp6       0      0 [::]:56881              [::]:*                  LISTEN
tcp6       0      0 [::]:12434              [::]:*                  LISTEN
tcp6       0      0 localhost:7092          [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
udp        0      0 localhost:323           0.0.0.0:*
udp        0      0 0.0.0.0:37190           0.0.0.0:*
udp        0      0 0.0.0.0:876             0.0.0.0:*
udp        0      0 localhost:877           0.0.0.0:*
udp        0      0 0.0.0.0:mountd          0.0.0.0:*
```

```
udp        0      0 0.0.0.0:38588           0.0.0.0:*
udp        0      0 0.0.0.0:nfs             0.0.0.0:*
udp        0      0 0.0.0.0:bootpc          0.0.0.0:*
udp        0      0 0.0.0.0:sunrpc          0.0.0.0:*
udp6       0      0 localhost:323           [::]:*
udp6       0      0 [::]:57683              [::]:*
udp6       0      0 [::]:876                [::]:*
udp6       0      0 [::]:mountd             [::]:*
udp6       0      0 [::]:40874              [::]:*
udp6       0      0 [::]:nfs                [::]:*
udp6       0      0 [::]:sunrpc             [::]:*
```

8.7. 保护 NFS

NFS 适合透明地共享具有大量已知主机的整个文件系统。但是，由于易于使用，存在各种潜在安全问题。为最大程度地降低 NFS 安全风险并保护服务器上的数据，在服务器上导出 NFS 文件系统或将其挂载到客户端上时，请考虑以下部分：

8.7.1. 具有 AUTH_SYS 和导出控制的 NFS 安全性

传统上，为了控制对导出文件的访问， NFS 提供了两个选项。

首先，服务器限制允许哪些主机通过 IP 地址或主机名挂载哪些文件系统。

其次，服务器对 NFS 客户端上的用户强制执行文件系统权限的方式与对本地用户执行的方式相同。传统上，它使用 AUTH_SYS （也称为 AUTH_UNIX）执行此操作，它依赖于客户端来声明用户的 UID 和 GID。请注意，这意味着恶意或配置错误的客户端可能会很容易地出现这个错误，并允许用户访问他不应该访问的文件。

为限制潜在的风险，管理员通常允许只读访问权限，或将用户权限压缩为公共的用户和组 ID 。不幸的是，这些解决方案阻止了以最初预期的方式使用 NFS 共享。

另外，如果攻击者获得了导出 NFS 文件系统的系统所使用的 DNS 服务器的控制，则与特定主机名或完全限定域名关联的系统可以会指向未经授权的机器。此时，未经授权的机器 是 允许挂载 NFS 共享的系统，因为没有交换用户名或密码信息来为 NFS 挂载提供额外的安全。

在通过 NFS 导出目录时应谨慎使用通配符，因为通配符的范围可能包含比预期更多的系统。

也可以使用 TCP 包装程序限制对 rpcbind[1] 服务的访问。使用 iptables 创建规则也可以限制对 rpcbind、rpc.mountd 和 rpc.nfsd 使用的端口的访问。

有关保护 NFS 和 rpcbind 的更多信息，请参阅 man iptables。

8.7.2. 带有 AUTH_GSS的 NFS 安全性

NFSv4 通过规定 RPCSEC_GSS 和 Kerberos 版本 5 GSS-API 机制的实施来促进 NFS 安全性。但是，RPCSEC_GSS 和 Kerberos 机制也可用于所有版本的 NFS。在 FIPS 模式中，只能使用 FIPS 批准的算法。

与 AUTH_SYS 不同，使用 RPCSEC_GSS Kerberos 机制，服务器不依赖于客户端就可以正确地表示哪个用户正在访问文件。相反，加密用于向服务器验证用户的身份，这可防止恶意客户端在没有用户的 Kerberos 凭据的情况下模拟用户。使用 RPCSEC_GSS Kerberos 机制是保护挂载的最直接方法，因为配置了 Kerberos 后不需要额外的设置。

配置 Kerberos

在配置 NFSv4 Kerberos 感知服务器前，您需要安装和配置 Kerberos 密钥分发中心(KDC)。Kerberos 是一种网络身份验证系统，其允许客户端和服务器使用对称加密和信任的第三方 KDC 来互相进行身份验证。红帽建议使用身份管理(IdM)来设置 Kerberos。

过程 8.3. 为 IdM 配置 NFS 服务器和客户端以使用 RPCSEC_GSS

1.
   - 在 NFS 服务器端创建 nfs/hostname.domain@REALM 主体。
   - 在服务器和客户端端创建 host/hostname.domain@REALM 主体。

     注意

     主机名 必须与 NFS 服务器主机名相同。

   - 将对应的密钥添加到客户端和服务器的 keytab 中。

   具体步骤请查看 Red Hat Enterprise Linux 7 Linux Domain Identity, Authentication, and Policy Guide 中的 Adding and Editing Service Entries and Keytabs and setting an Kerberos-aware NFS Server 部分。

2. 在服务器端，使用 sec= 选项启用所需的安全类型。启用所有安全类型和非加密挂载：

```
/export *(sec=sys:krb5:krb5i:krb5p)
```

与 sec= 选项一起使用的有效安全类型为：

- sys: 没有加密保护，默认
- krb5: authentication only
- krb5i: 完整性保护
- krb5p: 隐私保护

3. 在客户端，将 sec=krb5 （或 sec=krb5i, 或 sec=krb5p，具体取决于设置）添加到挂载选项：

```
# mount -o sec=krb5 server:/export /mnt
```

有关如何配置 NFS 客户端的详情，请参考 Red Hat Enterprise Linux 7 Linux 域身份、身份验证和策略指南中的 设置 Kerberos 感知 NFS 客户端 部分。

其它资源

- 虽然红帽建议使用 IdM，但也支持 Active Directory (AD) Kerberos 服务器。详情请查看以下红帽知识库文章：如何使用 SSSD 和 Active Directory 在 RHEL 7 中使用 Kerberos 身份验证设置 NFS。
- 如果您需要以 root 用户身份在 Kerberos 保护的 NFS 共享中写入文件，并对这些文件保留 root 所有权，请参阅 https://access.redhat.com/articles/4040141。请注意，我们不推荐进行此配置。
-

有关 NFS 客户端配置的详情请参考 exports(5) 和 nfs(5) 手册页，以及 第 8.4 节 “常用 NFS 挂载选项”。

- 有关 RPCSEC_GSS 框架的详情，包括 gssproxy 和 rpc.gssd 互操作性，请参阅 GSSD 流描述。

8.7.2.1. 使用 NFSv4 的 NFS 安全性

NFSv4 包括基于 Microsoft Windows NT 模型的 ACL 支持，而不是 POSIX 模型，因为 Microsoft Windows NT 模型的功能和广泛的部署。

NFSv4 的另一个重要安全功能是删除使用 MOUNT 协议来挂载文件系统。MOUNT 协议存在安全风险，因为协议处理文件句柄的方式。

8.7.3. 文件权限

远程主机一旦将 NFS 文件系统挂载为读取或读写，则保护每个共享文件的唯一方法就是其权限。如果共享同一用户 ID 值的两个用户挂载同一个 NFS 文件系统，则他们可以修改彼此的文件。此外，在客户端系统上以 root 身份登录的任何人都可以使用 su - 命令访问 NFS 共享的任何文件。

默认情况下，Red Hat Enterprise Linux 的 NFS 支持访问控制列表（ACL）。红帽建议保持启用此功能。

默认情况下，NFS 在导出文件系统时使用 root 压缩。这会将本地机器上以 root 用户身份访问 NFS 共享的用户 ID 设置为 nobody。root squashing 由默认选项 root_squash 控制；有关这个选项的详情请参考 第 8.6.1 节 “/etc/exports 配置文件”。如果可能的话，切勿禁用 root 压缩。

将 NFS 共享导出为只读时，请考虑使用 all_squash 选项。这个选项使访问导出的文件系统的每个用户都使用 nfsnobody 用户的用户 ID。

8.8. NFS 和 RPCBIND

注意

以下部分仅适用于需要 rpcbind 服务以便向后兼容的 NFSv3 实现。

有关如何配置只使用 NFSv4 的服务器（不需要 rpcbind ）的详情，请参考 第 8.6.7 节 “配置只使用 NFSv4 的服务器”。

rpcbind[1] 工具将 RPC 服务映射到它们侦听的端口。RPC 进程在启动时通知 rpcbind，注册它们正在侦听的端口以及它们期望提供服务的 RPC 程序号。然后，客户端系统会使用特定的 RPC 程序号联系服务器上的 rpcbind。rpcbind 服务将客户端重定向到正确的端口号，这样它就可以与请求的服务进行通信。

由于基于 RPC 的服务依赖 rpcbind 来与所有传入的客户端请求建立连接，因此 rpcbind 必须在这些服务启动之前可用。

rpcbind 服务使用 TCP 包装程序进行访问控制，而 rpcbind 的访问控制规则 会影响所有 基于 RPC 的服务。另外，也可以为每个 NFS RPC 守护进程指定访问控制规则。rpc.mountd 和 rpc.statd 的 man page 包含有关这些规则的确切语法的信息。

8.8.1. NFS 和 rpcbind故障排除

由于 rpcbind[1] 在 RPC 服务和用于与其通信的端口号之间提供协调，因此在进行故障排除时，使用 rpcbind 查看当前 RPC 服务的状态非常有用。rpcinfo 命令显示每个基于 RPC 的服务，以及端口号、RPC 程序号、版本号和 IP 协议类型(TCP 或 UDP)。

要确保为 rpcbind 启用了正确的基于 NFS RPC 的服务，请使用以下命令：

```
# rpcinfo -p
```

例 8.7. rpcinfo -p 命令输出

下面是一个这个命令的输出示例：

```
program vers proto  port service
    100021    1   udp  32774  nlockmgr
    100021    3   udp  32774  nlockmgr
    100021    4   udp  32774  nlockmgr
    100021    1   tcp  34437  nlockmgr
    100021    3   tcp  34437  nlockmgr
```

```
100021    4   tcp  34437  nlockmgr
100011    1   udp    819  rquotad
100011    2   udp    819  rquotad
100011    1   tcp    822  rquotad
100011    2   tcp    822  rquotad
100003    2   udp   2049  nfs
100003    3   udp   2049  nfs
100003    2   tcp   2049  nfs
100003    3   tcp   2049  nfs
100005    1   udp    836  mountd
100005    1   tcp    839  mountd
100005    2   udp    836  mountd
100005    2   tcp    839  mountd
100005    3   udp    836  mountd
100005    3   tcp    839  mountd
```

如果其中一个 NFS 服务没有正确启动，rpcbind 将不能将来自客户端的对该服务的 RPC 请求映射到正确的端口。在很多情况下，如果 rpcinfo 输出中不存在 NFS，重启 NFS 会导致服务使用 rpcbind 正确注册并开始工作。

有关 rpcinfo 的更多信息和选项列表，请查看其 man page。

8.9. PNFS

从 Red Hat Enterprise Linux 6.4 开始，支持 Parallel NFS (pNFS)作为 NFS v4.1 标准的一部分。pNFS 架构提高了 NFS 的可扩展性，并可能提高性能。也就是说，当服务器也实现 pNFS 时，客户端也可以同时通过多个服务器访问数据。它支持三种存储协议或布局：文件、对象和块。

注意

该协议允许三种可能的 pNFS 布局类型：file、object 和 blocks。虽然 Red Hat Enterprise Linux 6.4 客户端只支持文件布局类型，但 Red Hat Enterprise Linux 7 支持 files 布局类型，对象和块布局类型作为技术预览包含。

pNFS Flex 文件

灵活的文件是 pNFS 的新布局，它允许将独立 NFSv3 和 NFSv4 服务器的聚合到一个扩展命名空间中。Flex Files 功能是 NFSv4.2 标准的一部分，如 RFC 7862 规范中所述。

从 Red Hat Enterprise Linux 7.4 开始，Red Hat Enterprise Linux 可以从 Flex Files 服务器挂载 NFS 共享。

挂载 pNFS 共享

- 要启用 pNFS 功能，使用 NFS 版本 4.1 或更高版本从启用了 pNFS 的服务器挂载共享：

```
# mount -t nfs -o v4.1 server:/remote-export /local-directory
```

服务器启用 pNFS 后，在第一次挂载时会自动载入 nfs_layout_nfsv41_files 内核。输出中的挂载条目应包含 minorversion=1。使用以下命令验证模块是否已加载：

```
$ lsmod | grep nfs_layout_nfsv41_files
```

- 要从支持 Flex Files 的服务器中挂载带有 Flex Files 功能的 NFS 共享，请使用 NFS 版本 4.2 或更高版本：

```
# mount -t nfs -o v4.2 server:/remote-export /local-directory
```

验证 nfs_layout_flexfiles 模块是否已加载：

```
$ lsmod | grep nfs_layout_flexfiles
```

其它资源

有关 pNFS 的详情，请参考： http://www.pnfs.com

8.10. 在 NFS 中启用 PNFS SCSI 布局

您可以将 NFS 服务器和客户端配置为使用 pNFS SCSI 布局访问数据。

先决条件

- 客户端和服务器必须能够向同一个块设备发送 SCSI 命令。就是说块设备必须位于共享的 SCSI 总线中。
- 块设备必须包含 XFS 文件系统。
- SCSI 设备必须支持 SCSI Persistent Reservations，如 SCSI-3 Ppriary Commands 规格中所述。

8.10.1. pNFS SCSI 布局

SCSI 布局基于 pNFS 块布局的工作。布局在 SCSI 设备中定义。它包含一系列固定大小的块来作为逻辑单元(LU)，这些逻辑单元必须能够支持 SCSI 持久保留。LU 设备识别通过其 SCSI 设备识别。

在涉及长时间的单客户端访问文件的用例中，pNFS SCSI 表现良好。例如：邮件服务器或者虚拟机。

客户端和服务器之间的操作

当 NFS 客户端从文件读取或写入文件时，客户端会执行 LAYOUTGET 操作。服务器会使用文件在 SCSI 设备中的位置进行响应。客户端可能需要执行 GETDEVICEINFO 的额外操作，以确定要使用哪个 SCSI 设备。如果这些操作正常工作，客户端可以直接向 SCSI 设备发出 I/O 请求，而不是向服务器发送 READ 和 WRITE 操作。

客户端之间的错误或争用可能会导致服务器重新调用布局，或者不将它们发送给客户端。在这些情况下，客户端回退到向服务器发出 READ 和 WRITE 操作，而不是直接向 SCSI 设备发送 I/O 请求。

要监控操作，请参阅 第 8.10.6 节 “监控 pNFS SCSI 布局功能”。

设备保留

pNFS SCSI 通过分配保留来处理保护。在服务器向客户端发送布局之前，它会保留 SCSI 设备，以确保只有注册的客户端才可以访问该设备。如果客户端可以向那个 SCSI 设备发送命令，但没有在该设备上注册，那么该设备上的客户端的许多操作都会失败。例如，如果服务器没有向客户端提供该设备的布局，客户端上的 blkid 命令将无法显示 XFS 文件系统的 UUID。

服务器不会删除其自身的持久性保留。这样可在重启客户端和服务器后保护该设备中的文件系统中的数据。为了重新使用 SCSI 设备，您可能需要手动删除 NFS 服务器中的持久性保留。

8.10.2. 检查与 pNFS 兼容的 SCSI 设备

这个过程检查 SCSI 设备是否支持 pNFS SCSI 布局。

先决条件

- 安装 sg3_utils 软件包：

```
# yum install sg3_utils
```

过程 8.1. 检查与 NFS 兼容的 SCSI 设备

过程 8.4. 检查与 pNFS 兼容的 SCSI 设备

- 在服务器和客户端中检查正确的 SCSI 设备支持：

```
# sg_persist --in --report-capabilities --verbose path-to-scsi-device
```

确保设置了 Persist Through Power Los Active (PTPL_A)位。

例 8.8. 支持 pNFS SCSI 的 SCSI 设备

以下是支持 pNFS SCSI 的 SCSI 设备的 sg_persist 输出示例。PTPL_A 位报告 1。

```
  inquiry cdb: 12 00 00 00 24 00
  Persistent Reservation In cmd: 5e 02 00 00 00 00 00 20 00 00
LIO-ORG   block11          4.0
Peripheral device type: disk
Report capabilities response:
 Compatible Reservation Handling(CRH): 1
 Specify Initiator Ports Capable(SIP_C): 1
 All Target Ports Capable(ATP_C): 1
 Persist Through Power Loss Capable(PTPL_C): 1
 Type Mask Valid(TMV): 1
 Allow Commands: 1
 Persist Through Power Loss Active(PTPL_A): 1
  Support indicated in Type mask:
   Write Exclusive, all registrants: 1
   Exclusive Access, registrants only: 1
   Write Exclusive, registrants only: 1
   Exclusive Access: 1
   Write Exclusive: 1
   Exclusive Access, all registrants: 1
```

其它资源

- sg_persist(8) man page

8.10.3. 在服务器中设置 pNFS SCSI

这个过程将 NFS 服务器配置为导出 pNFS SCSI 布局。

过程 8.5. 在服务器中设置 pNFS SCSI

1. 在服务器中挂载在 SCSI 设备中创建的 XFS 文件系统。

2. 将 NFS 服务器配置为导出 NFS 版本 4.1 或更高版本。在 /etc/nfs.conf 文件的 [nfsd] 部分中设置以下选项：

```
[nfsd]
vers4.1=y
```

3. 将 NFS 服务器配置为通过 NFS 导出 XFS 文件系统，使用 pnfs 选项：

例 8.9. /etc/exports 中的条目导出 pNFS SCSI

/etc/exports 配置文件中的以下条目将挂载到 /exported/directory/ 的文件系统导出到 allowed.example.com 客户端，来作为 pNFS SCSI 布局：

```
/exported/directory allowed.example.com(pnfs)
```

其它资源

- 有关配置 NFS 服务器的详情请参考 第 8.6 节 “配置 NFS 服务器”。

8.10.4. 在客户端中设置 pNFS SCSI

这个过程将 NFS 客户端配置为挂载 pNFS SCSI 布局。

先决条件

- NFS 服务器被配置为通过 pNFS SCSI 导出 XFS 文件系统。请参阅 第 8.10.3 节 “在服务器中设置 pNFS SCSI”。

过程 8.6. 在客户端中设置 pNFS SCSI

- 在客户端中使用 NFS 版本 4.1 或更高版本挂载导出的 XFS 文件系统：

```
# mount -t nfs -o nfsvers=4.1 host:/remote/export /local/directory
```

不要在没有 NFS 的情况下直接挂载 XFS 文件系统。

其它资源

- 有关挂载 NFS 共享的详情请参考 第 8.2 节 “配置 NFS 客户端”。

8.10.5. 在服务器中释放 pNFS SCSI 保留

此流程释放 NFS 服务器在 SCSI 设备中拥有的持久保留。这可让您在不再需要导出 pNFS SCSI 时重新使用 SCSI 设备。

您必须从服务器中删除保留。它不能从不同的 IT Nexus 中删除。

先决条件

- 安装 sg3_utils 软件包：

```
# yum install sg3_utils
```

过程 8.7. 在服务器中释放 pNFS SCSI 保留

1. 在服务器上查询现有保留：

```
# sg_persist --read-reservation path-to-scsi-device
```

例 8.10. 在 /dev/sda 上查询保留

```
# sg_persist --read-reservation /dev/sda

  LIO-ORG   block_1          4.0
  Peripheral device type: disk
  PR generation=0x8, Reservation follows:
    Key=0x100000000000000
    scope: LU_SCOPE,  type: Exclusive Access, registrants only
```

2. 删除服务器上的现有注册：

```
# sg_persist --out \
```

```
        --release \
        --param-rk=reservation-key \
        --prout-type=6 \
        path-to-scsi-device
```

例 8.11. 删除 /dev/sda 上的保留

```
# sg_persist --out \
          --release \
          --param-rk=0x100000000000000 \
          --prout-type=6 \
          /dev/sda

  LIO-ORG   block_1          4.0
  Peripheral device type: disk
```

其它资源

- **sg_persist(8) man page**

8.10.6. 监控 pNFS SCSI 布局功能

您可以监控 pNFS 客户端和服务器是否交换正确的 pNFS SCSI 操作，或者它们是否回退到常规的 NFS 操作。

先决条件

- **配置了 pNFS SCSI 客户端和服务器。**

8.10.6.1. 从使用 nfsstat 的服务器检查 pNFS SCSI 操作

这个过程使用 nfsstat 工具来监控服务器的 pNFS SCSI 操作。

过程 8.8. 从使用 nfsstat 的服务器检查 pNFS SCSI 操作

1. **监控服务器中服务的操作：**

```
# watch --differences \
      "nfsstat --server | egrep --after-context=1 read\|write\|layout"

Every 2.0s: nfsstat --server | egrep --after-context=1 read\|write\|layout

putrootfh    read         readdir      readlink     remove  rename
```

```
2        0% 0        0% 1        0% 0        0% 0        0% 0        0%
--
setcltidconf verify   write       rellockowner bc_ctl  bind_conn
0        0% 0        0% 0        0% 0        0% 0        0% 0        0%
--
getdevlist   layoutcommit layoutget    layoutreturn secinfononam sequence
0        0% 29       1% 49       1% 5        0% 0        0% 2435     86%
```

2. 客户端和服务器在以下情况下使用 pNFS SCSI 操作：

- layoutget、layoutreturn 和 layoutcommit 计数器递增。这意味着服务器提供布局。
- 服务器 读写 计数器不会递增。这意味着客户端正在直接向 SCSI 设备执行 I/O 请求。

8.10.6.2. 使用 mountstats 从客户端检查 pNFS SCSI 操作

这个过程使用 /proc/self/mountstats 文件来监控客户端的 pNFS SCSI 操作。

过程 8.9. 使用 mountstats 从客户端检查 pNFS SCSI 操作

1. 列出每个挂载的操作计数器：

```
# cat /proc/self/mountstats \
    | awk /scsi_lun_0/,/^$/ \
    | egrep device\|READ\|WRITE\|LAYOUT

device 192.168.122.73:/exports/scsi_lun_0 mounted on /mnt/rhel7/scsi_lun_0 with fstype
nfs4 statvers=1.1
   nfsv4:
bm0=0xfdffbfff,bm1=0x40f9be3e,bm2=0x803,acl=0x3,sessions,pnfs=LAYOUT_SCSI
          READ: 0 0 0 0 0 0 0 0
         WRITE: 0 0 0 0 0 0 0 0
     READLINK: 0 0 0 0 0 0 0 0
      READDIR: 0 0 0 0 0 0 0 0
    LAYOUTGET: 49 49 0 11172 9604 2 19448 19454
  LAYOUTCOMMIT: 28 28 0 7776 4808 0 24719 24722
  LAYOUTRETURN: 0 0 0 0 0 0 0 0
   LAYOUTSTATS: 0 0 0 0 0 0 0 0
```

2. 在结果中：

- LAYOUT 统计指示客户端和服务器使用 pNFS SCSI 操作的请求。

- READ 和 WRITE 统计指示客户端和服务器回退到 NFS 操作的请求。

8.11. NFS 参考

管理 NFS 服务器可能是一个挑战。许多选项（包括本章中未提及的一些选项）可用于导出或挂载 NFS 共享。如需更多信息，请参阅以下源：

安装的文档

- man mount - 包含了全面查看 NFS 服务器和客户端配置的挂载选项。

- man fstab - 提供在启动时挂载文件系统的 /etc/fstab 文件格式的详细信息。

- man nfs - 提供特定于 NFS 文件系统导出和挂载选项的详情。

- man exports - 显示导出 NFS 文件系统时 /etc/exports 文件中使用的常见选项。

有用的网站

- http://linux-nfs.org - 开发人员查看项目状态更新的的当前站点。

- http://nfs.sourceforge.net/ - 开发人员的老家，里面仍然包含了很多有用的信息。

- http://www.citi.umich.edu/projects/nfsv4/linux/ - Linux 2.6 内核资源的 NFSv4。

- http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.111.4086 - 关于 NFS 版本 4 协议的功能和加强的完美白皮书。

相关的图书

- Hal Stern、Mike Eisler 和 Ricardo labiaga 管理 NFS 和 NIS ；O'Reilly & Associates - 为

许多不同的 NFS 导出和挂载选项提供了很好的参考指南。

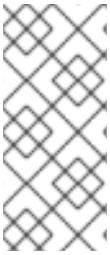
- *Brent Callaghan 的 NFS Illustrated; Addison-Wesley 出版公司 - 提供了 NFS 与其他网络文件系统的比较，并详细介绍了 NFS 通信是如何发生的。*

---

[1] rpcbind 服务取代了 portmap，这在以前的 Red Hat Enterprise Linux 版本中用于将 RPC 程序号映射到 IP 地址端口号组合。如需更多信息，请参阅 第 8.1.1 节 “所需的服务”。

第 9 章 服务器消息块(SMB)

服务器消息块(SMB)协议实现用于访问服务器上资源的应用层网络协议，如文件共享和共享打印机。在 Microsoft Windows 上，默认实施 SMB 。如果您运行 Red Hat Enterprise Linux，请使用 Samba 提供 SMB 共享，以及 cifs-utils 工具从远程服务器挂载 SMB 共享。



注意

在 SMB 的上下文中，您有时会阅读有关通用互联网文件系统(CIFS)协议的信息，这是 SMB 的一种问题。SMB 和 CIFS 协议都支持，并且挂载 SMB 和 CIFS 共享时涉及的内核模块和工具都使用名称 cifs。

9.1. 提供 SMB 共享

请参阅 Red Hat 系统管理员指南中的 Samba 部分。

9.2. 挂载 SMB 共享

在 Red Hat Enterprise Linux 上，内核的 cifs.ko 文件系统模块提供对 SMB 协议的支持。但是，要挂载并使用 SMB 共享，还必须安装 cifs-utils 软件包：

```
# yum install cifs-utils
```

cifs-utils 软件包提供以下工具：

- 挂载 SMB 和 CIFS 共享
- 在内核的密钥环中管理 NT Lan Manager (NTLM)凭据
- 在 SMB 和 CIFS 共享上的安全描述符中设置和显示访问控制列表(ACL)

9.2.1. 支持的 SMB 协议版本

cifs.ko 内核模块支持以下 SMB 协议版本：

- *SMB 1*
- *SMB 2.0*
- *SMB 2.1*
- *SMB 3.0*

注意

根据协议版本，并非所有 SMB 功能都已实施。

9.2.1.1. UNIX 扩展支持

Samba 在 SMB 协议中使用 CAP_UNIX 功能位来提供 UNIX 扩展功能。cifs.ko 内核模块也支持这些扩展。但是，Samba 和内核模块仅支持 SMB 1 协议中的 UNIX 扩展。

要使用 UNIX 扩展：

1. *将 /etc/samba/smb.conf 文件中的 [global] 部分中的 server min protocol 选项设置为 NT1。这是 Samba 服务器中的默认设置。*

2. *通过向 mount 命令提供 -o vers=1.0 选项，使用 SMB 1 协议 挂载共享。例如：*

```
mount -t cifs -o vers=1.0,username=user_name //server_name/share_name /mnt/
```

默认情况下，内核模块使用 SMB 2 或服务器支持的最高协议版本。将 -o vers=1.0 选项传给 mount 命令会强制内核模块使用 SMB 1 协议，该协议在使用 UNIX 扩展时是必需的。

要验证是否启用了 UNIX 扩展，请显示挂载共享的选项：

```
# mount
...
//server/share on /mnt type cifs (...,unix,...)
```

如果在挂载选项列表中显示了 unix 条目，则启用了 UNIX 扩展。

9.2.2. 手动挂载 SMB 共享

要手动挂载 SMB 共享，在 -t cifs 参数中使用 mount 工具：

```
# mount -t cifs -o username=user_name //server_name/share_name /mnt/
Password for user_name@//server_name/share_name:  ********
```

在 -o options 参数中，您可以指定用于挂载共享的选项。详情请查看 第 9.2.6 节 “常用挂载选项” man page 中的 mount.cifs(8) 和 OPTIONS 部分。

例 9.1. 使用加密的 SMB 3.0 连接挂载共享

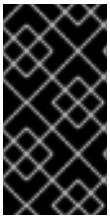
要将 \\服务器\示例\ 共享作为 DOMAIN\Administrator 用户通过加密的 SMB 3.0 连接挂载到 /mnt/ 目录：

```
# mount -t cifs -o username=DOMAIN\Administrator,seal,vers=3.0 //server/example /mnt/
Password for user_name@//server_name/share_name:  ********
```

9.2.3. 在系统引导时自动挂载 SMB 共享

要在系统引导时自动挂载 SMB 共享，请将共享条目添加到 /etc/fstab 文件中。例如：

```
//server_name/share_name  /mnt  cifs  credentials=/root/smb.cred  0 0
```



重要

要让系统自动挂载共享，您必须将用户名、密码和域名存储在凭据文件中。详情请查看 第 9.2.4 节 “使用凭据文件对 SMB 共享进行身份验证”。

在 /etc/fstab 文件的第四个字段中，指定挂载选项，如凭据文件的路径。详情请查看 第 9.2.6 节 “常用挂载选项” man page 中的 mount.cifs(8) 和 OPTIONS 部分。

要验证共享挂载是否成功，请输入：

```
# mount /mnt/
```

9.2.4. 使用凭据文件对 SMB 共享进行身份验证

在某些情况下，管理员希望在不输入用户名和密码的情况下挂载共享。要实施此操作，请创建一个凭据文件。例如：

过程 9.1. 创建凭证文件

1. 创建一个文件，如 ~/smb.cred，并指定该文件的用户名、密码和域名：

```
username=user_name
password=password
domain=domain_name
```

2. 将权限设置为只允许所有者可以访问该文件：

```
# chown user_name ~/smb.cred
# chmod 600 ~/smb.cred
```

现在，您可以将 credentials=file_name 挂载选项传给 mount 工具，或者在 /etc/fstab 文件中使用它来挂载共享，而无需提示输入用户名和密码。

9.2.5. 执行多用户 SMB 挂载

您为挂载共享提供的凭据默认确定对挂载点的访问权限。例如，如果您在挂载共享时使用 DOMAIN\example 用户，则共享上的所有操作都将以该用户的身份执行，无论哪个本地用户执行该操作。

然而，在某些情况下，管理员希望在系统启动时自动挂载共享，但用户应使用自己的凭据对共享的内容执行操作。multiuser 挂载选项允许您配置此场景。

重要

要使用 multiuser，还必须将 sec=security_type 挂载选项设置为支持以非交互方式提供凭据的安全类型，如 krb5 或带有凭据文件的 ntlmssp 选项。请参阅 “以用户身份访问共享”一节。

root 用户使用 multiuser 选项以及对共享内容具有最少访问权限的帐户挂载共享。然后，常规用户可以使用 cifscreds 实用程序将其用户名和密码提供给当前会话的内核密钥环。如果用户访问挂载的共享的内容，则内核将使用内核密钥环中的凭据，而不是最初用来挂载共享的凭据。

使用 multiuser 选项挂载共享

在系统引导时，使用 multiuser 选项自动挂载共享：

过程 9.2. 使用 multiuser 选项创建 /etc/fstab 文件条目

1. 在 /etc/fstab 文件中为共享创建条目。例如：

```
//server_name/share_name  /mnt  cifs  multiuser,sec=ntlmssp,credentials=/root/smb.cred  0 0
```

2. 挂载共享：

```
# mount /mnt/
```

如果您不想在系统引导时自动挂载共享，请通过将 -o multiuser,sec=security_type 传递给 mount 命令手动挂载它。有关手动挂载 SMB 共享的详情，请参考 第 9.2.2 节 “手动挂载 SMB 共享”。

验证 SMB 共享是否使用 multiuser 选项挂载

使用 multiuser 选项验证共享是否挂载：

```
# mount
...
//server_name/share_name on /mnt type cifs (sec=ntlmssp,multiuser,...)
```

以用户身份访问共享

如果使用 multiuser 选项挂载 SMB 共享，用户可以向内核的密钥环提供其服务器凭证：

```
# cifscreds add -u SMB_user_name server_name
Password: ********
```

现在，当用户在包含挂载的 SMB 共享的目录中执行操作时，服务器会为此用户应用文件系统权限，而不是挂载共享时最初使用的权限。

注意

多个用户可以同时对挂载的共享使用自己的凭据来执行操作。

9.2.6. 常用挂载选项

当您挂载 SMB 共享时，挂载选项将决定：

- 如何与服务器建立连接。例如：连接到服务器时使用 SMB 协议版本。
- 如何将共享挂载到本地文件系统.例如，如果系统覆盖了远程文件和目录的权限，使多个本地用户能够访问服务器上的内容。

要在 /etc/fstab 文件的第四个字段或 挂载 命令的 -o 参数中设置多个选项，请使用逗号分隔它们。例如，请参阅 过程 9.2, “使用 multiuser 选项创建 /etc/fstab 文件条目”。

以下列表提供了常用挂载选项的概述：

表 9.1. 常用挂载选项

选项	描述
credentials=file_name	设置凭证文件的路径。请参阅 第 9.2.4 节 “使用凭据文件对 SMB 共享进行身份验证”。
dir_mode=mode	如果服务器不支持 CIFS UNIX 扩展，则设置目录模式。
file_mode=mode	如果服务器不支持 CIFS UNIX 扩展，则设置文件模式。

选项	描述
password=password	设置在 SMB 服务器中验证的密码。另外，也可使用 credentials 选项指定凭据文件。
seal	使用 SMB 3.0 或更高的协议版本启用对连接的加密支持。因此，使用 seal 和 vers 挂载选项来设置成 3.0 或更高版本。请参阅 例 9.1 “使用加密的 SMB 3.0 连接挂载共享”。
sec=security_mode	设置安全模式，如 ntlmsspi，来启用 NTLMv2 密码哈希和已启用的数据包签名。有关支持的值列表，请查看 mount.cifs(8) man page 中的选项描述。 如果服务器不支持 ntlmv2 安全模式，则使用 sec=ntlmssp，这是默认值。出于安全考虑，请不要使用不安全的 ntlm 安全模式。
username=user_name	设置在 SMB 服务器中验证的用户名。另外，也可使用 credentials 选项指定凭据文件。
vers=SMB_protocol_version	设定用于与服务器通信的 SMB 协议版本。

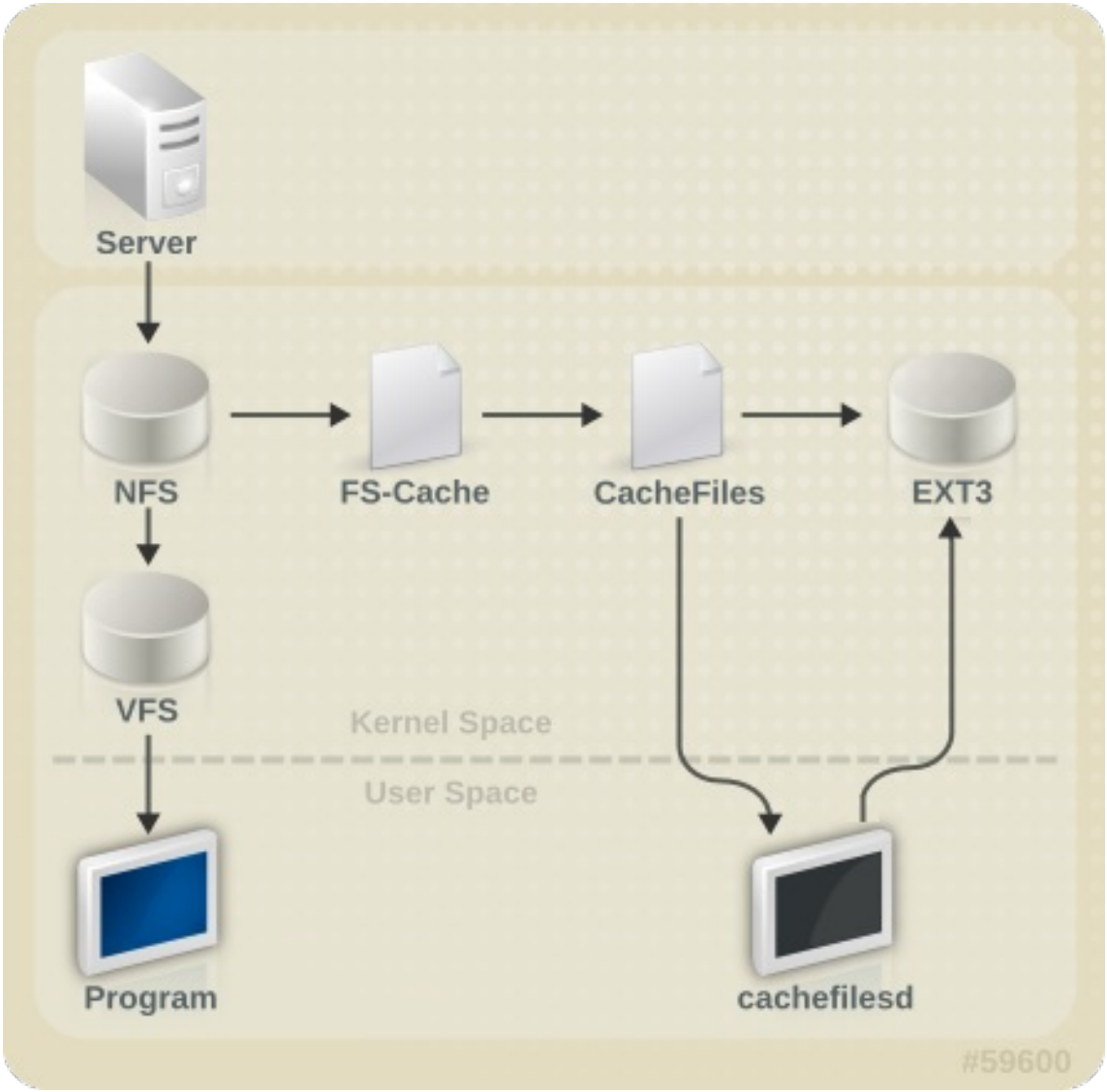
有关完整列表，请查看 mount.cifs(8) man page 中的 OPTIONS 部分。

第 10 章 FS-CACHE

FS-Cache 是一种永久的本地缓存，可供文件系统用于通过网络检索数据并将其缓存在本地磁盘上。这有助于最小化网络流量，以便用户从通过网络（例如 NFS）挂载的文件系统来访问数据。

下图显示了 FS-Cache 的工作原理：

图 10.1. FS-Cache 概述



[D]

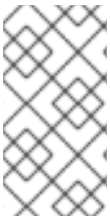
FS-Cache 旨在对系统的用户和管理员尽可能透明。与 Solaris 上的 cachefs 不同，FS-Cache 允许服务器上的文件系统直接与客户端的本地缓存交互，而无需创建过度挂载的文件系统。使用 NFS 时，挂载选项指示客户端挂载启用了 FS-cache 的 NFS 共享。

FS-Cache 不会改变通过网络进行的文件系统的基本操作 - 它只是为文件系统提供一个永久的、可以缓存数据的地方。例如，客户端仍然可以挂载 NFS 共享，无论是否启用了 FS-Cache。此外，缓存的 NFS

可以处理不适合缓存的文件（无论是单独的还是全部的），因为文件可以部分缓存，且无需事先完全读取。FS-Cache 还会隐藏发生在客户端文件系统驱动程序的缓存中的所有 I/O 错误。

要提供缓存服务，FS-Cache 需要 缓存后端。缓存后端是配置为提供缓存服务（例如 cachefiles）的存储驱动程序。在这种情况下，FS-Cache 需要一个挂载的基于块的文件系统，它支持 bmap 和扩展属性（如 ext3）作为其缓存后端。

FS-Cache 无法任意缓存任何文件系统，无论是通过网络还是通过其他方式：必须更改共享文件系统的驱动程序，以允许与 FS-Cache、数据存储/检索以及元数据设置和验证进行交互。FS-Cache 需要来自缓存文件系统的 索引密钥 和 一致性数据 来支持持久性：索引密钥将文件系统对象与缓存对象匹配，一致性数据确定缓存对象是否仍然有效。



注意

在 Red Hat Enterprise Linux 7 中，默认情况下不安装 cachefilesd 软件包，需要手动安装。

10.1. 性能保证

FS-Cache 不 保证性能的提高，但它通过避免网络拥塞来确保一致的性能。使用缓存后端会导致性能下降：例如，缓存的 NFS 共享会为跨网络查找添加磁盘访问。虽然 FS-Cache 尝试尽可能异步，但有一些同步路径（例如读取）是不可能的。

例如，使用 FS-Cache 通过其它未装载的 GigE 网络在两台计算机之间缓存 NFS 共享，将不会在文件访问方面显示出任何性能改进。相反，通过服务器内存而不是本地磁盘可以更快地满足 NFS 请求。

因此，使用 FS-Cache 是各种因素之间的 折衷。例如，如果使用 FS-Cache 来缓存 NFS 流量，它可能会减慢一点儿客户端的速度，但通过满足本地的读请求而无需消耗网络带宽，可以大量减少网络和服务器加载。

10.2. 设置缓存

目前，Red Hat Enterprise Linux 7 只提供 cachefiles 缓存后端。cachefilesd 守护进程启动并管理 cachefiles。/etc/cachefilesd.conf 文件控制 cachefile 如何提供缓存服务。

在缓存后端中配置的第一个设置是要将哪个目录用作缓存。要配置它，请使用以下参数：

```
$ dir /path/to/cache
```

通常，缓存后端目录在 /etc/cachefilesd.conf 中被设置为 /var/cache/fscache，如下所示：

```
$ dir /var/cache/fscache
```

如果要更改缓存后端目录，selinux 上下文必须与 /var/cache/fscache 相同：

```
# semanage fcontext -a -e /var/cache/fscache /path/to/cache
# restorecon -Rv /path/to/cache
```

在设置缓存时，将 /path/to/cache 替换为目录名称。

注意

如果给定的设置 selinux 上下文的命令无法工作，请使用以下命令：

```
# semanage permissive -a cachefilesd_t
# semanage permissive -a cachefiles_kernel_t
```

FS-Cache 会将缓存存储在托管 /path/to/cache 的文件系统中。在笔记本电脑上，建议使用 root 文件系统(/)作为主机文件系统，但对于桌面计算机而言，更谨慎地挂载专门用于缓存的磁盘分区。

支持 FS-Cache 缓存后端所需的功能的文件系统包括以下文件系统的 Red Hat Enterprise Linux 7 实现：

- ext3（启用了扩展属性）
- ext4
- Btrfs
- XFS

主机文件系统必须支持用户定义的扩展属性；FS-Cache 使用这些属性来存储一致的维护信息。要为

ext3 文件系统（例如 设备）启用用户定义的扩展属性，请使用：

```
# tune2fs -o user_xattr /dev/device
```

另外，也可在挂载时启用文件系统的扩展属性，如下所示：

```
# mount /dev/device /path/to/cache -o user_xattr
```

缓存后端的工作原理是在托管缓存的分区上维护一定数量的空闲空间。当系统的其他元素耗尽空闲空间时，它会增长和收缩缓存，使得可以在根文件系统（例如，在笔记本电脑上）上安全地使用。FS-Cache 对此行为设置默认值，可以通过 缓存剔除限制 进行配置。有关配置缓存剔除限制的详情，请参考 第 10.4 节 “设置缓存剔除限制”。

配置文件就位后，启动 cachefilesd 服务：

```
# systemctl start cachefilesd
```

要将 cachefilesd 配置为在引导时启动，请以 root 用户身份执行以下命令：

```
# systemctl enable cachefilesd
```

10.3. 在 NFS 中使用缓存

除非明确指示，否则 NFS 将不会使用缓存。要将 NFS 挂载配置为使用 FS-Cache，请在 mount 命令中包含 -o fsc 选项：

```
# mount nfs-share:/ /mount/point -o fsc
```

对 /mount/point 下文件的所有访问都将通过缓存，除非打开该文件以进行直接 I/O 或写入。如需更多信息，请参阅 第 10.3.2 节 “使用 NFS 的缓存限制”。NFS 使用 NFS 文件句柄 而不是 文件名来索引缓存内容，这意味着硬链接的文件可以正确共享缓存。

NFS 版本2、3 和 4 支持缓存。但是，每个版本使用不同的分支进行缓存。

10.3.1. 缓存共享

与 NFS 缓存共享相关的一些潜在问题。因为缓存是持久的，所以缓存中的数据块会根据由四个键组成的序列来索引的：

- 第 1 级：服务器详情
- 第 2 级：一些挂载选项；安全类型；FSID；uniquifier
- 第 3 级：文件处理
- 第 4 级：文件中的页号

为避免超级块之间一致性管理问题，希望缓存数据的所有 NFS 超级块都具有唯一的第 2 级密钥。通常，两个 NFS 挂载使用相同的源卷和选项共享超级块，因此共享缓存，即使它们在该卷中挂载不同的目录。

> 例 10.1. 缓存共享
>
> 使用以下两个 mount 命令：
>
> ```
> mount home0:/disk0/fred /home/fred -o fsc
>
> mount home0:/disk0/jim /home/jim -o fsc
> ```
>
> 在这里，/home/fred 和 /home/jim 可能会共享超级块，因为它们具有相同的选项，特别是当它们来自 NFS 服务器(home0)上的相同卷/分区时。现在，考虑接下来的两个挂载命令：
>
> ```
> mount home0:/disk0/fred /home/fred -o fsc,rsize=230
>
> mount home0:/disk0/jim /home/jim -o fsc,rsize=231
> ```
>
> 在这种情况下，/home/fred 和 /home/jim 不会共享超级块，因为它们具有不同的网络访问参数，这些参数是第 2 级密钥的一部分。以下的挂载序列也是如此：

mount home0:/disk0/fred /home/fred1 -o fsc,rsize=230

mount home0:/disk0/fred /home/fred2 -o fsc,rsize=231

此处，两个子树的内容(/home/fred1 和 /home/fred2)将缓存 两次。

避免超级块共享的另一种方法是使用 nosharecache 参数显式阻止它。使用相同的示例：

mount home0:/disk0/fred /home/fred -o nosharecache,fsc

mount home0:/disk0/jim /home/jim -o nosharecache,fsc

然而，在这种情况下，只允许其中一个超级块使用缓存，因为无法区分 home0:/disk0/fred 和 home0:/disk0/jim 的第 2 级密钥。要解决这个问题，请在至少一个挂载上添加一个 唯一标识符，即 fsc=unique-identifier。例如：

mount home0:/disk0/fred /home/fred -o nosharecache,fsc

mount home0:/disk0/jim /home/jim -o nosharecache,fsc=jim

在这里，唯一标识符 jim 添加到 /home/jim 缓存中使用的第 2 级键中。

10.3.2. 使用 NFS 的缓存限制

- 为直接 I/O 打开共享文件系统的文件将自动绕过缓存。这是因为这种访问类型必须与服务器直接进行。

- 打开共享文件系统的文件进行写入不适用于 NFS 版本 2 和 3。这些版本的协议无法为客户端提供足够的一致性管理信息，来检测另一客户端对同一文件的并发写入。

-

从共享文件系统打开一个文件直接 I/O 或写入清除文件缓存的副本。FS-Cache 不会再次缓存文件，直到它不再为直接 I/O 或写操作而打开。

- 另外，FS-Cache 的这个发行版本只缓存常规 NFS 文件。FS-Cache 不会 缓存目录、符号链接、设备文件、FIFO 和套接字。

10.4. 设置缓存剔除限制

cachefilesd 守护进程的工作原理是：缓存来自共享文件系统的远程数据，以释放磁盘上的空间。这可能会消耗掉所有空闲空间，如果磁盘还存放 root 分区，这可能会很糟糕。为了控制这一点，cachefiled 会尝试通过丢弃缓存中的旧对象（例如，最近未访问的）来维护一定数量的可用空间。这个行为被称为 缓存剔除。

缓存剔除是根据块的百分比以及底层文件系统中可用文件的百分比来完成的。/etc/cachefilesd.conf 中的设置控制六个限制：

brun N%（块百分比），frun N%（文件百分比）

如果可用空间量和缓存中的可用文件数超过这两个限制，则剔除将关闭。

bcull N%（块百分比），fcull N%（文件百分比）

如果可用空间量或缓存中的文件数量低于其中任何一个限制，则启动剔除。

bstop N%（块百分比），fstop N%（文件百分比）

如果缓存中可用空间的数量或可用文件的数量低于其中任何一个限制，则不允许进一步分配磁盘空间或文件，直到筛选再次引发超过这些限制的情况。

每个设置 N 的默认值如下：

- brun/frun - 10%
- bcull/fcull - 7%

- *bstop/fstop - 3%*

在配置这些设置时，必须满足以下条件：

- *0 PROFILE bstop < bcull < brun < 100*
- *0 PROFILE fstop < fcull < frun < 100*

这些是可用空间和可用文件的百分比，不显示为 100 减去 df 程序所显示的百分比。

重要

剔除同时依赖于 bxxx 和 fxxx 对；它们不能被单独处理。

10.5. 统计信息

FS-Cache 还跟踪一般统计信息。要查看此信息，请使用：

```
# cat /proc/fs/fscache/stats
```

FS-Cache 统计数据包括有关决策点和对象计数器的信息。如需更多信息，请参阅以下内核文档：

/usr/share/doc/kernel-doc-version/Documentation/filesystems/caching/fscache.txt

10.6. FS-CACHE 参考

有关 cachefilesd 以及如何配置它的更多信息，请参阅 man cachefilesd 和 man cachefilesd.conf。以下内核文档还提供附加信息：

- */usr/share/doc/cachefilesd-version-number/README*
- */usr/share/man/man5/cachefilesd.conf.5.gz*

- */usr/share/man/man8/cachefilesd.8.gz*

有关 FS-Cache 的常规信息，包括其设计约束、可用统计和功能的详情，请查看以下内核文档：
/usr/share/doc/kernel-doc-version/Documentation/filesystems/caching/fscache.txt

部分 II. 存储管理

存储管理部分从 Red Hat Enterprise Linux 7 的存储注意事项开始。有关分区、逻辑卷管理和交换分区的说明如下。接下来是磁盘配额、RAID 系统，然后是 mount 命令、volume_key 和 acls 的功能。SSD 调优、写屏障、I/O 限制和无盘系统遵循这一步。接下来是在线存储的大篇章，最后是需要完成的设备映射器多路径和虚拟存储。

第 11 章 安装过程中的存储注意事项

许多存储设备和文件系统设置只能在安装时配置。其他设置（如文件系统类型）只能修改到某个点，而无需重新格式化。因此，在安装 Red Hat Enterprise Linux 7 前，建议您相应地规划存储配置。

本章讨论为您的系统规划存储配置时需要考虑的事项。有关安装说明（包括安装期间的存储配置），请参阅红帽提供的 安装指南。

有关红帽官方支持大小和存储限制的详情，请参考文章 http://www.redhat.com/resourcelibrary/articles/articles-red-hat-enterprise-linux-6-technology-capabilities-and-limits。

11.1. 特殊注意事项

本节列举了对于特定存储配置需要考虑的几个问题和因素。

/home、/opt、/usr/local 的单独分区

如果您将来可能会升级您的系统，请将 /home、/opt 和 /usr/local 放在单独的设备中。这可让您重新格式化包含操作系统的设备或文件系统，同时保留您的用户和应用程序数据。

IBM System Z 上的 DASD 和 zFCP 设备

在 IBM System Z 平台上，DASD 和 zFCP 设备通过 通道命令字 (CCW)机制进行配置。CCW 路径必须明确添加到系统，然后将其上线。对于 DASD 设备，这意味着在引导命令行或 CMS 配置文件中将设备号（或设备号范围）列为 DASD= 参数。

对于 zFCP 设备，您必须列出设备号、逻辑单元号 (LUN)和 全局端口名称 (WWPN)。初始化 zFCP 设备后，它将映射到 CCW 路径。引导命令行（或 CMS 配置文件）中的 FCP_x= 行允许您为安装程序指定此信息。

使用 LUKS 加密块设备

使用 LUKS/dm-crypt 格式化块设备以进行加密，会破坏该设备上的任何现有格式。因此，在将新系统的存储配置作为安装过程的一部分激活之前，您应该决定对哪些设备进行加密(如果有的话)。

过时的 BIOS RAID 元数据

从为固件 RAID 配置的系统中移动磁盘，而不 从磁盘中删除 RAID 元数据，可能会阻止 Anaconda 正确检测磁盘。

警告

从磁盘中移除/删除 RAID 元数据可能会破坏任何存储的数据。红帽建议您在继续前备份数据。

注意

如果您使用 dmraid 创建 RAID 卷（现已弃用），请使用 dmraid 工具删除它：

```
# dmraid -r -E /device/
```

有关管理 RAID 设备的详情，请参考 man dmraid 和 第 18 章 独立磁盘冗余阵列(RAID)。

iSCSI 检测和配置

对于 iSCSI 驱动器的即插即用检测，请在支持启动的 网络接口卡 (NIC)的固件中配置它们。安装过程中支持 iSCSI 目标的 CHAP 验证。但是，在安装过程中不支持 iSNS 发现。

FCoE 检测和配置

对于 以太网光纤通道 (FCoE)驱动器的插件和播放检测，请在支持 EDD 引导的 NIC 的固件中配置它们。

DASD

无法在安装过程中添加或配置 直接访问的存储设备 (DASD)。此类设备在 CMS 配置文件中指定。

启用了 DIF/DIX 的块设备

DIF/DIX 是某些 SCSI 主机总线适配器和块设备提供的硬件校验和功能。启用 DIF/DIX 时，如果块设备用作通用块设备，则会出现错误。缓冲的 I/O 或 mmap (2)将无法可靠地工作，因为缓冲的写入路径中没

有交集，以防止在计算 DIF/DIX 校验和后缓冲的数据被覆盖。

这会导致 I/O 稍后失败，并显示 checksum 错误。这个问题对于所有块设备（或基于文件系统）缓冲的 I/O 或 mmap (2) I/O 来说很常见，因此无法解决由于覆盖导致的这些错误。

因此，启用了 DIF/DIX 的块设备应仅与使用 O_DIRECT 的应用一起使用。此类应用应使用裸块设备。或者，只要通过文件系统发布 O_DIRECT I/O，在启用了 DIF/DIX 的块设备上使用 XFS 文件系统也很安全。XFS 是唯一在执行某些分配操作时不会回退到缓冲的 I/O 的文件系统。

确保在计算 DIF/DIX 校验和后 I/O 数据不会发生变化，因此只有设计用于 O_DIRECT I/O 和 DIF/DIX 硬件的应用程序应使用 DIF/DIX。

第 12 章 文件系统检查

可以使用特定于文件系统的用户空间工具检查文件系统的一致性，并选择性地进行修复。这些工具通常被称为 fsck 工具，其中 fsck 是 文件系统检查 的简化版本。

注意

这些文件系统检查程序只保证跨文件系统的元数据的一致性；它们不知道文件系统中包含的实际数据，且不是数据恢复工具。

发生文件系统不一致的原因可能有多种，包括但不限于硬件错误、存储管理错误和软件 bug 。

在现代元数据日志文件系统变得常见之前，每当系统崩溃或断电时，都需要进行文件系统检查。这是因为文件系统更新可能已被中断，从而导致状态不一致。因此，通常会在启动时在 /etc/fstab 中列出的每个文件系统上运行文件系统检查。对于日志记录文件系统，这通常是一个非常短的操作，因为即使崩溃，文件系统的元数据日志也会确保一致性。

但是，有时可能会出现文件系统不一致或损坏，即使日志记录文件系统也是如此。发生这种情况时，必须使用文件系统检查程序来修复文件系统。以下提供了执行此流程时的最佳实践和其他有用信息。

重要

除非机器没有引导，否则红帽不推荐这样做，否则文件系统非常大，或者文件系统位于远程存储中。通过将 /etc/fstab 中的第六个字段设置为 0，可以在引导时禁用文件系统检查。

12.1. FSCK 的最佳实践

通常，运行文件系统检查和修复工具至少可以自动修复发现的一些不一致问题。在某些情况下，如果无法修复严重损坏的 inode 或目录，可能会将它们丢弃。可能会对文件系统进行大量更改。要确保意外或不必要的更改不会永久发生，请执行以下预防步骤：

空运行

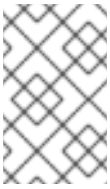
大多数文件系统检查程序都有一个操作模式，用于检查但不修复文件系统。在这个模式中，检查程序会输出它发现的任何错误以及它所执行的操作，而无需实际修改文件系统。

注意

稍后一致性检查阶段可能会打印额外的错误，因为在修复模式下运行时，它会发现可能在早期阶段已经修复了的不一致问题。

首先对文件系统镜像进行操作

大多数文件系统都支持创建 元数据镜像，这是仅包含元数据的文件系统的稀疏副本。由于文件系统检查程序仅在元数据上运行，因此此类镜像可用于对实际文件系统修复执行空运行，以评估实际要进行的更改。如果更改可以接受，则可以对文件系统本身进行修复。



注意

严重损坏的文件系统可能会导致元数据镜像创建出现问题。

保存文件系统镜像以进行支持调查

如果因为软件错误导致损坏，则预修复文件系统元数据镜像通常有助于支持调查。修复前镜像中存在的损坏模式可能有助于根本原因的分析。

仅在卸载的文件系统上运行

文件系统修复必须只在卸载的文件系统上运行。该工具必须具有对文件系统的唯一访问权限，或者可能导致进一步的损坏。大多数文件系统工具在修复模式下强制实施这个要求，虽然有些文件系统只支持在挂载的文件系统中只检查模式。如果在挂载的文件系统上运行只检查模式，则可能发现在卸载的文件系统上运行时未找到的错误。

磁盘错误

文件系统检查工具不能修复硬件问题。如果修复操作成功，文件系统必须是完全可读写的。如果文件系统因为硬件错误而损坏，则必须首先将文件系统移至良好磁盘，例如使用 dd (8) 工具。

12.2. FSCK 的文件系统特定信息

12.2.1. ext2、ext3 和 ext4

所有这些文件 sytems 使用 e2fsck 二进制文件来执行文件系统检查和修复。文件名 fsck.ext2、fsck.ext3 和 fsck.ext4 是同一二进制文件的硬链接。这些二进制文件在引导时自动运行，其行为因正在检查的文件系统和文件系统的状态而异。

对于不是元数据日志记录文件系统的 ext2 和没有日志的 ext4 文件系统，会调用完整的文件系统检查和修复。

对于带有元数据日志的 ext3 和 ext4 文件系统，日志会在用户空间中重新执行，二进制文件会退出。这是默认操作，因为日志重播可确保崩溃后文件系统的一致性。

如果这些文件系统在挂载时遇到元数据不一致的情况，它们会在文件系统超级块中记录此事实。如果 e2fsck 发现文件系统标记为此类错误，e2fsck 会在重播日志（如果存在）后执行完整检查。

如果未指定 -p 选项，e2fsck 可能会在运行期间要求用户输入。p 选项告知 e2fsck 自动执行所有可以安全完成的修复。如果需要用户干预，e2fsck 会在其输出中指示未修复的问题，并在退出代码中反映此状态。

常用的 e2fsck 运行时选项包括：

-n

无修改模式.仅检查操作.

-b superblock

如果主块被损坏，请指定备用的 suprerblock 的块数。

-f

如果超级块没有记录的错误，也强制进行全面检查。

-j journal-dev

指定外部日志设备（若有的话）。

-p

自动修复或"复制"文件系统，无需用户输入。

-y

假设所有问题的答案都为"是"。

e2fsck 的所有选项都在 e2fsck (8) 手册页中指定。

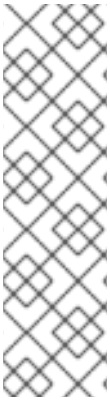
e2fsck 在运行时执行以下五个基本阶段：

1. Inode、块和大小检查.
2. 目录结构检查。
3. 目录连接性检查。
4. 参考计数检查。
5. 组摘要信息检查。

e2image (8) 工具可用于在修复前创建元数据镜像，以进行诊断或测试。-r 选项用于测试目的，以便创建与文件系统本身大小相同的稀疏文件。然后 e2fsck 可以直接对生成的文件进行操作。如果要存档或提供用于诊断的镜像，则应指定 -Q 选项。这会创建更精简的文件格式，适合传输。

12.2.2. XFS

在引导时不自动执行任何修复。要启动文件系统检查或修复，请使用 xfs_repair 工具。



注意

虽然 xfsprogs 软件包中存在 fsck.xfs 二进制文件，但这仅用于满足在启动时查找 fsck 的 initscripts。文件系统 二进制文件。fsck.xfs 立即退出，退出代码为 0。

旧的 xfsprogs 软件包包含一个 xfs_check 工具。这个工具非常慢，对于大型文件系统无法很好地扩展。因此，它已被 xfs_repair -n 替代。

xfs_repair 需要文件系统上的干净日志才能运行。如果没有完全卸载文件系统，则应在使用 xfs_repair 之前挂载和卸载文件系统。如果日志损坏且无法重新执行，则使用 -L 选项可将日志归零。

重要

只有无法重新执行日志时，必须使用 -L 选项。选项丢弃日志中的所有元数据更新，并产生进一步的不一致。

可以使用 -n 选项，以空运行（仅检查）模式运行 xfs_repair。指定此选项时，不会对文件系统进行任何更改。

xfs_repair 使用非常少的选项。常用的选项包括：

-n

无修改模式。仅检查操作.

-L

将元数据日志归零。仅当无法通过 mount 重新执行日志时才使用。

-m maxmem

将运行期间使用的内存限制为 maxmem MB。可以指定 0 来粗略估算所需的最小内存。

-l logdev

如果存在，指定外部日志设备。

xfs_repair 的所有选项都在 xfs_repair (8) 手册页中指定。

xfs_repair 在运行时执行以下八个基本阶段：

1. inode 和 inode 块映射（寻址）检查.

2. inode 分配映射检查。

3. inode 大小检查。

4. 目录检查。

5. 路径名称检查。

6. 链接数检查。

7. 空闲映射检查。

8. 超级块检查。

如需更多信息，请参阅 xfs_repair (8) 手册页。

xfs_repair 不是交互式的。所有操作都自动执行，无需用户输入。

如果需要在修复前为诊断或测试目的创建元数据镜像，则可以使用 xfs_metadump (8) 和 xfs_mdrestore (8) 工具。

12.2.3. Btrfs

注意

Btrfs 在 Red Hat Enterprise Linux 7 中作为技术预览功能提供，但自 Red Hat Enterprise Linux 7.4 发行版本起已被弃用。它将在以后的 Red Hat Enterprise Linux 主发行版本中删除。

如需更多信息，请参阅 Red Hat Enterprise Linux 7.4 发行注记中的 已弃用的功能。

btrfsck 工具用于检查和修复 btrfs 文件系统。此工具仍处于早期开发阶段，可能无法检测或修复所有类型的文件系统损坏。

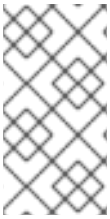
默认情况下，btrfsck 不会对文件系统进行更改；也就是说，它默认运行仅检查模式。如果需要修复，则必须指定 --repair 选项。

btrfsck 在运行时执行以下三个基本阶段：

1. *扩展检查。*
2. *文件系统 root 检查。*
3. *根参考计数检查。*

btrfs-image (8) 工具可用于在修复前创建元数据镜像，以进行诊断或测试。

第 13 章 分区



注意

有关在块设备上使用分区的优缺点的概述，请参阅以下 KBase 文章：
https://access.redhat.com/solutions/163853

使用 parted 工具，您可以：

- 查看现有的分区表。
- 更改现有分区的大小。
- 从可用空间或其他硬盘添加分区。

parted 软件包默认安装在 Red Hat Enterprise Linux 7 中。要启动 parted，以 root 用户身份登录并输入以下命令：

```
# parted /dev/sda
```

使用要配置的驱动器的设备名称替换 /dev/sda。

使用中的设备处理分区

对于不在使用中的设备，设备上的任何分区都不能挂载，且无法启用该设备上的交换空间。

如果要删除或调整分区大小，则该分区所在的设备不能处于使用中。

可以在正在使用的设备上创建新分区，但不建议这样做。

修改分区表

通常不建议使用同一磁盘上的另一个分区修改分区表，因为内核无法重新读取分区表。因此，更改不会应用到正在运行的系统。在上述情况下，重启系统，或者使用以下命令使系统注册新的或修改的分区：

```
# partx --update --nr partition-number disk
```

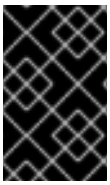
修改当前使用的磁盘的最简单方法是：

1. 如果磁盘中的分区无法卸载，则以救援模式引导系统，例如在系统磁盘的情况下。

2. 当提示挂载文件系统时，请选择 **Skip**。

如果驱动器不包含任何正在使用的分区，没有系统进程使用或锁定文件系统，您可以使用 **umount** 命令卸载分区，并使用 **swapoff** 命令关闭硬盘驱动器上的所有交换空间。

要查看常用的 **parted** 命令，请参阅 表 13.1 “parted 命令”。



重要

不要使用 **parted** 实用程序来创建文件系统。改为使用 **mkfs** 工具。

表 13.1. parted 命令

命令	描述
帮助	显示可用命令列表
mklabel label	为分区表创建磁盘标签
mkpart part-type [fs-type] start-mb end-mb	在不创建新文件系统的情况下创建分区
name minor-num name	只为 **Mac** 和 **PC98** 磁盘标签命名分区
print	显示分区表

命令	描述
quit	退出 parted
rescue start-mb end-mb	抢救丢失的分区（从 Start-mb 到 end-mb）
rm minor-num	删除分区
选择设备	选择要配置的其他设备
set minor-num flag state	在分区中设置标志 ; state 为 on 或 off
toggle [NUMBER [FLAG]	切换分区 NUMBER上的 FLAG 状态
unit UNIT	将默认单位设置为 UNIT

13.1. 查看分区表

查看分区表：

1. 启动 parted。
2. 使用以下命令查看分区表：

```
(parted) print
```

此时会出现类似如下的表：

例 13.1. 分区表

```
Model: ATA ST3160812AS (scsi)
Disk /dev/sda: 160GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End    Size    Type      File system  Flags
 1      32.3kB  107MB  107MB   primary   ext3         boot
 2      107MB   105GB  105GB   primary   ext3
 3      105GB   107GB  2147MB  primary   linux-swap
 4      107GB   160GB  52.9GB  extended  root
```

```
5     107GB   133GB  26.2GB  logical   ext3
6     133GB   133GB  107MB   logical   ext3
7     133GB   160GB  26.6GB  logical             lvm
```

以下是分区表的描述：

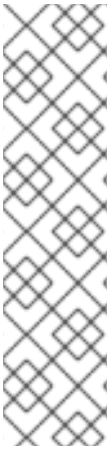
- 模型：ATA ST3160812AS (scsi) ：解释磁盘类型、制造商、型号号和接口。
- 磁盘 /dev/sda: 160GB ：显示块设备和存储容量的文件路径。
- partition Table: msdos: 显示磁盘标签类型。
- 在分区表中，Number 是分区号。例如，次号 1 的分区对应于 /dev/sda1。Start 和 End 值以 MB 为单位。有效的 Types 是 metadata, free, primary, extended, or logical。File system 是文件系统类型。Flags 列中列出了为分区设置的标志。可用的标志有 boot、root、swap、hidden、raid、lvm 或 lba。

分区表中的 File system 可以是以下任意一种：

- ext2
- ext3
- fat16
- fat32
- hfs
- jfs

- *linux-swap*
- *ntfs*
- *reiserfs*
- *hp-ufs*
- *sun-ufs*
- *xfs*

如果设备的 File system 没有显示值，这意味着其文件系统类型未知。



注意

要在不需要重启 parted 的情况下选择不同的设备，请使用以下命令，并将 /dev/sda 替换为您要选择的设备：

```
(parted) select /dev/sda
```

它允许您查看或配置设备的分区表。

13.2. 创建分区

警告

不要试图在正在使用的设备上创建分区。

过程 13.1. 创建分区

1. 在创建分区前，引导进入救援模式，或者卸载该设备中的任何分区，并关闭该设备上的任何交换空间。

2. 启动 **parted** ：

```
# parted /dev/sda
```

使用您要在其上创建分区的设备名称替换 **/dev/sda**。

3. 查看当前的分区表来确定是否有足够空闲空间：

```
(parted) print
```

如果没有足够的可用空间，您可以调整现有分区的大小。如需更多信息，请参阅 *第 13.5 节 “使用 fdisk 重新定义分区大小”*。

通过分区表来确定新分区的开始点和结束点，以及它的分区类型。在一个设备上只能有四个主分区（没有扩展分区）。如果您需要四个以上的分区，您可以有三个主分区、一个扩展分区，以及扩展分区中的多个逻辑分区。有关磁盘分区的概述，请参阅 **Red Hat Enterprise Linux 7** *安装指南* 中的 *磁盘分区简介* 附录。

4. 创建分区：

```
(parted) mkpart part-type name fs-type start end
```

根据您的要求，将 **part-type** 替换为 **primary**、**logical** 或 **extended**。

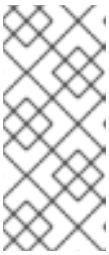
使用分区名称替换 **name** ；GPT 分区表需要 **name**。

使用 **btrfs**、**ext2**、**ext3**、**ext4**、**fat16**、**fat32**、**hfs**、**hfs+**、**linux-swap**、**ntfs**、**reiserfs** 或 **xfs**; **fs-type** 之一替换 **fs-type**。

根据您的要求，将 **start** 以 **MB** 为单位表示。

例如，要在硬盘上创建从 1024MB 到 2048 MB 的 ext3 文件系统的主分区，请输入以下命令：

```
(parted) mkpart primary 1024 2048
```



注意

如果您使用 mkpartfs 命令，文件系统会在分区创建后创建。但是 parted 不支持创建 ext3 文件系统。因此，如果要创建 ext3 文件系统，请使用 mkpart 并使用 mkfs 命令创建文件系统，如稍后所述。

按 Enter 键后，这些更改就会生效，因此请在执行前检查命令。

5.

使用以下命令，查看分区表以确认创建的分区位于分区表中，并具有正确的分区类型、文件系统类型和大小：

```
(parted) print
```

还请记住新分区的次要号，以便您可以在其上面标记任何文件系统。

6.

退出 parted shell：

```
(parted) quit
```

7.

在 parted 关闭后使用以下命令，以确保内核识别新分区：

```
# cat /proc/partitions
```

parted 可以创建的最大分区数为 128。尽管 GUID 分区表 (GPT)规范通过增加为分区表保留的区域来允许更多的分区，但 parted 的常见做法是将其限制为足够容纳 128个 分区的区域。

13.2.1. 格式化和标记分区

要格式化和标记分区，请使用以下流程：

过程 13.2. 格式化和标记分区

过程 13.2. 格式化和标记分区

1. 分区没有文件系统。要创建 ext4 文件系统，请使用：

```
# mkfs.ext4 /dev/sda6
```

警告

格式化分区会永久销毁分区上当前存在的任何数据。

2. 标记分区中的文件系统。例如：如果新分区中的文件系统是 /dev/sda6，而您想要将其标记为 Work，请使用：

```
# e2label /dev/sda6 "Work"
```

默认情况下，安装程序使用分区挂载点作为标签，以确保标签是唯一的。您可以使用您想要的任何标签。

3. 以 root 身份创建挂载点（如 /work）。

13.2.2. 将分区添加到 /etc/fstab

1. 以 root 用户身份，编辑 /etc/fstab 文件，以使用分区的 UUID 包含新分区。

使用命令 blkid -o list 获取分区 UUID 的完整列表，或使用 blkid 设备 获取单个设备详情。

在 /etc/fstab 中：

- 第一列应当包含 UUID=，后跟文件系统的 UUID。
- 第二列应包含新分区的挂载点。

- 第三列应为文件系统类型：例如 ext4 或 swap。
- 第四列列出了文件系统的挂载选项。此处的单词 defaults 表示分区在引导时使用默认选项挂载。
- 第五个字段和第六个字段指定 backup 和 check 选项。非 root 分区的值示例为 0 2。

2. 重新生成挂载单元以便您的系统注册新配置：

```
# systemctl daemon-reload
```

3. 尝试挂载文件系统来验证配置是否正常工作：

```
# mount /work
```

其它信息

- 如果您需要有关 /etc/fstab 格式的更多信息，请参阅 fstab(5) man page。

13.3. 删除分区

警告

不要试图删除正在使用的设备上的分区。

过程 13.3. 删除分区

1. 在删除分区前，请执行以下操作之一：
   - 引导至救援模式，或者

- 卸载该设备中的任何分区，并关闭该设备上的任何交换空间。

2. 启动 **parted** 工具：

```
# parted device
```

使用删除分区的设备替换 **device** ：例如 **/dev/sda**。

3. 查看当前的分区表以确定要删除的分区的次号：

```
(parted) print
```

4. 使用命令 **rm** 删除分区。例如：要删除次要号为 **3** 的分区：

```
(parted) rm 3
```

按 **Enter** 键后，这些更改就会生效，因此请在提交前检查命令。

5. 删除分区后，使用 **print** 命令确认它已从分区表中删除：

```
(parted) print
```

6. 从 **parted shell** 退出：

```
(parted) quit
```

7. 检查 **/proc/partitions** 文件的内容，以确保内核知道分区已被删除：

```
# cat /proc/partitions
```

8. 从 **/etc/fstab** 文件中删除分区。找到声明删除的分区的行，并将其从文件中删除。

9. 重新生成挂载单元，以便您的系统注册新的 /etc/fstab 配置：

```
# systemctl daemon-reload
```

13.4. 设置分区类型

运行的系统很少使用分区类型，而不是与文件系统类型混淆。但是，分区类型对于 on-the-fly 生成器很重要，如 systemd-gpt-auto-generator，它使用分区类型来自动识别和挂载设备。

您可以启动 fdisk 工具，并使用 t 命令设置分区类型。以下示例演示了如何将第一个分区的分区类型改为 0x83，在 Linux 中默认：

```
# fdisk /dev/sdc
Command (m for help): t
Selected partition 1
Partition type (type L to list all types): 83
Changed type of partition 'Linux LVM' to 'Linux'.
```

parted 工具尝试将分区类型映射到"flags"（对最终用户来说并不方便）来提供一些分区类型控制。parted 工具只能处理某些分区类型，如 LVM 或 RAID。例如，要使用 parted 从第一个分区中删除 lvm 标志，请使用：

```
# parted /dev/sdc 'set 1 lvm off'
```

有关常用分区类型和用于代表它们的十六进制数字的列表，请查看分区中的分区类型表 ： Red Hat Enterprise Linux 7 安装指南 的"打开 One Drive Into Many Appendix"部分。

13.5. 使用 FDISK 重新定义分区大小

fdisk 实用程序允许您创建和操作 GPT、R MBR、SGI 和 BSD 分区表。建议在带有 GUID 分区表 (GPT)的磁盘上，建议使用 parted 工具，因为 fdisk GPT 支持处于实验性阶段。

在调整分区大小前，备份 文件系统中存储的数据并测试流程，因为使用 fdisk 更改分区大小的唯一方法是通过删除和重新创建分区。

重要

您大小的分区必须是特定磁盘上的最后一个分区。

红帽只支持扩展和重新定义 LVM 分区大小。

过程 13.4. 调整分区大小

以下过程仅提供 reference. follows
使用 fdisk 调整分区大小：

1. 卸载该设备：

```
# umount /dev/vda
```

2. 运行 fdisk disk_name。例如：

```
# fdisk /dev/vda
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them. Be careful before using the write command.

Command (m for help):
```

3. 使用 p 选项确定要删除的分区的行号。

```
Command (m for help): p
Disk /dev/vda: 16.1 GB, 16106127360 bytes, 31457280 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x0006d09a

Device    Boot      Start         End      Blocks   Id  System
/dev/vda1   *        2048     1026047      512000   83  Linux
/dev/vda2         1026048    31457279    15215616   8e  Linux LVM
```

4. 使用 d 选项删除分区。如果有多个分区可用，fdisk 会提示您提供要删除的多个分区：

```
Command (m for help): d
Partition number (1,2, default 2): 2
Partition 2 is deleted
```

5.

使用 n 选项创建分区并按照提示进行操作。允许足够空间以供将来重新定义大小。fdisk 默认行为（按 Enter）是使用设备上的所有空间。您可以根据扇区指定分区的结束，或使用 + <size> <suffix> 指定人类可读的大小，如 +500M 或 +10G。

如果您不想使用所有可用空间，红帽建议使用人类可读的大小规格，因为 fdisk 将分区的末尾与物理扇区一致。如果您通过提供准确数字（扇区）来指定大小，fdisk 不会对齐分区的末尾。

```
Command (m for help): n
Partition type:
  p   primary (1 primary, 0 extended, 3 free)
  e   extended
Select (default p): *Enter*
Using default response p
Partition number (2-4, default 2): *Enter*
First sector (1026048-31457279, default 1026048): *Enter*
Using default value 1026048
Last sector, +sectors or +size{K,M,G} (1026048-31457279, default 31457279): +500M
Partition 2 of type Linux and of size 500 MiB is set
```

6.

将分区类型设置为 LVM：

```
Command (m for help): t
Partition number (1,2, default 2): *Enter*
Hex code (type L to list all codes): 8e
Changed type of partition 'Linux' to 'Linux LVM'
```

7.

当您确定更改正确时，使用 w 选项写入更改，因为错误可能会导致所选分区不稳定。

8.

在设备上运行 e2fsck 来检查一致性：

```
# e2fsck /dev/vda
e2fsck 1.41.12 (17-May-2010)
Pass 1:Checking inodes, blocks, and sizes
Pass 2:Checking directory structure
Pass 3:Checking directory connectivity
Pass 4:Checking reference counts
Pass 5:Checking group summary information
ext4-1:11/131072 files (0.0% non-contiguous),27050/524128 blocks
```

9. *挂载该设备：*

```
# mount /dev/vda
```

*如需更多信息，请参阅 **fdisk(8)** 手册页。*

第 14 章 使用 SNAPPER 创建和维护快照

快照卷是目标卷的时间副本中的一个点，它提供了一种将文件系统恢复到之前的状态的方法。snapper 是一个命令行工具，用于为 Btrfs 和精简配置的 LVM 文件系统创建和维护快照。

14.1. 创建 INITIAL SNAPPER 配置

snapper 需要每个在其上运行的卷的离散配置文件。您必须手动设置配置文件。默认情况下，只允许 root 用户执行 snapper 命令。

警告

如果您使用精简配置，请监控 thin-pool 中的可用空间。完全消耗的精简池可能会导致数据丢失。如需更多信息，请参阅 Red Hat Enterprise Linux 7 Logical Volume Manager Administration Guide 中的 Thinly-Provisioned Logical Volumes (Thin Volumes)。

请注意，Btrfs 工具和文件系统作为技术预览提供，这使得它们不适合生产系统。

虽然允许 root 以外的用户或组使用某些 Snapper 命令，但 红帽建议您不要 向其他非特权用户或组群添加升级的权限。这种配置会绕过 SELinux，并可能导致安全风险。红帽建议您与安全团队审核这些功能，并考虑使用 sudo 基础架构。

注意

Btrfs 在 Red Hat Enterprise Linux 7 中作为技术预览功能提供，但自 Red Hat Enterprise Linux 7.4 发行版本起已被弃用。它将在以后的 Red Hat Enterprise Linux 主发行版本中删除。

如需更多信息，请参阅 Red Hat Enterprise Linux 7.4 发行注记中的 已弃用的功能。

过程 14.1. 创建 Snapper 配置文件

1. 创建或选择：

   - 一个精简配置的逻辑卷，它有一个红帽支持的文件系统，或者
   - Btrfs 子卷。

2. 挂载文件系统。

3. 创建定义此卷的配置文件。

   对于 LVM2：

   ```
   # snapper -c config_name create-config -f "lvm(fs_type)" /mount-point
   ```

   例如，要使用 ext4 文件系统（挂载于 /lvm_mount）在 LVM2 子卷上创建一个名为 lvm_config 的配置文件，请使用：

   ```
   # snapper -c lvm_config create-config -f "lvm(ext4)" /lvm_mount
   ```

   对于 Btrfs：

   ```
   # snapper -c config_name create-config -f btrfs /mount-point
   ```

   - -c config_name 选项指定配置文件的名称。
   - create-config 告知 snapper 创建配置文件。
   - f file_system 告知 snapper 要使用的文件系统；如果忽略了 snapper，则尝试检测文件系统。
   - /mount-point 是挂载子卷或精简配置的 LVM2 文件系统的位置。

或者，要在挂载到 /btrfs_mount 的 Btrfs 子卷上创建一个名为 btrfs_config 的配置文件，请使用：

```
# snapper -c btrfs_config create-config -f btrfs /btrfs_mount
```

配置文件存储在 /etc/snapper/configs/ 目录中。

14.2. 创建 SNAPPER 快照

snapper 可以创建以下类型的快照：

pre Snapshot

pre 快照充当后快照的原始卷。这两个操作都紧密关联，旨在跟踪两个点之间的文件系统修改。必须在创建快照前创建预快照。

后快照

post 快照充当预快照的端点。组合 pre 和 post 快照定义了要比较的范围。默认情况下，每个新 snapper 卷都配置为在成功创建相关后创建快照后创建后台比较。

单个快照

单个快照是在特定时间创建的独立快照。它们可用于跟踪修改时间表，并具有以后返回的一般点。

14.2.1. 创建 Pre 和 Post Snapshot Pair

14.2.1.1. 使用 Snapper 创建 Pre Snapshot

要创建预快照，请使用：

```
# snapper -c config_name create -t pre
```

-c config_name 选项根据指定配置文件中的规格创建快照。如果配置文件尚不存在，请参阅 第 14.1 节 “创建 Initial Snapper 配置”。

create -t 选项指定要创建的快照类型。接受的条目为 **pre**、**post** 或 **single**。

例如，要使用 **lvm_config** 配置文件创建预快照，如 第 14.1 节 “创建 Initial Snapper 配置” 中创建，请使用：

```
# snapper -c SnapperExample create -t pre -p
1
```

-p 选项打印创建快照的数量，并是可选的。

14.2.1.2. 使用 Snapper 创建 Post Snapshot

post 快照是快照的端点，应该按照 第 14.2.1.1 节 “使用 Snapper 创建 Pre Snapshot” 中的说明在父预快照后创建。

过程 14.2. 创建后快照

1. 确定预快照的数量：

```
# snapper -c config_name list
```

例如，要显示使用配置文件 **lvm_config** 创建的快照列表，请使用：

```
# snapper -c lvm_config list
Type   | # | Pre # | Date                | User | Cleanup  | Description | Userdata
-------+---+-------+---------------------+------+----------+-------------+---------
single | 0 |       |                     | root |          | current     |
pre    | 1 |       | Mon 06<...>         | root |          |             |
```

此输出显示预快照为数字 **1**。

2. 创建链接到之前创建的预快照的后快照：

```
# snapper -c config_file create -t post --pre-num pre_snapshot_number
```

- **t post** 选项指定创建快照类型的创建。

- --pre-num 选项指定对应的 pre snapshot。

例如，要使用 lvm_config 配置文件创建后快照并链接到预快照号 1，请使用：

```
# snapper -c lvm_config create -t post --pre-num 1 -p
2
```

-p 选项打印创建快照的数量，并是可选的。

3. 现在创建并配对了 pre 和 post 快照 1 和 2。使用 list 命令验证：

```
# snapper -c lvm_config list
Type   | # | Pre # | Date             | User | Cleanup  | Description | Userdata
-------+---+-------+------------------+------+----------+-------------+---------
single | 0 |       |                  | root |          | current     |
pre    | 1 |       | Mon 06<...>      | root |          |             |
post   | 2 | 1     | Mon 06<...>      | root |          |             |
```

14.2.1.3. 在 Pre 和 Post Snapshots 中嵌套命令

您还可以在预快照和后快照中嵌套命令，这在测试时很有用。请参阅 过程 14.3, “在 Pre 和 Post Snapshots 中嵌套命令”，它是以下步骤的快捷方式：

1. 运行 snapper create pre snapshot 命令。

2. 运行命令或命令列表，以对文件系统内容有可能的影响。

3. 运行 snapper create post snapshot 命令。

过程 14.3. 在 Pre 和 Post Snapshots 中嵌套命令

1. 在预快照和后快照中嵌套命令：

```
# snapper -c lvm_config create --command "command_to_be_tracked"
```

例如，要跟踪 /lvm_mount/hello_file 文件的创建：

```
# snapper -c lvm_config create --command "echo Hello > /lvm_mount/hello_file"
```

2. 要验证这一点，请使用 status 命令：

```
# snapper -c config_file status first_snapshot_number..second_snapshot_number
```

例如，要跟踪第一步中所做的更改：

```
# snapper -c lvm_config status 3..4
+..... /lvm_mount/hello_file
```

如果需要，使用 list 命令验证快照的数量。

有关 status 命令的详情请参考 第 14.3 节 “跟踪 Snapper Snapshots 间的更改”。

请注意，给定示例中 命令不能保证快照捕获的唯一内容。snapper 还记录系统修改的任何内容，而不仅仅是用户修改的内容。

14.2.2. 创建单个 Snapper 快照

创建单个 snapper 快照与创建预快照或后快照类似，只有 create -t 选项指定单个。单个快照用于随着时间的推移创建单个快照，而不与任何其他快照相关。但是，如果您想创建 LVM2 精简卷的快照，而无需自动生成比较或列出附加信息，红帽建议为此使用 System Storage Manager 而不是 Snapper，如 第 16.2.6 节 “Snapshot” 所述。

要创建单个快照，请使用：

```
# snapper -c config_name create -t single
```

例如，以下命令使用 lvm_config 配置文件创建单个快照。

```
# snapper -c lvm_config create -t single
```

虽然单个快照没有专门设计来跟踪更改，但您可以使用 snapper diff、xadiff 和 status 命令比较任何两个快照。有关这些命令的详情请参考 第 14.3 节 “跟踪 Snapper Snapshots 间的更改”。

14.2.3. 配置 Snapper 以获取自动化快照

进行自动快照是 Snapper 的主要功能之一。默认情况下，当您为卷配置 Snapper 时，Snapper 会每小时开始对卷执行快照。

在默认配置下，Snapper 保留：

- 10 每小时快照，最终每小时快照保存为"每天"快照。
- 10 个每日快照，一个月的最终每日快照会保存为"月"快照。
- 10 个月快照，最终每月快照保存为"每年"快照。
- 10 年快照。

请注意，Snapper 默认不会超过 50 个快照。但是，Snapper 默认保留在 1,800 秒以前创建的所有快照。

默认配置在 /etc/snapper/config-templates/default 文件中指定。当您使用 snapper create-config 命令创建配置时，会根据默认配置设置任何未指定的值。您可以编辑 /etc/snapper/configs/config_name 文件中任何定义的卷的配置。

14.3. 跟踪 SNAPPER SNAPSHOTS 间的更改

使用 状态、diff 和 xadiff 命令来跟踪在快照之间对子卷所做的更改：

status

status 命令显示在两个快照之间创建、修改或删除的文件和目录列表，这是两个快照之间更改的完整列表。您可以使用此命令获得更改的概述，而无需过量详情。

如需更多信息，请参阅 第 14.3.1 节 “将更改与 status 命令进行比较”。

diff

如果至少检测到一次修改，则 diff 命令显示从 status 命令接收的两个快照之间的修改文件和目录差异。

如需更多信息，请参阅 第 14.3.2 节 “将更改与 diff 命令进行比较”。

xadiff

xadiff 命令比较两个快照之间文件或目录的扩展属性变化。

如需更多信息，请参阅 第 14.3.3 节 “将更改与 xadiff 命令进行比较”。

14.3.1. 将更改与 status 命令进行比较

status 命令显示在两个快照之间创建、修改或删除的文件和目录列表。

要显示两个快照之间的文件状态，请使用：

```
# snapper -c config_file status first_snapshot_number..second_snapshot_number
```

如果需要，使用 list 命令确定快照号。

例如，以下命令使用配置文件 lvm_config 显示快照 1 和 2 之间所做的更改。

```
#snapper -c lvm_config status 1..2
tp.... /lvm_mount/dir1
-..... /lvm_mount/dir1/file_a
c.ug.. /lvm_mount/file2
```

```
+..... /lvm_mount/file3
....x. /lvm_mount/file4
cp..xa /lvm_mount/file5
```

作为列，将输出的第一个部分中的字母和点读为：

```
+..... /lvm_mount/file3
||||||
123456
```

列 1 表示对文件（目录条目）类型的任何修改。可能的值有：

列 1

输出	含义
.	没有改变。
+	文件已创建。
-	删除文件。
c	内容更改。
t	目录条目的类型已更改。例如，前一个符号链接已更改为具有相同文件名的常规文件。

列 2 表示文件权限中的任何更改。可能的值有：

列 2

输出	含义
.	没有更改权限。
p	更改了权限。

列 3 表示用户所有权的任何更改。可能的值有：

列 3

输出	含义
.	没有更改用户所有权。
u	用户所有权已更改。

列 4 表示组所有权的任何更改。可能的值有：

列 4

输出	含义
.	没有更改组所有权。
g	组所有权已更改。

列 5 表示扩展属性中的任何更改。可能的值有：

列 5

输出	含义
.	没有更改扩展属性。
x	扩展属性已更改。

列 6 表示访问控制列表(ACL)中的任何更改。可能的值有：

列 6

输出	含义
.	没有更改 ACL。
a	修改 ACL。

14.3.2. 将更改与 diff 命令进行比较

diff 命令显示两个快照之间修改的文件和目录的更改。

```
# snapper -c config_name diff first_snapshot_number..second_snapshot_number
```

如果需要，使用 list 命令确定快照的数量。

例如，要比较使用 lvm_config 配置文件在快照 1 和快照 2 之间所做的更改，请使用：

```
# snapper -c lvm_config diff 1..2
--- /lvm_mount/.snapshots/13/snapshot/file4 19<...>
+++ /lvm_mount/.snapshots/14/snapshot/file4 20<...>
@@ -0,0 +1 @@
+words
```

此输出显示 file4 已被修改，以将"词语"添加到文件中。

14.3.3. 将更改与 xadiff 命令进行比较

xadiff 命令比较文件或目录的扩展属性在两个快照之间变化的方式：

```
# snapper -c config_name xadiff first_snapshot_number..second_snapshot_number
```

如果需要，使用 list 命令确定快照的数量。

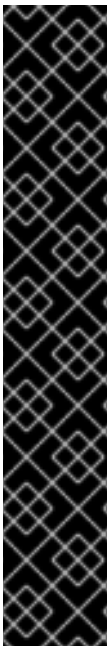
例如，要显示使用 lvm_config 配置文件执行的快照号 1 和快照号 2 之间的 xadiff 输出，请使用：

```
# snapper -c lvm_config xadiff 1..2
```

14.4. 在快照之间撤销更改

要在两个现有的 Snapper 快照之间反向更改，请使用 undochange 命令，其中 1 是第一个快照，2 是第二个快照：

```
snapper -c config_name undochange 1..2
```



重要

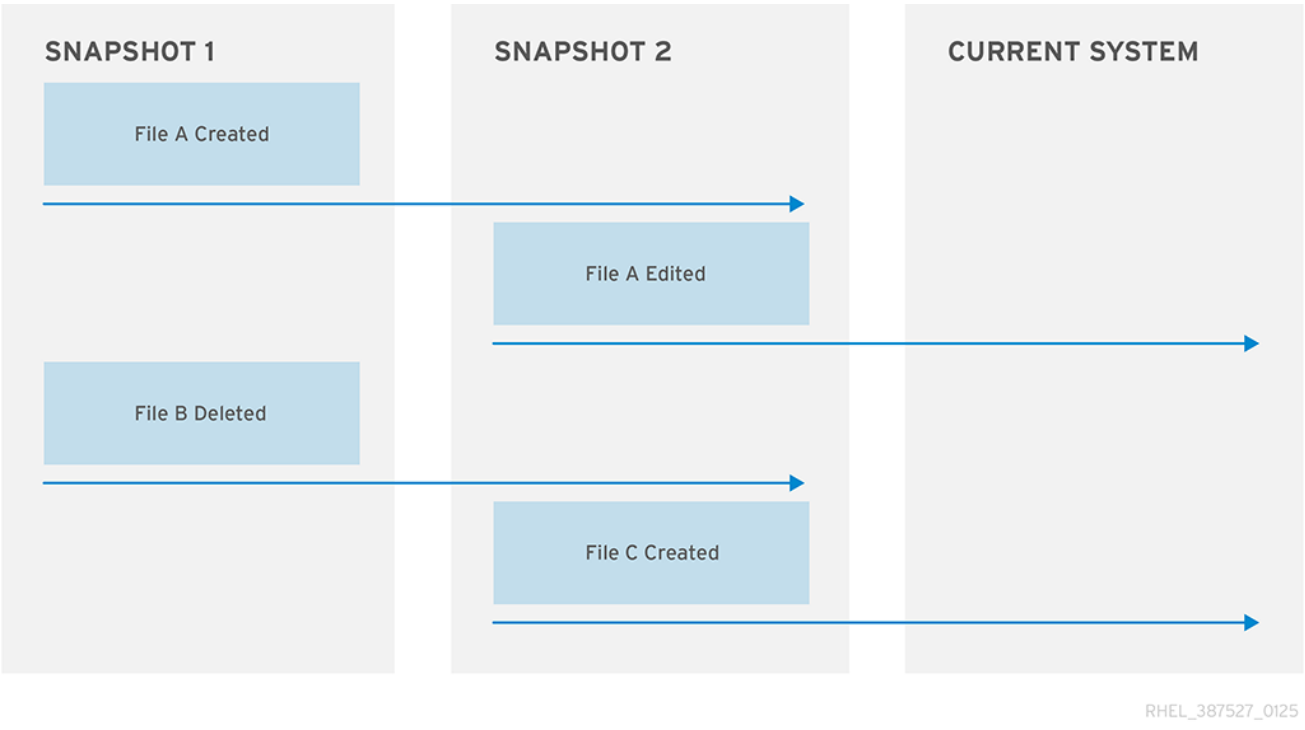
使用 undochange 命令不会将 Snapper 卷恢复到其原始状态，且不会提供数据一致性。任何在指定范围之外的文件修改（如快照 2 后）将在恢复后保持不变，例如快照 1 的状态。例如，如果运行 undochange 来撤销用户的创建，则该用户拥有的任何文件仍然可以保留。

此外，也没有保证文件一致性为快照的机制，因此使用 undochange 命令时，已存在的任何不一致的不一致都将传输回快照。

不要将 Snapper undochange 命令与 root 文件系统一起使用，因为这样做可能会导致失败。

下图显示了 undochange 命令如何工作：

图 14.1. snapper Status over Time



[D]

图中显示了创建 snapshot_1 的时间点，创建 file_a，然后删除 file_b。然后，会创建 Snapshot_2，之后编辑 file_a 并创建了 file_c。现在，这是系统的当前状态。当前系统具有 file_a 的编辑版本，无 file_b 以及新创建的 file_c。

当调用 undochange 命令时，Snapper 会在第一个列出的快照和第二个快照之间生成修改的文件列表。在图中，如果您使用 snapper -c SnapperExample undochange 1..2 命令，Snapper 会创建修改的文件列表（即 file_a 已创建；删除了 file_b ），并将其应用到当前的系统。因此：

- 当前系统没有 file_a，因为它必须在 snapshot_1 创建时创建。
- file_b 将存在，从 snapshot_1 复制到当前系统。
- file_c 将存在，因为它的创建是在指定时间之外。

请注意，如果 file_b 和 file_c 冲突，系统可能会损坏。

您还可以使用 snapper -c SnapperExample undochange 2..1 命令。在这种情况下，当前系统会将 edit 的 file_a 版本替换为从 snapshot_1 复制的版本，这会撤销在创建 snapshot_2 后对该文件的编辑。

使用 mount 和 unmount 命令反向更改

undochange 命令并不总是是恢复修改的最佳方法。使用 status 和 diff 命令，您可以做出合格的决定，并使用 mount 和 unmount 命令而不是 Snapper。只有在您要挂载快照并浏览独立于 Snapper 工作流的内容时，mount 和 unmount 命令才有用。

如果需要，mount 命令在挂载前激活相应的 LVM Snapper 快照。例如，如果您有兴趣挂载快照并手动提取多个文件的旧版本，请使用 mount 和 unmount 命令。要手动恢复文件，请将它们从挂载的快照复制到当前文件系统中。当前文件系统，快照 0 是 过程 14.1, “创建 Snapper 配置文件” 中创建的实时文件系统。将文件复制到原始 /mount-point 的子树中。

对于显式客户端请求，使用 mount 和 unmount 命令。/etc/snapper/configs/config_name 文件包含 ALLOW_USERS= 和 ALLOW_GROUPS= 变量，您可以在其中添加用户和组。然后，snapper d 允许您为添加的用户和组执行挂载操作。

14.5. 删除 SNAPPER 快照

删除快照：

```
# snapper -c config_name delete snapshot_number
```

您可以使用 list 命令验证快照是否已成功删除。

第 15 章 SWAP 空间

当物理内存(RAM)已满时，将使用 Linux 中的 交换空间。如果系统需要更多的内存资源并且 RAM 已满，内存中的不活动页面将移到交换空间。虽然交换空间可以帮助具有少量 RAM 的计算机，但不应将其视为更多 RAM 的替代品。交换空间位于硬盘驱动器上，其访问时间比物理内存要慢。交换空间可以是专用的交换分区（推荐）、交换文件，或者交换分区和交换文件的组合。请注意，Btrfs 不支持 swap 空间。

过去数年，推荐的 swap 空间会随系统中的 RAM 量增加而线性增大。然而，现代系统通常包含了成百 GB 内存。因此，推荐的交换空间被视为系统内存工作负载的功能，而不是系统内存的功能。

表 15.1 “推荐的系统交换空间” 根据系统中的 RAM 量以及是否有足够的内存供系统休眠显示推荐的交换分区的大小。推荐的 swap 分区会在安装过程中自动建立。但是，为了允许休眠，您需要在自定义分区阶段编辑交换空间。

表 15.1 “推荐的系统交换空间” 中的建议对于内存不足的系统（1 GB 及更少）尤为重要。无法在这些系统中分配足够 swap 空间可能会导致问题，如不稳定，甚至会导致安装的系统无法引导。

表 15.1. 推荐的系统交换空间

系统中的 RAM 量	推荐的 swap 空间	如果允许休眠则推荐使用 swap 空间
⩽ 2 GB	RAM 量的 2 倍	RAM 量的 3 倍
> 2 GB – 8 GB	与 RAM 量相等	RAM 量的 2 倍
> 8 GB – 64 GB	至少 4 GB	RAM 量的 1.5 倍
> 64 GB	至少 4 GB	不推荐休眠

注意

不建议使用超过 64 GB RAM 的系统进行休眠的原因有两个。首先，休眠需要额外的空间用于 inflated （可能不经常使用）交换区。其次，将驻留的数据从 RAM 移到磁盘并回到磁盘的过程可以花费大量时间完成。

在 表 15.1 “推荐的系统交换空间” 中列出的每个范围之间的边界（例如，具有 2 GB、8 GB 或 64 GB 系统 RAM 的系统），可以根据所选交换空间和休眠支持进行判断。如果您的系统资源允许这样做，增加

交换空间可能会导致更好的性能。

请注意，将交换空间分布到多个存储设备也可以提高交换空间的性能，特别是在具有快速驱动器、控制器和接口的系统上。

重要

在修改时，不应使用分配被为交换空间的文件系统和 LVM2 卷。如果系统进程或内核正在使用交换空间，则任何修改交换的尝试都会失败。使用 free 和 cat /proc/swaps 命令验证交换的使用量以及位置。

在系统以 救援模式 引导时，您应该修改交换空间，请参阅 Red Hat Enterprise Linux 7 安装指南 中的 在 Rescue Mode 中引导您的计算机。当提示挂载文件系统时，请选择 Skip。

15.1. 添加交换空间

有时，需要在安装后添加更多的交换空间。例如，您可以将系统中的 RAM 量从 1 GB 升级到 2 GB，但只有 2 GB 的交换空间。如果您执行内存密集型操作或运行需要大量内存的应用程序，则最好将交换空间大小增加到 4 GB。

您有三个选项：创建一个新的交换分区、创建一个新的交换文件，或者扩展现有 LVM2 逻辑卷上的交换空间。建议您扩展现有逻辑卷。

15.1.1. 在 LVM2 逻辑卷上扩展交换空间

默认情况下，Red Hat Enterprise Linux 7 在安装过程中使用所有可用空间。如果您的系统是这种情况，那么您必须首先向交换空间使用的卷组中添加一个新的物理卷。

在交换空间的卷组中添加额外的存储后，现在可以扩展它了。要做到这一点，请执行以下步骤（假设 /dev/VolGroup00/LogVol01 是您要扩展为 2 GB 的卷）：

过程 15.1. 在 LVM2 逻辑卷上扩展交换空间

1. 为关联的逻辑卷禁用交换：

   ■

```
# swapoff -v /dev/VolGroup00/LogVol01
```

2. 将 LVM2 逻辑卷大小调整 2 GB：

```
# lvresize /dev/VolGroup00/LogVol01 -L +2G
```

3. 格式化新 swap 空间：

```
# mkswap /dev/VolGroup00/LogVol01
```

4. 启用扩展的逻辑卷：

```
# swapon -v /dev/VolGroup00/LogVol01
```

5. 要测试是否成功扩展并激活 swap 逻辑卷，请检查活跃 swap 空间：

```
$ cat /proc/swaps
$ free -h
```

15.1.2. 为交换空间创建 LVM2 逻辑卷

要添加一个 swap 卷组 2 GB，假设 /dev/VolGroup00/LogVol02 是您要添加的交换卷：

1. *创建大小为 2 GB 的 LVM2 逻辑卷：*

```
# lvcreate VolGroup00 -n LogVol02 -L 2G
```

2. *格式化新 swap 空间：*

```
# mkswap /dev/VolGroup00/LogVol02
```

3. *在 /etc/fstab 文件中添加以下条目：*

```
/dev/VolGroup00/LogVol02   swap    swap    defaults    0 0
```

4. *重新生成挂载单元以便您的系统注册新配置：*

```
# systemctl daemon-reload
```

5. *在逻辑卷中激活 swap：*

```
# swapon -v /dev/VolGroup00/LogVol02
```

6. *要测试是否成功创建并激活 swap 逻辑卷，请检查活跃 swap 空间：*

```
$ cat /proc/swaps
$ free -h
```

15.1.3. 创建一个交换文件

添加一个交换文件：

过程 15.2. 添加交换文件

1. *以 MB 为单位确定新交换文件的大小，再乘以 1024 来确定块的数量。例如： 64MB swap 文件的块大小为 65536。*

2. 创建一个空文件：

```
# dd if=/dev/zero of=/swapfile bs=1024 count=65536
```

使用与所需块大小相等的值替换 **count**。

3. 使用以下命令设定 **swap** 文件：

```
# mkswap /swapfile
```

4. 更改交换文件的安全性，使其不可读。

```
# chmod 0600 /swapfile
```

5. 要在引导时启用 **swap** 文件，以 **root** 用户身份编辑 **/etc/fstab**，使其包含以下条目：

```
/swapfile       swap            swap    defaults        0 0
```

下次系统引导时，它会激活新的 **swap** 文件。

6. 重新生成挂载单元，以便您的系统注册新的 **/etc/fstab** 配置：

```
# systemctl daemon-reload
```

7. 立即激活 **swap** 文件：

```
# swapon /swapfile
```

8. 要测试新 swap 文件是否已成功创建并激活，请检查活跃 swap 空间：

```
$ cat /proc/swaps
$ free -h
```

15.2. 删除交换空间

有时，安装后减少交换空间要谨慎。例如：您将系统中的 RAM 大小从 1GB 降到 512MB，但仍分配了 2GB swap 空间。最好将交换空间大小减少到 1 GB，因为较大的 2 GB 可能会浪费磁盘空间。

您有三个选项：删除用于交换空间的整个 LVM2 逻辑卷、删除交换文件或减少现有 LVM2 逻辑卷上的交换空间。

15.2.1. 减少 LVM2 逻辑卷上的交换空间

要减少 LVM2 交换逻辑卷（假设 /dev/VolGroup00/LogVol01 是您要减少的卷）：

过程 15.3. 减少 LVM2 交换空间

1. 为关联的逻辑卷禁用交换：

```
# swapoff -v /dev/VolGroup00/LogVol01
```

2. 将 LVM2 逻辑卷减少 512 MB：

```
# lvreduce /dev/VolGroup00/LogVol01 -L -512M
```

3. 格式化新 swap 空间：

```
# mkswap /dev/VolGroup00/LogVol01
```

4. 在逻辑卷中激活 swap：

```
# swapon -v /dev/VolGroup00/LogVol01
```

5.
要测试是否成功缩小 swap 逻辑卷，请检查活跃 swap 空间：

```
$ cat /proc/swaps
$ free -h
```

15.2.2. 为交换空间删除一个 LVM2 逻辑卷

要删除交换卷组（假设 /dev/VolGroup00/LogVol02 是您要删除的交换卷）：

过程 15.4. 删除一个交换空间卷组

1.
为关联的逻辑卷禁用交换：

```
# swapoff -v /dev/VolGroup00/LogVol02
```

2.
删除 LVM2 逻辑卷：

```
# lvremove /dev/VolGroup00/LogVol02
```

3.
从 /etc/fstab 文件中删除以下关联的条目：

```
/dev/VolGroup00/LogVol02   swap    swap    defaults    0 0
```

4.
重新生成挂载单元以便您的系统注册新配置：

```
# systemctl daemon-reload
```

5.
从 /etc/default/grub 文件中删除对已删除 swap 存储的所有引用：

```
# vi /etc/default/grub
```

6.
重建 grub 配置：

a.
在基于 BIOS 的机器上运行：

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

b. 在基于 UEFI 的机器上，运行：

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

7. 要测试是否成功删除了逻辑卷，请检查活跃 swap 空间：

```
$ cat /proc/swaps
$ free -h
```

15.2.3. 删除一个交换文件

要删除一个交换文件：

过程 15.5. 删除一个交换文件

1. 在 shell 提示符下，执行以下命令禁用交换文件（其中 /swapfile 是交换文件）：

```
# swapoff -v /swapfile
```

2. 从 /etc/fstab 文件中删除其条目。

3. 重新生成挂载单元以便您的系统注册新配置：

```
# systemctl daemon-reload
```

4. 删除实际的文件：

```
# rm /swapfile
```

15.3. 移动交换空间

将 swap 空间从一个位置移到另一个位置：

1. 删除 swap 空间 第 15.2 节 “删除交换空间”。

2. 添加 swap 空间 第 15.1 节 “添加交换空间”。

第 16 章 系统存储管理器(SSM)

系统存储管理器 (SSM)提供了一个命令行界面来管理各种技术中的存储。使用设备映射器(DM)、逻辑卷管理器(LVM)和多设备(MD)，存储系统变得越来越复杂。这会创建一个不友好用户的系统，并更容易出现错误和问题。SSM 通过创建一个统一用户界面来缓解这个问题。此界面允许用户以简单的方式运行复杂的系统。例如，要创建并挂载没有 SSM 的新文件系统，必须使用五个命令。SSM 只需要一个。

本章介绍了 SSM 如何与各种后端交互，以及一些常见用例。

16.1. SSM 后端

SSM 使用 ssmlib/main.py 中的核心抽象层，它符合设备、池和卷抽象，忽略底层技术的细节。后端可以在 ssmlib/main.py 中注册，以处理特定的存储技术方法，如创建、快照 或移除 卷和池。

SSM 中已经注册了几个后端。以下小节提供有关它们的基本信息，以及它们如何处理池、卷、快照和设备的定义。

16.1.1. Btrfs 后端

注意

Btrfs 在 Red Hat Enterprise Linux 7 中作为技术预览功能提供，但自 Red Hat Enterprise Linux 7.4 发行版本起已被弃用。它将在以后的 Red Hat Enterprise Linux 主发行版本中删除。

如需更多信息，请参阅 Red Hat Enterprise Linux 7.4 发行注记中的 已弃用的功能。

Btrfs 是一个具有许多高级功能的文件系统，用作 SSM 中的卷管理后端。可以使用 Btrfs 后端创建池、卷和快照。

16.1.1.1. Btrfs Pool

Btrfs 文件系统本身是池。通过移除设备可以添加更多设备或缩小它进行扩展。当创建 Btrfs 池时，SSM 会创建一个 Btrfs 文件系统。这意味着，每个新的 Btrfs 池都有一个名称与池相同的卷，这些卷在没有删除整个池的情况下不能被删除。默认的 Btrfs 池名称是 btrfs_pool。

池的名称用作文件系统标签。如果系统中已有没有标签的 Btrfs 文件系统，则 Btrfs 池将生成用于内

部使用的名称，格式为 btrfs_device_base_name。

16.1.1.2. Btrfs 卷

在池中的第一个卷后创建的卷与子卷相同。SSM 会在卸载时临时挂载 Btrfs 文件系统，以创建子卷。

卷的名称用作 Btrfs 文件系统中的子卷路径。例如，子卷显示为 /dev/lvm_pool/lvol001。此路径中的每个对象都必须存在，才能创建卷。也可通过其挂载点引用卷。

16.1.1.3. Btrfs Snapshot

可以使用 SSM 在系统中生成任何 Btrfs 卷的快照。请注意，Btrfs 不区分子卷和快照。虽然这意味着 SSM 无法识别 Btrfs 快照目的地，但它会尝试识别特殊名称格式。如果创建快照时指定的名称是特定的模式，则不能识别快照，而是列为常规 Btrfs 卷。

16.1.1.4. Btrfs 设备

Btrfs 不需要在其上创建任何特殊设备。

16.1.2. LVM 后端

可以使用 LVM 创建池、卷和快照。以下定义来自 LVM 点的视图中。

16.1.2.1. LVM 池

LVM 池与 LVM 卷组相同。这意味着，可以在 LVM 池中创建对设备和新逻辑卷进行分组。默认的 LVM 池名称是 lvm_pool。

16.1.2.2. LVM 卷

LVM 卷与常规逻辑卷相同。

16.1.2.3. LVM 快照

从 LVM 卷创建快照时，将创建一个新的 快照卷，然后可以像任何其他 LVM 卷一样处理。与 Btrfs 不同，LVM 能够从常规卷区分快照，因此不需要快照名称来匹配特定的模式。

16.1.2.4. LVM 设备

SSM 使得在物理设备上创建 LVM 后端的需求对用户来说是透明的。

16.1.3. crypt 后端

SSM 中的 crypt 后端使用 cryptsetup 和 dm-crypt 目标来管理加密卷。crypt 后端可用作在常规块设备（或其他卷（如 LVM 或 MD 卷）上创建加密卷的常规后端，或者在单一步骤中创建加密的 LVM 卷。

只能使用 crypt 后端创建卷；不支持池，且不需要特殊设备。

以下小节从 crypt 视图定义卷和快照。

16.1.3.1. 加密卷

加密卷由 dm-crypt 创建，并以未加密的形式代表原始加密设备中的数据。它不支持 RAID 或任何设备连接。

支持两种模式或扩展： luks 和 plain。默认情况下使用 LUKS。有关扩展的更多信息，请参阅 man cryptsetup。

16.1.3.2. crypt Snapshot

虽然 crypt 后端不支持快照，但如果加密卷是在 LVM 卷之上创建的，则可以对卷本身进行快照。然后可以使用 cryptsetup 来打开快照。

16.1.4. 多个设备(MD)后端

目前，MD 后端仅限于收集系统中 MD 卷的信息。

16.2. 常见 SSM 任务

以下小节描述了常见的 SSM 任务。

16.2.1. 安装 SSM

要安装 SSM，请使用以下命令：

```
# yum install system-storage-manager
```

只有在安装了支持的软件包时，才会启用几个后端：

- LVM 后端需要 lvm2 软件包。
- Btrfs 后端需要 btrfs-progs 软件包。
- Crypt 后端需要 device-mapper 和 cryptsetup 软件包。

16.2.2. 显示有关所有删除的设备的信息

使用 list 命令显示所有检测到的设备、池、卷和快照的信息。没有选项的 ssm list 命令会显示以下输出：

```
# ssm list
----------------------------------------------------------
Device        Free      Used      Total  Pool  Mount point
----------------------------------------------------------
/dev/sda                       2.00 GB        PARTITIONED
/dev/sda1                      47.83 MB       /test
/dev/vda                       15.00 GB       PARTITIONED
/dev/vda1                      500.00 MB      /boot
/dev/vda2  0.00 KB  14.51 GB   14.51 GB  rhel
----------------------------------------------------------
------------------------------------------------
Pool  Type  Devices     Free      Used     Total
------------------------------------------------
rhel  lvm   1        0.00 KB  14.51 GB  14.51 GB
------------------------------------------------
--------------------------------------------------------------------------------
Volume          Pool  Volume size  FS     FS size       Free  Type    Mount point
--------------------------------------------------------------------------------
/dev/rhel/root  rhel     13.53 GB  xfs   13.52 GB    9.64 GB  linear  /
/dev/rhel/swap  rhel   1000.00 MB                             linear
/dev/sda1               47.83 MB  xfs   44.50 MB   44.41 MB  part    /test
/dev/vda1              500.00 MB  xfs  496.67 MB  403.56 MB  part    /boot
--------------------------------------------------------------------------------
```

通过使用参数来指定应显示的内容，可以进一步缩小此显示范围。可用选项列表可以通过 ssm list --

help 命令找到。

注意

根据给定的参数，SSM 可能无法显示所有内容。

- *运行 devices 或 dev 参数会省略一些设备。例如，CDROMS 和 DM/MD 设备被有意隐藏，因为它们被列为卷。*
- *有些后端不支持快照，且无法区分快照和常规卷。在其中一个后端上运行 snapshot 参数会导致 SSM 尝试识别卷名称来识别快照。如果 SSM 正则表达式与快照模式不匹配，则不能识别快照。*
- *除了主 Btrfs 卷（文件系统本身）外，任何卸载的 Btrfs 卷都不会显示。*

16.2.3. 创建新池、逻辑卷和文件系统

在本节中，使用默认名称创建一个新池，其具有设备 /dev/vdb 和 /dev/vdc、1G 的逻辑卷和 XFS 文件系统。

要创建此场景的命令是 ssm create --fs xfs -s 1G /dev/vdb /dev/vdc。使用以下选项：

- *--fs 选项指定所需的文件系统类型。当前支持的文件系统类型有：*
  - *ext3*
  - *ext4*
  - *xfs*
  - *btrfs*

- **-s 指定逻辑卷的大小。支持以下后缀来定义单元：**
  - **k 或 k 表示 KB**
  - **M 或 m 用于 MB**
  - **G 或 G 表示千兆字节**
  - **T 或 t for TB**
  - **P 或 p for PBs**
  - **e 或 e 用于 exabytes**
- **Additionaly 使用 -s 选项，新大小可指定为百分比。查看示例：**
  - **总池大小达到 10%**
  - **空闲池空间的 10%FREE**
  - **10%USED 是已用池空间的 10%**

列出的两个设备 /dev/vdb 和 /dev/vdc 是您要创建的两个设备。

```
# ssm create --fs xfs -s 1G /dev/vdb /dev/vdc
  Physical volume "/dev/vdb" successfully created
  Physical volume "/dev/vdc" successfully created
  Volume group "lvm_pool" successfully created
  Logical volume "lvol001" created
```

ssm 命令还有另外两个可能有用的选项。第一个是 -p pool 命令。这将指定要在其上创建卷的池。如果尚不存在，则 SSM 会创建它。在给定示例中省略了，这会导致 SSM 使用默认名称 lvm_pool。但是，

要使用特定名称来适应任何现有命名约定，应使用 -p 选项。

第二个有用选项是 -n name 命令。这会命名新创建的逻辑卷。与 -p 一样，需要使用特定名称来适合任何现有命名约定。

以下是使用这两个选项的示例：

```
# ssm create --fs xfs -p new_pool -n XFS_Volume /dev/vdd
  Volume group "new_pool" successfully created
  Logical volume "XFS_Volume" created
```

SSM 现在创建了两个物理卷、一个池和一个逻辑卷，且只有一个命令。

16.2.4. 检查文件系统的一致性

ssm check 命令检查卷上的文件系统一致性。可以指定多个卷来检查。如果卷中没有文件系统，则会跳过该卷。

要检查卷 lvol001 中的所有设备，请运行命令 ssm check /dev/lvm_pool/lvol001。

```
# ssm check /dev/lvm_pool/lvol001
Checking xfs file system on '/dev/mapper/lvm_pool-lvol001'.
Phase 1 - find and verify superblock...
Phase 2 - using internal log
        - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
        - scan (but don't clear) agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        - agno = 2
        - agno = 3
        - agno = 4
        - agno = 5
        - agno = 6
        - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - check for inodes claiming duplicate blocks...
        - agno = 0
        - agno = 1
        - agno = 2
        - agno = 3
```

```
        - agno = 4
        - agno = 5
        - agno = 6
No modify flag set, skipping phase 5
Phase 6 - check inode connectivity...
        - traversing filesystem ...
        - traversal finished ...
        - moving disconnected inodes to lost+found ...
Phase 7 - verify link counts...
No modify flag set, skipping filesystem flush and exiting.
```

16.2.5. 增加卷的大小

ssm resize 命令更改指定卷和文件系统的大小。如果没有文件系统，则仅调整卷本身的大小。

在本例中，我们目前在 **/dev/vdb** 上有一个逻辑卷，名为 **lvol001**。

```
# ssm list
----------------------------------------------------------
Device        Free      Used      Total  Pool      Mount point
----------------------------------------------------------
/dev/vda                       15.00 GB           PARTITIONED
/dev/vda1                      500.00 MB          /boot
/dev/vda2     0.00 KB   14.51 GB   14.51 GB  rhel
/dev/vdb   120.00 MB  900.00 MB    1.00 GB  lvm_pool
/dev/vdc                       1.00 GB
----------------------------------------------------------
-----------------------------------------------------
Pool      Type  Devices      Free      Used      Total
-----------------------------------------------------
lvm_pool  lvm   1        120.00 MB  900.00 MB  1020.00 MB
rhel      lvm   1          0.00 KB   14.51 GB    14.51 GB
-----------------------------------------------------
-------------------------------------------------------------------------------------------
Volume                Pool      Volume size  FS     FS size       Free  Type    Mount point
-------------------------------------------------------------------------------------------
/dev/rhel/root        rhel         13.53 GB  xfs   13.52 GB    9.64 GB  linear  /
/dev/rhel/swap        rhel       1000.00 MB                           linear
/dev/lvm_pool/lvol001  lvm_pool    900.00 MB  xfs  896.67 MB  896.54 MB  linear
/dev/vda1                        500.00 MB  xfs  496.67 MB  403.56 MB  part    /boot
-------------------------------------------------------------------------------------------
```

逻辑卷需要被另外 **500MB** 增加。要做到这一点，我们需要在池中添加额外的设备：

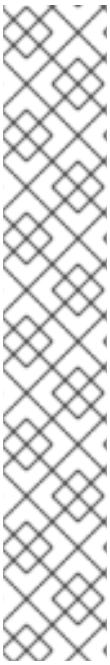
```
~]# ssm resize -s +500M /dev/lvm_pool/lvol001 /dev/vdc
  Physical volume "/dev/vdc" successfully created
  Volume group "lvm_pool" successfully extended
Phase 1 - find and verify superblock...
Phase 2 - using internal log
```

```
        - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
        - scan (but don't clear) agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        - agno = 2
        - agno = 3
        - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - check for inodes claiming duplicate blocks...
        - agno = 0
        - agno = 1
        - agno = 2
        - agno = 3
No modify flag set, skipping phase 5
Phase 6 - check inode connectivity...
        - traversing filesystem ...
        - traversal finished ...
        - moving disconnected inodes to lost+found ...
Phase 7 - verify link counts...
No modify flag set, skipping filesystem flush and exiting.
  Extending logical volume lvol001 to 1.37 GiB
  Logical volume lvol001 successfully resized
meta-data=/dev/mapper/lvm_pool-lvol001 isize=256    agcount=4, agsize=57600 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0
data     =                       bsize=4096   blocks=230400, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal               bsize=4096   blocks=853, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
data blocks changed from 230400 to 358400
```

SSM 在设备上运行检查，然后根据指定数量扩展卷。这可以通过 ssm list 命令验证。

```
# ssm list
----------------------------------------------------------
Device        Free       Used      Total  Pool      Mount point
----------------------------------------------------------
/dev/vda                        15.00 GB           PARTITIONED
/dev/vda1                       500.00 MB          /boot
/dev/vda2    0.00 KB    14.51 GB   14.51 GB  rhel
/dev/vdb     0.00 KB  1020.00 MB    1.00 GB  lvm_pool
/dev/vdc   640.00 MB   380.00 MB    1.00 GB  lvm_pool
----------------------------------------------------------
--------------------------------------------------
Pool      Type  Devices      Free      Used     Total
--------------------------------------------------
lvm_pool  lvm   2        640.00 MB   1.37 GB   1.99 GB
```

```
rhel     lvm   1        0.00 KB  14.51 GB  14.51 GB
--------------------------------------------------
------------------------------------------------------------------------------------------
Volume              Pool      Volume size  FS      FS size       Free  Type    Mount point
------------------------------------------------------------------------------------------
/dev/rhel/root      rhel      13.53 GB  xfs    13.52 GB     9.64 GB  linear  /
/dev/rhel/swap      rhel      1000.00 MB                      linear
/dev/lvm_pool/lvol001  lvm_pool     1.37 GB  xfs     1.36 GB     1.36 GB  linear
/dev/vda1                     500.00 MB  xfs   496.67 MB   403.56 MB  part    /boot
------------------------------------------------------------------------------------------
```



注意

只能缩小 LVM 卷的大小；不支持其他卷类型。这可以通过使用 - 而不是 + 完成。例如：要将 LVM 卷的大小减小到 50M，命令将是：

```
# ssm resize -s-50M /dev/lvm_pool/lvol002
  Rounding size to boundary between physical extents: 972.00 MiB
  WARNING: Reducing active logical volume to 972.00 MiB
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce lvol002? [y/n]: y
  Reducing logical volume lvol002 to 972.00 MiB
  Logical volume lvol002 successfully resized
```

如果没有 + 或 -，则该值会取为绝对值。

16.2.6. Snapshot

要为现有卷执行快照，请使用 ssm snapshot 命令。

注意

如果卷所属的后端不支持快照，则此操作会失败。

要为 lvol001 创建快照，请使用以下命令：

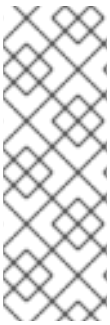
```
# ssm snapshot /dev/lvm_pool/lvol001
  Logical volume "snap20150519T130900" created
```

要验证这一点，请使用 ssm 列表，并记下额外的快照部分。

```
# ssm list
----------------------------------------------------------
Device        Free       Used      Total  Pool      Mount point
----------------------------------------------------------
/dev/vda                        15.00 GB           PARTITIONED
/dev/vda1                       500.00 MB          /boot
/dev/vda2  0.00 KB    14.51 GB   14.51 GB  rhel
/dev/vdb   0.00 KB  1020.00 MB    1.00 GB  lvm_pool
/dev/vdc                        1.00 GB
----------------------------------------------------------
-----------------------------------------------------
Pool      Type  Devices     Free       Used       Total
-----------------------------------------------------
lvm_pool  lvm   1        0.00 KB  1020.00 MB  1020.00 MB
rhel      lvm   1        0.00 KB    14.51 GB    14.51 GB
-----------------------------------------------------
-------------------------------------------------------------------------------------------
Volume               Pool      Volume size  FS      FS size       Free  Type    Mount point
-------------------------------------------------------------------------------------------
/dev/rhel/root       rhel        13.53 GB  xfs    13.52 GB     9.64 GB  linear  /
/dev/rhel/swap       rhel       1000.00 MB                             linear
/dev/lvm_pool/lvol001  lvm_pool    900.00 MB  xfs   896.67 MB   896.54 MB  linear
/dev/vda1                       500.00 MB  xfs   496.67 MB   403.56 MB  part    /boot
-------------------------------------------------------------------------------------------
--------------------------------------------------------------------------
Snapshot                        Origin   Pool      Volume size     Size  Type
--------------------------------------------------------------------------
/dev/lvm_pool/snap20150519T130900  lvol001  lvm_pool    120.00 MB  0.00 KB  linear
--------------------------------------------------------------------------
```

16.2.7. 删除项

ssm remove 用于删除项目，可以是设备、池或卷。



注意

如果在删除时池使用了设备，它将会失败。这可以通过使用 -f 参数强制使用。

如果卷在删除时挂载，它将会失败。与设备不同，无法使用 -f 参数强制使用它。

要删除 lvm_pool 及其其中的所有内容，请使用以下命令：

```
# ssm remove lvm_pool
Do you really want to remove volume group "lvm_pool" containing 2 logical volumes? [y/n]: y
Do you really want to remove active logical volume snap20150519T130900? [y/n]: y
  Logical volume "snap20150519T130900" successfully removed
```

```
Do you really want to remove active logical volume lvol001? [y/n]: y
  Logical volume "lvol001" successfully removed
  Volume group "lvm_pool" successfully removed
```

16.3. SSM 资源

有关 SSM 的更多信息，请参阅以下资源：

- man ssm 页面提供了很好的描述和示例，以及有关所有命令和选项的详细信息，具体要记录在此处。
- SSM 的本地文档存储在 doc/ 目录中。
- SSM wiki 可从 访问 http://storagemanager.sourceforge.net/index.html。
- 邮件列表可从 中订阅，并从 https://lists.sourceforge.net/lists/listinfo/storagemanager-devel 中邮件列表存档 http://sourceforge.net/mailarchive/forum.php?forum_name=storagemanager-devel。邮件列表是开发人员通信的位置。当前没有用户邮件列表，因此也可以自由地发布问题。

第 17 章 磁盘配额

通过实施磁盘配额来限制磁盘空间，配额可在用户消耗太多磁盘空间或分区已满前提醒系统管理员。

可以为单独的用户以及用户组配置磁盘配额。这样便可以将分配给用户特定文件(例如电子邮件)的空间与分配给用户工作的项目的空间分开管理(假设项目有自己的组)。

此外，配额不仅可用来控制所消耗磁盘块的数量，还可以用来控制 inode （包含 UNIX 文件系统中文件信息的数据结构）数。由于 inode 用于包含与文件有关信息，因此允许控制可以创建的文件数。

必须安装 quota RPM 才能实现磁盘配额。

注意

本章适用于所有文件系统，但有些文件系统都有自己的配额管理工具。有关适用的文件系统，请查看相应的描述。

对于 XFS 文件系统，请参阅 第 3.3 节 “XFS 配额管理”。

Btrfs 没有磁盘配额，因此不涵盖。

17.1. 配置磁盘配额

要实现磁盘配额，请执行以下步骤：

1. 通过修改 /etc/fstab 文件，为每个文件系统启用配额。
2. 重新挂载文件系统。
3. 创建配额数据库文件，并生成磁盘使用情况表。
4. 分配配额策略。

在以下小节中会详细讨论这些步骤的具体内容。

17.1.1. 启用配额

过程 17.1. 启用配额

1. 以 root 身份登录。

2. 编辑 /etc/fstab 文件。

3. 将 usrquota 或 grpquota 或两个选项添加到需要配额的文件系统中。

例 17.1. 编辑 /etc/fstab

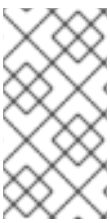
例如，要使用文本编辑器 vim，请键入以下内容：

```
# vim /etc/fstab
```

例 17.2. 添加配额

```
/dev/VolGroup00/LogVol00 /        ext3    defaults        1 1
LABEL=/boot             /boot     ext3    defaults        1 2
none                    /dev/pts  devpts  gid=5,mode=620  0 0
none                    /dev/shm  tmpfs   defaults        0 0
none                    /proc     proc    defaults        0 0
none                    /sys      sysfs   defaults        0 0
/dev/VolGroup00/LogVol02 /home    ext3    defaults,usrquota,grpquota  1 2
/dev/VolGroup00/LogVol01 swap     swap    defaults        0 0 . . .
```

在这个示例中，/home 文件系统同时启用了用户和组配额。



备注

以下示例假定在安装 Red Hat Enterprise Linux 过程中创建了单独的 /home 分区。root (/)分区可用于设置 /etc/fstab 文件中的配额策略。

17.1.2. 重新挂载文件系统

添加 usrquota 或 grpquota 两个选项后，重新挂载修改 fstab 条目的每个文件系统。如果文件系统没有被任何进程使用，请使用以下方法之一：

- 运行 umount 命令，后跟 mount 命令以重新挂载文件系统。有关挂载和卸载各种文件系统类型的具体语法，请参阅 umount 和 mount man page。

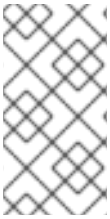
- 运行 mount -o remount file-system 命令（其中 file-system 是文件系统的名称）以重新挂载文件系统。例如，要重新挂载 /home 文件系统，请运行 mount -o remount /home 命令。

如果文件系统当前正在使用，则重新挂载文件系统的最简单方法是重新启动系统。

17.1.3. 创建配额数据库文件

重新挂载每个启用了配额的文件系统后，请运行 quotacheck 命令。

quotacheck 命令检查启用了配额的文件系统，并为每个文件系统构建一个当前磁盘使用情况的表。该表随后用于更新操作系统磁盘使用情况的副本。此外，还会更新文件系统的磁盘配额文件。



注意

quotacheck 命令对 XFS 没有影响，因为磁盘使用量表在挂载时会自动完成。如需更多信息，请参阅 man page xfs_quota (8)。

过程 17.2. 创建配额数据库文件

1. 使用以下命令在文件系统中创建配额文件：

```
# quotacheck -cug /file system
```

2. 使用以下命令为每个文件系统生成当前磁盘用量表：

```
# quotacheck -avug
```

以下是用于创建配额文件的选项：

c

指定应为每个启用配额的文件系统创建配额文件。

u

检查用户配额。

g

检查组配额。如果只指定 -g，则仅创建组配额文件。

如果未指定 -u 或 -g 选项，则仅创建用户配额文件。

以下选项用于生成当前磁盘用量表：

a

检查所有启用了配额的、本地挂载的文件系统

v

在配额检查进行时显示详细的状态信息

u

检查用户磁盘配额信息

g

检查组磁盘配额信息

在 quotacheck 完成运行后，与启用的配额（用户或组或两者）对应的配额文件会填充每个启用了配额的本地挂载的文件系统（如 /home ）的数据。

17.1.4. 为每个用户分配配额

最后一步是使用 edquota 命令分配磁盘配额。

前提条件

- 用户必须在设置用户配额前存在。

过程 17.3. 为每个用户分配配额

1. 要为用户分配配额，请使用以下命令：

```
# edquota username
```

使用您要为其分配配额的用户替换 username。

2. 要验证是否为该用户设定了配额，使用以下命令：

```
# quota username
```

例 17.3. 为用户分配配额

例如，如果在 /etc/fstab 中为 /home 分区（以下示例中的/dev/VolGroup00/LogVol02 ）启用了配额，并且执行了命令 edquota testuser，则会在配置为系统默认设置的编辑器中显示以下内容：

```
Disk quotas for user testuser (uid 501):
  Filesystem                   blocks     soft     hard    inodes   soft   hard
  /dev/VolGroup00/LogVol02  440436        0        0     37418      0      0
```

备注

edquota 使用由 EDITOR 环境变量定义的文本编辑器。要更改编辑器，将 ~/.bash_profile 文件中的 EDITOR 环境变量设置为您选择的编辑器的完整路径。

第一列是启用了配额的文件系统的名称。第二列显示目前该用户使用的块数。下面的两列是为该用户在文件系统中设定软限制和硬限制。inodes 列显示用户当前使用的 inode 数。最后两列是为该用户在文件系统中设定软和硬的内节点限制。

硬块限制是用户或者组群可以使用的绝对最大磁盘空间量。达到这个限制后，就无法再使用其他磁盘空间。

软块限制定义可以使用的最大磁盘空间量。然而，与硬限制不同，在一定时间内可以超过软限制。该时间称为 宽限期。宽限期可以用秒、分钟、小时、天、周或月表示。

如果值为 0，则代表没有设定那个限制。在文本编辑器中，更改所需的限制。

例 17.4. 更改限制

例如：

```
Disk quotas for user testuser (uid 501):
Filesystem              blocks    soft    hard   inodes   soft   hard
/dev/VolGroup00/LogVol02  440436   500000   550000    37418      0      0
```

要验证是否为用户设定了配额，请使用以下命令：

```
# quota testuser
Disk quotas for user username (uid 501):
  Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
    /dev/sdb    1000*   1000    1000               0       0       0
```

17.1.5. 为每个组分配配额

配额可以针对单独组群进行分配。

前提条件

- 组群在设定组群配额前必须已经存在。

过程 17.4. 为每个组分配配额

1. 要设置组配额，请使用以下命令：

```
# edquota -g groupname
```

2. 要验证是否设定了组群配额，请使用以下命令：

```
# quota -g groupname
```

例 17.5. 为组分配配额

例如，要为 **devel** 组设置组配额，请使用以下命令：

```
# edquota -g devel
```

这个命令在文本编辑器中显示该组群的现有配额：

```
Disk quotas for group devel (gid 505):
Filesystem                blocks    soft     hard    inodes    soft    hard
/dev/VolGroup00/LogVol02  440400       0        0     37418       0       0
```

修改限制后保存文件。

要验证是否设置了组配额，请使用以下命令：

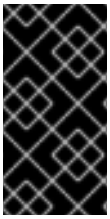
```
# quota -g devel
```

17.1.6. 为软限制设置宽限期

如果给定配额有软限制，您可以使用以下命令编辑宽限期（即可以超过软限制的时间）：

```
# edquota -t
```

此命令适用于针对用户或组的 **inode** 或块的配额。



重要

虽然其他 edquota 命令针对特定用户或组的配额进行操作，但 -t 选项在每个启用了配额的文件系统中运行。

17.2. 管理磁盘配额

如果实施了配额，它们主要需要一些维护，以便查看是否超过了配额，并确保配额是准确的。

如果用户重复超过配额，或者持续达到其软限制，系统管理员可根据用户类型以及影响其工作的空间数量而决定一些选择。管理员可以帮助用户决定如何使用较少的磁盘空间，或者增加用户的磁盘配额。

17.2.1. 启用和禁用

可以在不将其设置为 0 的情况下禁用配额。要关闭所有用户和组配额，请使用以下命令：

```
# quotaoff -vaug
```

如果未指定 -u 或 -g 选项，则仅禁用用户配额。如果只指定 -g，则只禁用组配额。-v 开关会在命令执行时显示详细状态信息。

要再次启用用户和组群配额，请使用以下命令：

```
# quotaon
```

要为所有文件系统启用用户和组群配额，请使用以下命令：

```
# quotaon -vaug
```

如果未指定 -u 或 -g 选项，则只启用用户配额。如果只指定 -g，则只启用组配额。

要为特定文件系统（如 /home ）启用配额，请使用以下命令：

```
# quotaon -vug /home
```

注意

XFS 并不总是需要 quotaon 命令，因为它是在挂载时自动执行的。如需更多信息，请参阅 man page quotaon (8)。

17.2.2. 报告磁盘配额

创建磁盘用量报告需要运行 repquota 工具。

例 17.6. repquota 命令的输出

例如，命令 repquota /home 生成此输出：

```
*** Report for user quotas on device /dev/mapper/VolGroup00-LogVol02
Block grace time: 7days; Inode grace time: 7days
  Block limits   File limits
User  used soft hard grace used soft hard grace
----------------------------------------------------------------------
root      --      36       0       0              4     0     0
kristin   --     540       0       0            125     0     0
testuser  --  440400  500000  550000          37418     0     0
```

要查看所有启用了配额的文件系统的磁盘使用情况报告（选项 -a），请使用命令：

```
# repquota -a
```

虽然报告易于阅读，但有以下几点需要解释：每个用户后面显示的 -- 是确定是否超过块或内节点限制的快速方法。如果超过了任何一个软限制，则 + 会出现在对应的 - 的位置，第一个 - 代表块限制，第二个代表内节点限制。

grace 列通常为空。如果超过了软限制，则该列包含的时间规格等同于宽限期中剩余的时间量。如果宽限期已过期，none 会出现在其位置上。

17.2.3. 使配额保持准确

当文件系统无法完全卸载时，例如因为系统崩溃，需要运行以下命令：

```
# quotacheck
```

但是，quotacheck 可以定期运行，即使系统没有崩溃。定期运行 quotacheck 的安全方法包括：

确保 quotacheck 在下次重启时运行

大多数系统的最佳方法

这个方法最适合定期重启的多用户系统。

使用以下命令将 shell 脚本保存到 /etc/cron.daily/ 或 /etc/cron.weekly/ 目录中，或者使用以下命令调度一个：

```
# crontab -e
```

crontab -e 命令包含 touch /forcequotacheck 命令。这会在根目录中创建一个空的 forcequotacheck 文件，系统初始化脚本会在引导时查找该文件。如果找到了，初始化脚本将运行 quotacheck。之后，初始化脚本会删除 /forcequotacheck 文件；因此，使用 cron 定期创建此文件，确保在下次重启时运行 quotacheck。

有关 cron 的更多信息，请参阅 man cron。

在单用户模式下运行 quotacheck

安全运行 quotacheck 的替代方法是将系统引导至单用户模式，以防止配额文件中出现数据崩溃并运行以下命令：

```
# quotaoff -vug /file_system
```

```
# quotacheck -vug /file_system
```

```
# quotaon -vug /file_system
```

在运行的系统上运行 quotacheck

如有必要，可以在没有用户登录的情况下在机器上运行 quotacheck，因此没有正在检查的文件系统上的打开文件。运行 quotacheck -vug file_system 命令；如果 quotacheck 无法以只读形式重新挂载给定的 file_system，此命令将失败。请注意，在检查后，文件系统将以读写形式重新挂载。

警告

不建议在以读写形式挂载的实时文件系统上运行 quotacheck，因为可能会损坏配额文件。

有关配置 cron 的更多信息，请参阅 man cron。

17.3. 磁盘配额参考

有关磁盘配额的详情，请参考以下命令的 手册页 ：

- quotacheck
- edquota
- repquota
- quota
- quotaon
- quotaoff

第 18 章 独立磁盘冗余阵列(RAID)

RAID 的基本理念是将多个小、成本低廉的磁盘驱动器合并到一个阵列中，以取得一个大而昂贵的驱动无法实现的性能或冗余目标。这个驱动器阵列在计算机上显示为一个逻辑存储单元或驱动器。

RAID 允许信息分布在多个磁盘上。RAID 使用 磁盘条带（RAID 0）、磁盘镜像（RAID 1）和 带奇偶校验的磁盘条带（RAID 5）来实现冗余、低延迟、增大带宽以及最大程度从硬盘崩溃恢复的能力。

RAID 通过将数据拆分为统一大小的块（通常为 256K 或 512k，但也接受其他值），从而将数据分布在阵列中的每个驱动器上。然后，每个块根据所使用的 RAID 级写到 RAID 阵列中的硬盘驱动器上。当读取数据时，过程会反过来，给人产生一种阵列中的多个驱动器实际上是一个大驱动器的错觉。

系统管理员以及管理大量数据的其他人将受益于使用 RAID 技术。部署 RAID 的主要原因包括：

- 加强速度
- 使用单一虚拟磁盘增加存储容量
- 尽可能减少磁盘失败的数据丢失

18.1. RAID 类型

有三种可能的 RAID 方法：固件 RAID、硬件 RAID 和软件 RAID。

固件 RAID

固件 RAID （也称为 ATARAID）是一种软件 RAID，可以使用基于固件的菜单配置 RAID 集。此类 RAID 使用的固件也会挂接到 BIOS 中，允许您从 RAID 集启动。不同的供应商使用不同的磁盘元数据格式来标记 RAID 集成员。Intel Matrix RAID 是固件 RAID 系统的一个很好的例子。

硬件 RAID

基于硬件的阵列独立于主机管理 RAID 子系统。它按每个 RAID 阵列一个磁盘的形式呈现给主机。

硬件 RAID 设备可能是系统内部或外部的，内部设备通常由专用的控制器卡组成，该卡通常包括对操作系统透明的 RAID 任务，以及外部设备通常通过 SCSI、光纤通道、iSCSI、InfiniBand 或其他高速度网络互连和向系统呈现逻辑卷。

RAID 控制器卡的功能与操作系统的 SCSI 控制器相同，用于处理所有实际驱动器间的通信。用户将驱动器插入 RAID 控制器（就像普通的 SCSI 控制器）中，然后将它们添加到 RAID 控制器配置中。操作系统将无法辨别它们的不同。

软件 RAID

软件 RAID 在内核磁盘（块设备）代码中实现各种 RAID 级。它提供最便宜的解决方案，如昂贵的磁盘控制器卡或热插拔机箱 [2] 不需要。软件 RAID 也适用于更便宜的 IDE 磁盘以及 SCSI 磁盘。随着当今 CPU 速度越来越快，软件 RAID 通常会优于硬件 RAID。

Linux 内核包含一个 多磁盘(MD)驱动程序，其允许 RAID 解决方案完全独立于硬件。基于软件的阵列的性能取决于服务器 CPU 的性能和负载。

Linux 软件 RAID 堆栈的主要特性：

- 多线程设计
- 在不同的 Linux 机器间移动磁盘阵列不需要重新构建数据
- 使用空闲系统资源进行后台阵列重构
- 热插拔驱动器的支持
- 自动 CPU 检测，以利用某些 CPU 功能，如流 SIMD 支持
- 自动更正阵列磁盘上坏扇区
- 定期检查 RAID 数据，以确保阵列健康

- 主动监控阵列，在发生重要事件时将电子邮件报警发送到指定的电子邮件地址

- 写意图位图通过允许内核准确了解磁盘的哪些部分需要重新同步，而不必重新同步整个阵列，从而大大提高了重新同步事件的速度

- 重新同步检查点，以便如果您在重新同步期间重新启动计算机，则在启动时重新同步会从其停止的地方开始，而不是从头开始

- 安装后更改阵列参数的能力。例如：当有新磁盘需要添加时，您可以将 4 磁盘 RAID5 阵列增加成 5 磁盘 RAID5 阵列。这种增加操作是实时的，不需要您在新阵列上重新安装。

18.2. RAID 级和线性支持

RAID 支持各种配置，包括 0、1、4、5、6、10 和 linear。这些 RAID 类型定义如下：

0 级

RAID 0，通常称为"条带化"，是一种面向性能的条带数据映射技术。这意味着写入阵列的数据被分成条，写入到阵列的成员磁盘上，从而以较低的成本提供较高的 I/O 性能，但不提供冗余。

许多 RAID 0 实现只会在成员设备上条带化数据，最大条带为阵列中最小设备的大小。这意味着，如果您有多个设备，它们的大小稍有不同，则每个设备都会被视为与最小驱动器的大小相同。因此，级别 0 阵列的一般存储容量等于，硬件 RAID 中容量最小的成员磁盘的容量，或软件 RAID 中的最小成员分区的容量，在乘以阵列中的磁盘或分区的数量。

1 级

RAID 1 或 "镜像" 比任何其它形式的 RAID 使用的时间都长。1 级通过将相同的数据写入阵列的每个成员磁盘来提供冗余，在每个磁盘上都保留一个"镜像"副本。因为其简单且数据高度可用，RAID 1 仍然被广泛使用。级别 1 需要两个或者多个磁盘，它提供了很好的数据可靠性，提高了需要读取的应用程序的性能，但是成本相对高。[3]

级别 1 阵列的存储容量等于硬件 RAID 中最小镜像硬盘或者软件 RAID 中最小镜像分区的容量相同。级别 1 所提供的冗余性是所有 RAID 级别中最高的，因为阵列只需要在有一个成员可以正常工作的情况下就可以提供数据。

4 级

4 级使用奇偶校验 [4] 集中在单个磁盘驱动器上来保护数据.因为 RAID 4 使用一个专门的偶校验磁盘，因此这个磁盘就会成为对 RAID 阵列的写入操作的一个固有的瓶颈。所以， RAID 4 较少被使用。因此， Anaconda 中并没有提供 RAID 4 这个选项。但是，如果真正需要，用户可以手动创建它。

硬件 RAID 4 的存储容量等于分区数量减一乘以最小成员分区的容量。RAID 4 阵列的性能总是非对称的，即读比写的性能要好。这是因为，写会在生成奇偶校验时消耗额外的 CPU 和主内存带宽，然后在将实际数据写入磁盘时也会消耗额外的总线带宽，因为您不仅写数据，而且还写奇偶校验。读取只需要读取数据而不是奇偶校验，除非该阵列处于降级状态。因此，在正常操作条件下，对于相同数量的数据传输，读会对驱动器和计算机总线产生较少的流量。

5 级

这是最常见的 RAID 类型。通过将奇偶校验分布在阵列的所有成员磁盘驱动器上，RAID 5 消除了 4 级中固有的写瓶颈。唯一性能瓶颈是奇偶校验计算过程本身。在使用现代 CPU 和软件 RAID 时，这通常不会成为瓶颈，因为现代 CPU 可能会非常快速地生成奇偶校验。然而，如果您的软件 RAID5 阵列中有大量成员设备，且在所有设备间有大量的数据进行传输时，就可能出现瓶颈。

与 4 级一样，5 级具有非对称性能，其读性能大大优于写性能。RAID 5 的存储容量的计算方法与级别 4 的计算方法是一样的。

级别 6

如果数据的冗余性和保护性比性能更重要，且无法接受 RAID 1 的空间利用率低的问题，则通常会选择使用级别 6。级别 6 使用一个复杂的奇偶校验方式，可以在阵列中出现任意两个磁盘失败的情况下进行恢复。因为使用的奇偶校验方式比较复杂，软件 RAID 设备会对 CPU 造成较大负担，同时对写操作造成更大的负担。因此，与级别 4 和 5 相比，级别 6 的性能不对称性更严重。

RAID 6 阵列的总容量与 RAID 5 和 4 类似，但您必须从额外奇偶校验存储空间的设备数中减去 2 个设备（而不是 1 个）。

级别 10

这个 RAID 级别将级别 0 的性能优势与级别 1 的冗余合并。它还有助于减少在有多于 2 个设备时，级别 1 阵列中的利用率低的问题。对于 10 级，可以创建一个 3 个驱动器阵列，来仅存储每块数据的 2 个副本，然后允许整个阵列的大小为最小设备的 1.5 倍，而不是只等于最小设备（这与 3 设备1 级阵列类似）。

创建 10 级阵列时可用的选项数量，以及为特定用例选择正确的选项的复杂性，使在安装过程中创建它们不切实际。可以使用命令行 mdadm 工具手动创建一个。有关选项及其各自性能权衡的更多信

息，请参阅 man md。

线性 RAID

线性 RAID 是创建更大的虚拟驱动器的一组驱动器。在线性 RAID 中，块会被从一个成员驱动器中按顺序分配，只有在第一个完全填充时才会进入下一个驱动器。这个分组方法不会提供性能优势，因为 I/O 操作不太可能在不同成员间同时进行。线性 RAID 也不提供冗余性并降低可靠性；如果任何一个成员驱动器失败，则整个阵列无法使用。该容量是所有成员磁盘的总量。

18.3. LINUX RAID 子系统

Linux 中的 RAID 由以下子系统组成：

Linux 硬件 RAID 控制器驱动程序

硬件 RAID 控制器在 Linux 中没有特定的 RAID 子系统。由于它们使用特殊的 RAID 芯片组，因此硬件 RAID 控制器有自己的驱动程序；这些驱动程序允许系统将 RAID 集作为常规磁盘来检测。

mdraid

mdraid 子系统设计为 Linux 的软件 RAID 解决方案；它也是 Linux 下软件 RAID 的首选解决方案。此子系统使用自己的元数据格式，通常称为原生 mdraid 元数据。

mdraid 还支持其他元数据格式，称为外部元数据。Red Hat Enterprise Linux 7 使用带有外部元数据的 mdraid 来访问 ISW / IMSM (Intel 固件 RAID)集。mdraid 集通过 mdadm 工具进行配置和控制。

dmraid

dmraid 工具用于各种固件 RAID 实现。dmraid 还支持 Intel 固件 RAID，但 Red Hat Enterprise Linux 7 使用 mdraid 访问 Intel 固件 RAID 集。

注意

从 Red Hat Enterprise Linux 7.5 发行版本起，dmraid 已被弃用。它将在以后的 Red Hat Enterprise Linux 主发行版本中删除。如需更多信息，请参阅 Red Hat Enterprise Linux 7.5 发行注记中的 已弃用的功能。

18.4. ANACONDA 安装程序中的 RAID 支持

Anaconda 安装程序自动检测系统上的任何硬件和固件 RAID 集，使其可用于安装。Anaconda 还支持使用 mdraid 的软件 RAID，并且可以识别现有的 mdraid 集。

Anaconda 提供了在安装过程中创建 RAID 集的工具，但这些工具只允许分区（而不是整个磁盘）成为新集合的成员。要将整个磁盘用于集合，请在上面创建一个跨整个磁盘的分区，并将该分区用作 RAID 设置成员。

当 root 文件系统使用 RAID 集时，Anaconda 会在引导装载程序配置中添加特殊的内核命令行选项，告知 initrd 在搜索根文件系统前要激活哪个 RAID 集。

有关在安装过程中配置 RAID 的详情，请查看 Red Hat Enterprise Linux 7 安装指南。

18.5. 安装后将根磁盘转换为 RAID1

如果您需要在安装 Red Hat Enterprise Linux 7 后将非 RAID 磁盘转换为 RAID1 镜像，请参阅以下红帽知识库文章：安装 Red Hat Enterprise Linux 7 后如何将我的根磁盘转换为 RAID1？

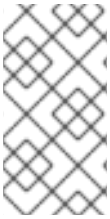
在 PowerPC (PPC) 构架中，执行以下步骤：

1. 将 PowerPC Reference Platform (PReP) 引导分区从 /dev/sda1 复制到 /dev/sdb1 ：

```
# dd if=/dev/sda1 of=/dev/sdb1
```

2. 在两个磁盘的第一个分区中更新 Prep 和 boot 标志：

```
$ parted /dev/sda set 1 prep on
$ parted /dev/sda set 1 boot on

$ parted /dev/sdb set 1 prep on
$ parted /dev/sdb set 1 boot on
```



注意

运行 grub2-install /dev/sda 命令无法在 PowerPC 机器上工作，并返回错误，但系统会按预期引导。

18.6. 配置 RAID 集

大多数 RAID 集在创建过程中配置，通常通过固件菜单或安装程序来配置。在某些情况下，您可能需要在安装系统后创建或修改 RAID 集，最好不需要重启计算机，进入到固件菜单来完成此操作。

某些硬件 RAID 控制器允许您动态配置 RAID 集，甚至在添加额外磁盘后定义全新的 RAID 集。这需要使用特定于驱动程序的工具，因为没有标准的 API。如需更多信息，请参阅硬件 RAID 控制器的驱动程序文档。

mdadm

mdadm 命令行工具用于管理 Linux 中的软件 RAID，如 mdraid。有关不同 mdadm 模式和选项的详情，请参考 man mdadm。man page 还包含创建、监控和组装软件 RAID 阵列等常见操作的有用示例。

dmraid

顾名思义，dmraid 用于管理设备映射器 RAID 集。dmraid 工具使用多个元数据格式处理程序查找 ATARAID 设备，每个处理程序都支持各种格式。如需支持的格式的完整列表，请运行 dmraid -l。

如前面 第 18.3 节 “Linux RAID 子系统” 所述，dmraid 工具无法在创建后配置 RAID 集。有关使用 dmraid 的更多信息，请参阅 man dmraid。

18.7. 创建高级 RAID 设备

在某些情况下，您可能想要在安装完成后在无法创建的阵列上安装操作系统。通常，这意味着在复杂的 RAID 设备上设置 /boot 或 root 文件系统阵列；在这种情况下，您可能需要使用 Anaconda 不支持的数组选项。要临时解决这个问题，请执行以下步骤：

过程 18.1. 创建高级 RAID 设备

1. 插入安装磁盘。

2. 在初始引导过程中，选择 Rescue Mode 而不是 Install 或 Upgrade。当系统完全引导到

Rescue mode 时，用户会看到一个命令行终端。

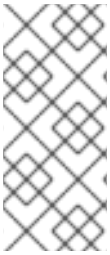
3.
在这个终端中，使用 parted 在目标硬盘上创建 RAID 分区。然后，使用 mdadm 使用任何以及所有可用的设置和选项从这些分区手动创建 raid 阵列。有关如何进行此操作的更多信息，请参阅 第 13 章 分区、man parted 和 man mdadm。

4.
创建阵列后，您可以选择在阵列上创建文件系统。

5.
重启计算机，此时选择 Install 或 Upgrade 以正常安装。当 Anaconda 搜索系统中的磁盘时，它将找到预先存在的 RAID 设备。

6.
当被问到如何使用系统中的磁盘时，请选择 Custom Layout 并单击 Next。在设备列表中，会列出预先存在的 MD RAID 设备。

7.
选择 RAID 设备，点 Edit 并配置其挂载点，并（可选）应使用的文件系统类型（如果您之前没有创建），然后点 Done。Anaconda 将对此预先存在的 RAID 设备执行安装，在 Rescue Mode 中创建时保留您选择的自定义选项。



注意

安装程序的有限 Rescue Mode 不包括 man page。man mdadm 和 man md 都包含创建自定义 RAID 阵列的有用信息，在整个临时解决方案中可能需要。因此，可以使用这些 man page 访问机器，或者在引导到救援模式并创建自定义阵列前打印它们。

---

[2]
热插拔机箱允许您移除硬盘驱动器，而无需关闭系统电源。

[3]
为了实现数据可靠性，需要向阵列中的所有磁盘写入相同的信息，所以 RAID 1 的成本会很高。与基于奇偶校验的其他级别（如级别 5）相比，空间的利用效率较低。然而，对空间利用率的牺牲提供了高性能：基于奇偶校验的 RAID 级别会消耗大量 CPU 资源以便获得奇偶校验，而 RAID 级别 1 只是一次向多个 RAID 成员中写入同样数据，其对 CPU 的消耗较小。因此，在使用软件 RAID 的系统中，或系统中有其他操作需要大量使用 CPU 资源时，RAID 1 可能会比使用基于奇偶校验的 RAID 级别的性能更好。

[4]
奇偶校验信息是根据阵列中剩余成员磁盘的内容计算出来的。然后当阵列中的一个磁盘失败时，这个信息就可以被用来重建数据。然后，在出现问题的磁盘被替换前，使用被重建的数据就可以满足 I/O 的请求。在磁盘被替换后，可以在上面重新生成数据。

第 19 章 使用 MOUNT 命令

在 Linux、UNIX 和类似操作系统上，不同分区和可移动设备（例如，CD、DVD 或者 USB 闪存驱动器）上的文件系统可以附加到目录树中的某个点（挂载点），然后再次分离。要附加或分离文件系统，请分别使用 mount 或 umount 命令。本章描述了这些命令的基本用途，以及一些高级主题，如移动挂载点或创建共享子树等。

19.1. 列出当前挂载的文件系统

要显示所有当前附加的文件系统，请使用以下命令，没有附加参数：

```
$ mount
```

此命令显示已知挂载点的列表。每行都提供有关设备名称、文件系统类型、挂载目录的重要信息，以及相关的挂载选项，格式如下：

```
device on directory type type (options)
```

Red Hat Enterprise Linux 6.1 也提供 findmnt 工具，它允许用户以类似于树形的形式列出挂载的文件系统。要显示所有当前附加的文件系统，请运行不带额外参数的 findmnt 命令：

```
$ findmnt
```

19.1.1. 指定文件系统类型

默认情况下，mount 命令的输出包括各种虚拟文件系统，如 sysfs 和 tmpfs。要只显示具有特定文件系统类型的设备，请提供 -t 选项：

```
$ mount -t type
```

同样，使用 findmnt 命令只显示具有特定文件系统的设备：

```
$ findmnt -t type
```

有关通用文件系统类型的列表，请参阅 表 19.1 “通用文件系统类型”。有关用法示例，请参阅 例 19.1 “列出当前挂载的 ext4 文件系统”。

例 19.1. 列出当前挂载的 ext4 文件系统

通常，/ 和 /boot 分区都被格式化为使用 ext4。要只显示使用此文件系统的挂载点，请使用以下命令：

```
$ mount -t ext4
/dev/sda2 on / type ext4 (rw)
/dev/sda1 on /boot type ext4 (rw)
```

要使用 findmnt 命令列出这些挂载点，请输入：

```
$ findmnt -t ext4
TARGET SOURCE    FSTYPE OPTIONS
/      /dev/sda2 ext4   rw,realtime,seclabel,barrier=1,data=ordered
/boot  /dev/sda1 ext4   rw,realtime,seclabel,barrier=1,data=ordered
```

19.2. 挂载文件系统

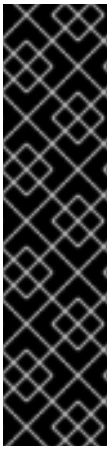
要附加某个文件系统，请使用以下格式的 mount 命令：

```
$ mount [option…] device directory
```

该设备 可通过以下方法识别：

- 块设备 的完整路径：例如 /dev/sda3
- 通用唯一标识符 (UUID)：例如 UUID=34795a28-ca6d-4fd8-a347-73671d0c19cb
- 卷标签 ：例如 LABEL=home

请注意，尽管挂载了文件系统，但无法访问 directory 中的原始内容。



重要：确保目录没有被使用

Linux 不阻止用户将文件系统挂载到已附加到其上的目录。要确定特定目录是否充当挂载点，请使用目录作为其参数运行 findmnt 工具，并验证退出代码：

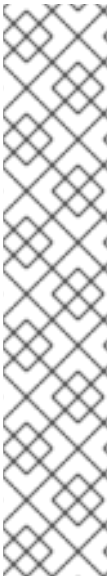
```
findmnt directory; echo $?
```

如果没有将文件系统附加到目录，则给定命令会返回 1。

当您运行不带所有所需信息的 mount 命令时，如果没有设备名称、目标目录或文件系统类型，挂载 会读取 /etc/fstab 文件的内容，以检查是否列出了给定的文件系统。/etc/fstab 文件包含设备名称列表、所选文件系统要挂载的目录，以及文件系统类型和挂载选项。因此，当挂载在 /etc/fstab 中指定的文件系统时，您可以选择以下选项之一：

```
mount [option...] directory
mount [option...] device
```

请注意，除非以 root 用户身份运行命令，否则挂载文件系统需要权限（请参阅 第 19.2.2 节 “指定挂载选项”）。



注： 确定 PARTICULAR 设备的 UUID 和标签

要确定 UUID 和-如果设备使用特定设备的标签，请使用以下格式的 blkid 命令：

```
blkid device
```

例如：要显示 /dev/sda3 的信息：

```
# blkid /dev/sda3
/dev/sda3: LABEL="home" UUID="34795a28-ca6d-4fd8-a347-73671d0c19cb"
TYPE="ext3"
```

19.2.1. 指定文件系统类型

在大多数情况下，mount 会自动检测文件系统。但是，有一些某些文件系统，如 NFS （网络文件系统）或 CIFS （通用互联网文件系统），它们不能被识别，需要手动指定。要指定文件系统类型，请使用以下格式的 mount 命令：

```
$ mount -t type device directory
```

表 19.1 “通用文件系统类型” 提供可与 mount 命令一起使用的常用文件系统类型列表。有关所有可用文件系统类型的完整列表，请参阅 “手册页文档”一节。

表 19.1. 通用文件系统类型

Type	描述
ext2	ext2 文件系统。
ext3	ext3 文件系统。
ext4	ext4 文件系统。
btrfs	btrfs 文件系统。
xfs	xfs 文件系统。
iso9660	ISO 9660 文件系统。它通常由光学介质（通常为 CD）使用。
nfs	NFS 文件系统。它通常用于通过网络访问文件。
nfs4	NFSv4 文件系统。它通常用于通过网络访问文件。
udf	UDF 文件系统。它通常由光学介质（通常为 DVD）使用。
vfat	FAT 文件系统。它通常用于运行 Windows 操作系统的机器，以及某些数字媒体，如 USB 闪存驱动器或软盘。

有关示例用法，请查看 例 19.2 “挂载 USB 闪存驱动器”。

例 19.2. 挂载 USB 闪存驱动器

较旧的 USB 闪存驱动器通常使用 FAT 文件系统。假设此类驱动器使用 /dev/sdc1 设备，并且 /media/flashdisk/ 目录存在，请以 root 用户身份在 shell 提示符后输入以下内容来将其挂载到此目录：

```
~]# mount -t vfat /dev/sdc1 /media/flashdisk
```

19.2.2. 指定挂载选项

要指定附加挂载选项，请使用以下格式的命令：

```
mount -o options device directory
```

在提供多个选项时，不要在逗号后插入空格，或者 mount 会将以下空格的值错误地解析为额外的参数。

表 19.2 “常用挂载选项” 提供了常用挂载选项的列表.有关所有可用选项的完整列表，请参阅 “手册页文档”一节 中的相关手册页。

表 19.2. 常用挂载选项

选项	描述
async	允许文件系统上的异步输入/输出操作。
auto	允许使用 mount -a 命令自动挂载文件系统。
默认值	为 async,auto,dev,exec,nouser,rw,suid 提供别名。
exec	允许在特定文件系统中执行二进制文件。
loop	将镜像挂载为 loop 设备。
noauto	默认行为不允许使用 mount -a 命令自动挂载文件系统。
noexec	不允许在特定文件系统中执行二进制文件。
nouser	不允许普通用户（即 root以外的用户）挂载和卸载文件系统。
remount	如果已经挂载文件系统，则会重新挂载文件系统。
ro	仅挂载文件系统以读取。
rw	挂载文件系统以进行读和写操作。
user	允许普通用户（即 root以外的用户）挂载和卸载文件系统。

有关示例用法，请查看 例 19.3 “挂载 ISO 镜像”。

例 19.3. 挂载 ISO 镜像

可以使用 loop 设备挂载 ISO 镜像（一般而言是磁盘镜像）。假设 Fedora 14 安装磁盘的 ISO 镜像存在于当前工作目录中，并且 /media/cdrom/ 目录存在，请运行以下命令将镜像挂载到这个目录中：

```
# mount -o ro,loop Fedora-14-x86_64-Live-Desktop.iso /media/cdrom
```

请注意，ISO 9660 设计为只读文件系统。

19.2.3. 共享挂载

有时，某些系统管理任务需要从目录树中的多个位置（例如，准备 chroot 环境时）访问同一文件系统。这可以实现，而且 Linux 允许您根据需要将同一文件系统挂载到多个目录。另外，mount 命令实现了 **--bind** 选项，它提供复制某些挂载的方法。其用法如下：

```
$ mount --bind old_directory new_directory
```

虽然此命令允许用户从两个位置访问文件系统，但它不适用于原始目录中挂载的文件系统。要包括这些挂载，请使用以下命令：

```
$ mount --rbind old_directory new_directory
```

此外，为了提供尽可能多的灵活性，Red Hat Enterprise Linux 7 实现了称为 共享子树 的功能。此功能允许使用以下四种挂载类型：

共享挂载

共享挂载允许创建给定挂载点的精确副本。当挂载点标记为共享挂载时，原始挂载点中的任何挂载都会反映在其中，反之亦然。要将挂载点类型改为共享挂载，请在 shell 提示符下输入以下内容：

```
$ mount --make-shared mount_point
```

或者，要更改所选挂载点的挂载类型，以及其下的所有挂载点：

```
$ mount --make-rshared mount_point
```

有关示例用法，请查看 例 19.4 “创建共享挂载点”。

例 19.4. 创建共享挂载点

其他文件系统通常挂载在两个位置：可移动介质的 /media/ 目录，以及用于临时挂载的文件系统的 /mnt/ 目录。通过使用共享挂载，您可以使这两个目录共享相同的内容。要做到这一点，以 root 用户身份将 /media/ 目录标记为 shared ：

```
# mount --bind /media /media
# mount --make-shared /media
```

使用以下命令在 /mnt/ 中创建副本：

```
# mount --bind /media /mnt
```

现在，可以验证 /media/ 中的挂载是否也出现在 /mnt/ 中。例如，如果 CD-ROM 驱动器包含非空介质，并且 /media/cdrom/ 目录存在，请运行以下命令：

```
# mount /dev/cdrom /media/cdrom
# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
# ls /mnt/cdrom
EFI  GPL  isolinux  LiveOS
```

同样，可以验证 /mnt/ 目录中挂载的任何文件系统是否反映在 /media/ 中。例如，如果插入了使用 /dev/sdc1 设备的非空 USB 闪存驱动器，并且存在 /mnt/flashdisk/ 目录，请输入：

```
# # mount /dev/sdc1 /mnt/flashdisk
# ls /media/flashdisk
en-US  publican.cfg
# ls /mnt/flashdisk
en-US  publican.cfg
```

从属挂载

从属挂载允许创建给定挂载点的有限副本。当挂载点标记为从属挂载时，原始挂载点中的任何挂载都会反映在其中，但从属挂载中的挂载不会反映在其原始挂载点中。要将挂载点类型改为从属挂载，在 shell 提示符下输入以下内容：

```
mount --make-slave mount_point
```

或者，要更改所选挂载点的挂载类型，以及其下的所有挂载点，请输入：

```
mount --make-rslave mount_point
```

有关示例用法，请查看 例 19.5 “创建从属挂载点”。

例 19.5. 创建从属挂载点

本例演示了如何使 /media/ 目录的内容也出现在 /mnt/ 中，但 /mnt/ 目录中的任何挂载都不会反映在 /media/ 目录中。以 root 用户身份，首先将 /media/ 目录标记为共享：

```
~]# mount --bind /media /media
~]# mount --make-shared /media
```

然后，在 /mnt/ 中创建副本，但将其标记为"slave"：

```
~]# mount --bind /media /mnt
~]# mount --make-slave /mnt
```

现在，验证 /media/ 中的挂载是否也出现在 /mnt/ 中。例如，如果 CD-ROM 驱动器包含非空介质，并且 /media/cdrom/ 目录存在，请运行以下命令：

```
~]# mount /dev/cdrom /media/cdrom
~]# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
~]# ls /mnt/cdrom
EFI  GPL  isolinux  LiveOS
```

同时还要验证 /mnt/ 目录中挂载的文件系统没有反映在 /media/ 中。例如，如果插入了使用 /dev/sdc1 设备的非空 USB 闪存驱动器，并且存在 /mnt/flashdisk/ 目录，请输入：

```
~]# mount /dev/sdc1 /mnt/flashdisk
~]# ls /media/flashdisk
~]# ls /mnt/flashdisk
en-US  publican.cfg
```

私有挂载

私有挂载是默认挂载类型，与共享或从属挂载不同，它不接收或转发任何传播事件。要将挂载点

显式标记为私有挂载，在 shell 提示符下输入以下内容：

```
mount --make-private mount_point
```

另外，也可以更改所选挂载点的挂载类型，以及其下的所有挂载点：

```
mount --make-rprivate mount_point
```

有关示例用法，请查看 例 19.6 “创建私有挂载点”。

例 19.6. 创建私有挂载点

考虑 例 19.4 “创建共享挂载点” 中的场景，假设之前已以 root 用户身份使用以下命令创建了共享挂载点：

```
~]# mount --bind /media /media
~]# mount --make-shared /media
~]# mount --bind /media /mnt
```

要将 /mnt/ 目录标记为私有，请输入：

```
~]# mount --make-private /mnt
```

现在，可以验证 /media/ 中的挂载没有出现在 /mnt/ 中。例如，如果 CD-ROM 驱动器包含非空介质，并且 /media/cdrom/ 目录存在，请运行以下命令：

```
~]# mount /dev/cdrom /media/cdrom
~]# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
~]# ls /mnt/cdrom
~]#
```

也可以验证 /mnt/ 目录中挂载的文件系统没有反映在 /media/ 中。例如，如果插入了使用 /dev/sdc1 设备的非空 USB 闪存驱动器，并且存在 /mnt/flashdisk/ 目录，请输入：

```
~]# mount /dev/sdc1 /mnt/flashdisk
~]# ls /media/flashdisk
~]# ls /mnt/flashdisk
en-US  publican.cfg
```

不可绑定挂载

为了防止指定挂载点重复，需要使用不可绑定挂载。要将挂载点类型更改为不可绑定挂载，在 shell 提示符下输入以下内容：

```
mount --make-unbindable mount_point
```

另外，也可以更改所选挂载点的挂载类型，以及其下的所有挂载点：

```
mount --make-runbindable mount_point
```

有关示例用法，请查看 例 19.7 “创建不可绑定挂载点”。

例 19.7. 创建不可绑定挂载点

要防止 /media/ 目录被共享，以 root 用户身份：

```
# mount --bind /media /media
# mount --make-unbindable /media
```

这样，任何后续尝试重复此挂载都会失败，并显示错误：

```
# mount --bind /media /mnt
mount: wrong fs type, bad option, bad superblock on /media,
missing codepage or helper program, or other error
In some cases useful info is found in syslog - try
dmesg | tail  or so
```

19.2.4. 移动挂载点

要更改挂载文件系统的目录，请使用以下命令：

```
# mount --move old_directory new_directory
```

有关示例用法，请查看 例 19.8 “移动现有的 NFS 挂载点”。

-

例 19.8. 移动现有的 NFS 挂载点

NFS 存储包含用户目录，并且已挂载到 /mnt/userdirs/ 中。以 root 用户身份，使用以下命令将此挂载点移到 /home ：

```
# mount --move /mnt/userdirs /home
```

要验证挂载点是否已移动，请列出两个目录中的内容：

```
# ls /mnt/userdirs
# ls /home
jill  joe
```

19.2.5. 为 root设置只读权限

有时，您需要使用只读权限挂载 root 文件系统。示例用例包括在系统意外断电后增强安全性或确保数据完整性。

19.2.5.1. 将 root 配置为在引导时使用只读权限挂载

1. 在 /etc/sysconfig/readonly-root 文件中，将 READONLY 更改为 yes ：

```
# Set to 'yes' to mount the file systems as read-only.
READONLY=yes
[output truncated]
```

2. 在 /etc/fstab 文件中的 root 条目(/)中将默认更改为 ro ：

```
/dev/mapper/luks-c376919e... / ext4 ro,x-systemd.device-timeout=0 1 1
```

3. 将 ro 添加到 /etc/default/grub 文件中的 GRUB_CMDLINE_LINUX 指令中，并确保它不包含 rw ：

```
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb
quiet ro"
```

4. 重新创建 GRUB2 配置文件：

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

5. 如果您需要在 tmpfs 文件系统中添加需要挂载具有写权限的文件和目录，请在 /etc/rwtab.d/ 目录中创建一个文本文件，并将配置放在其中。例如，要挂载具有写入权限的 /etc/example/file，请将此行添加到 /etc/rwtab.d/示例 文件中：

```
files /etc/example/file
```

重要

对 tmpfs 中的文件和目录所做的更改不会在启动后保留。

有关此步骤的详情，请查看 第 19.2.5.3 节 “保留写权限的文件和目录”。

6. 重启系统。

19.2.5.2. 重新挂载 root Instantly

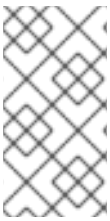
如果 root (/)在系统引导时以只读权限挂载，您可以使用写入权限重新挂载它：

```
# mount -o remount,rw /
```

当 / 错误地挂载了只读权限时，这特别有用。

要再次以只读权限重新挂载 /，请运行：

```
# mount -o remount,ro /
```



注意

此命令使用只读权限挂载 整个 /。更好的方法是将特定文件和目录的写入权限复制到 RAM 中，如 第 19.2.5.1 节 “将 root 配置为在引导时使用只读权限挂载” 所述。

19.2.5.3. 保留写权限的文件和目录

要使系统正常工作，一些文件和目录需要保留写权限。使用只读模式的 root 时，它们将挂载到

tmpfs 临时文件系统的 RAM 中。这些文件和目录的默认集合是从 /etc/rwtab 文件中读取的，该文件包含：

```
dirs /var/cache/man
dirs /var/gdm
[output truncated]
empty /tmp
empty /var/cache/foomatic
[output truncated]
files /etc/adjtime
files /etc/ntp.conf
[output truncated]
```

/etc/rwtab 文件中的条目遵循以下格式：

```
how the file or directory is copied to tmpfs        path to the file or directory
```

文件或目录可以通过以下三种方式复制到 tmpfs ：

- 空 路径 ：空路径复制到 tmpfs。示例： 空 /tmp
- 目录路径 ：目录树被复制到 tmpfs，空。示例： dirs /var/run
- 文件路径 ：将文件或目录树复制到 tmpfs。示例： 文件 /etc/resolv.conf

在向 /etc/rwtab.d/ 添加自定义路径时，也适用相同的格式。

19.3. 卸载文件系统

要分离之前挂载的文件系统，请使用以下 umount 命令的变体之一：

```
$ umount directory
$ umount device
```

请注意，除非以 root 身份登录时执行此操作，否则必须有正确的权限才能卸载文件系统。如需更多信息，请参阅 第 19.2.2 节 “指定挂载选项”。有关示例用法，请查看 例 19.9 “卸载 CD”。

重要：确保目录没有被使用

当使用文件系统时（例如，当某个进程正在读取此文件系统上的文件时，或者内核使用该文件时），运行 umount 命令会失败并显示错误。要确定哪个进程正在访问文件系统，请使用以下格式的 fuser 命令：

```
$ fuser -m directory
```

例如，列出正在访问挂载到 /media/cdrom/ 目录的文件系统的进程：

```
$ fuser -m /media/cdrom
/media/cdrom:        1793  2013  2022  2435 10532c 10672c
```

例 19.9. 卸载 CD

要卸载之前挂载到 /media/cdrom/ 目录的 CD，请使用以下命令：

```
$ umount /media/cdrom
```

19.4. MOUNT 命令参考

以下资源提供有关该主题的深入文档：

手册页文档

- man 8 mount ： mount 命令的手册页，提供了有关其使用的完整文档。
- man 8 umount: umount 命令的手册页，提供了有关其使用的完整文档。
- man 8 findmnt ： findmnt 命令的手册页，提供了有关其使用的完整文档。
- man 5 fstab ：手册页提供了 /etc/fstab 文件格式的详细描述。

有用的网站

- *共享子树* - *涵盖了共享子树概念的 LWN 文章。*

第 20 章 VOLUME_KEY FUNCTION

volume_key 功能提供两个工具：libvolume_key 和 volume_key. libvolume_key，是用于操作存储卷加密密钥的库，并将它们与卷分开存储。volume_key 是一个关联的命令行工具，用于提取密钥和密码短语，以便恢复对加密硬盘驱动器的访问。

这对于员工突然离开后，主要用户忘记了他们的密钥和密码，或者由于硬件或软件故障破坏了加密卷的头后提取数据时，非常有用。在公司设置中，IT 帮助台可以使用 volume_key 在将计算机移交至最终用户之前备份加密密钥。

目前，volume_key 只支持 LUKS 卷加密格式。

注意

volume_key 不包含在 Red Hat Enterprise Linux 7 服务器标准安装中。有关安装的详情，请参考：http://fedoraproject.org/wiki/Disk_encryption_key_escrow_use_cases

20.1. VOLUME_KEY 命令

volume_key 的格式是：

```
volume_key [OPTION]... OPERAND
```

volume_key 的操作对象和操作模式是通过指定以下选项之一来确定的：

--save

此命令需要操作对象 volume [packet]。如果提供了 数据包，则 volume_key 将从中提取密钥和密码短语。如果没有提供 数据包，则 volume_key 将从卷中提取密钥和密码短语，并在需要时提示用户输入。然后这些密钥和密码短语将存储在一个或多个输出数据包中。

--restore

此命令需要操作对象 volume packet。然后，它打开 volume，并使用 packet 中的密钥和密码短语使 volume 可再次访问，并在需要时提示用户，例如允许用户输入新的密语。

--setup-volume

此命令需要操作对象 volume packet name。然后，它打开 volume，并使用 packet 中的密钥和密码短语来设置 volume，以便将解密的数据用作 name。

Name 是 dm-crypt 卷的名称。此操作使解密的卷作为 /dev/mapper/名称 提供。

例如，此操作不会通过添加新密码短语来永久更改 volume。用户可以访问和修改解密的卷，从而可以在过程中修改 volume。

--reencrypt、--secrets 和 --dump

这三个命令使用不同的输出方法执行类似的功能。它们每个都需要操作对象 packet，并且每个都打开 packet，根据需要对其进行解密。然后 --reencrypt 将信息存储在一个或多个新输出数据包中。--secrets 输出包含在 数据包 中的密钥和密码短语。--dump 输出数据包的内容，但默认情况下不会输出密钥和密码短语。这可以通过在命令中附加 --with-secrets 来更改。也可以使用 --unencrypted 命令仅转储数据包的未加密部分（若有）。这不需要任何密语或私钥访问。

每个选项均可附加以下选项：

-o,--output packet

此命令将默认密钥或密语写入 packet。默认密钥或密语取决于卷格式。确保它不太可能过期，并且允许 --restore 恢复对卷的访问。

--output-format format

此命令对所有输出数据包使用指定的 format。目前，format 可以是以下之一：

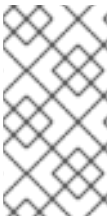
- 非对称 ：使用 CMS 来加密整个数据包，并且需要证书
- asymmetric_wrap_secret_only: 仅打包 secret 或密钥和密码短语，并且需要证书
- 密码短语 ：使用 GPG 来加密整个数据包，并且需要密码短语

--create-random-passphrase packet

此命令生成随机的字母数字密码短语，将它添加到 volume 中（而不影响其他密语），然后将此随机密语存储到 packet 中。

20.2. 将 VOLUME_KEY 用作单个用户

作为单个用户，可以按照以下流程使用 volume_key 来保存加密密钥。



注意

对于此文件中的所有示例，/path/to/volume 是 LUKS 设备，而不是包含在其中的明文设备。**blkid -s type /path/to/volume** 应该报告 **type="crypto_LUKS"**。

过程 20.1. 使用 volume_key Stand-alone

1. 运行：

```
volume_key --save /path/to/volume -o escrow-packet
```

然后，系统将出现提示，要求使用托管包密语来保护密钥。

2. 保存生成的 escrow-packet 文件，确保不会忘记密码短语。

如果忘记了卷密语，请使用保存的托管包恢复对数据的访问。

过程 20.2. 使用 Escrow Packet 将数据访问恢复到数据

1. 在可以运行 volume_key 且托管数据包可用的环境中引导系统（例如，救援模式）。

2. 运行：

```
volume_key --restore /path/to/volume escrow-packet
```

将显示一个提示，提示创建托管数据包时使用的托管数据包密码短语，以及卷的新密语。

3.

使用所选的密码短语挂载卷。

要在加密卷的 LUKS 标头中释放密码短语插槽，请使用 cryptsetup luksKillSlot 命令删除旧的、忘记的密码短语。

20.3. 在大型机构中使用 VOLUME_KEY

在大型组织中，使用每位系统管理员知道的单个密码并跟踪各个系统的单独密码是不切实际的，而且存在安全风险。为克服这一点，volume_key 可以使用非对称加密加密来最小化知道访问任何计算机上加密数据所需的密码的人员数量。

本节将涵盖在保存加密密钥前需要准备的流程、如何保存加密密钥、恢复对卷的访问以及设置紧急密码短语。

20.3.1. 准备保存加密密钥

为了开始保存加密密钥，需要进行一些准备。

过程 20.3. 准备

1.

创建 X509 证书/专用对。

2.

指定信任的用户，这些用户可信，不会泄露私钥。这些用户将能够解密托管数据包。

3.

选择将使用哪些系统解密托管数据包。在这些系统上，建立包含私钥的 NSS 数据库。

如果没有在 NSS 数据库中创建私钥，请按照以下步骤操作：

- 将证书和私钥存储在 PKCScriu 文件中。

- 运行：

```
certutil -d /the/nss/directory -N
```

此时，可以选择 NSS 数据库密码。每个 NSS 数据库都有不同的密码，因此如果每个用户使用单独的 NSS 数据库，则指定的用户无需共享单个密码。

- 运行：

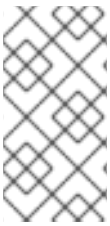
```
pk12util -d /the/nss/directory -i the-pkcs12-file
```

4. 将证书分发给安装系统或将密钥保存在现有系统上的任何人。

5. 对于保存的私钥，准备允许按机器和卷查找它们的存储。例如，这可以是一个简单的目录，每台机器有一个子目录，或者用于其他系统管理任务的数据库。

20.3.2. 保存加密密钥

完成所需的准备（请参阅 第 20.3.1 节 “准备保存加密密钥”）后，现在可以使用以下流程保存加密密钥。



注意

对于此文件中的所有示例，/path/to/volume 是 LUKS 设备，而不是包含的明文设备； blkid -s 类型 /path/to/volume 应报告 type="crypto_LUKS"。

过程 20.4. 保存加密密钥

1. 运行：

```
volume_key --save /path/to/volume -c /path/to/cert escrow-packet
```

2. 将生成的 escrow-packet 文件保存到准备好的存储中，将其与系统和卷相关联。

这些步骤可以手动执行，也可以作为系统安装的一部分编写成脚本。

20.3.3. 恢复对卷的访问

保存加密密钥（请参阅 第 20.3.1 节 “准备保存加密密钥” 和 第 20.3.2 节 “保存加密密钥”）后，可以根据需要恢复对驱动程序的访问。

过程 20.5. 恢复对卷的访问

1. 从数据包存储中获取卷的托管数据包，并将其发送到指定用户之一进行解密。

2. 指定的用户运行：

```
volume_key --reencrypt -d /the/nss/directory escrow-packet-in -o escrow-packet-out
```

提供 NSS 数据库密码后，指定的用户选择加密 escrow-packet-out 的密码短语。此密语每次都可能会有所不同，且只在加密密钥从指定用户移至目标系统时保护加密密钥。

3. 从指定的用户获取 escrow-packet-out 文件和密码短语。

4. 在可以运行 volume_key 且可以使用 escrow-packet-out 文件的环境中引导目标系统，比如在救援模式下。

5. 运行：

```
volume_key --restore /path/to/volume escrow-packet-out
```

将出现一个提示，提示指定用户所选择的数据包密码，以及卷的新密码。

6. 使用所选的卷密语挂载卷。

可以使用 cryptsetup luksKillSlot 删除忘记的旧密码短语，例如，在加密卷的 LUKS 标头中释放密语插槽。这可以通过 cryptsetup luksKillSlot device key-slot 命令完成。如需更多信息和示例，请参阅 cryptsetup --help。

20.3.4. 设置紧急密码

在某些情况下（如业务旅行），系统管理员直接操作受影响的系统是不切实际的，但用户仍需要访问这些系统的数据。在这种情况下，volume_key 可以使用密码短语和加密密钥。

在系统安装过程中，运行：

```
volume_key --save /path/to/volume -c /path/to/ert --create-random-passphrase passphrase-packet
```

这会生成随机密码短语，将其添加到指定的卷中，并将其保存到 passphrase-packet 中。也可以组合 --create-random-passphrase 和 -o 选项，以同时生成这两个数据包。

如果用户忘记了密码，指定用户运行：

```
volume_key --secrets -d /your/nss/directory passphrase-packet
```

这将显示随机密码短语。向最终用户提供此密码短语。

20.4. VOLUME_KEY 参考

有关 volume_key 的更多信息，请参阅：

- 在位于 /usr/share/doc/volume_keyMAP/README的 readme 文件中
- 在使用 man volume_key 的 volume_key 手册页上
- 在线上 http://fedoraproject.org/wiki/Disk_encryption_key_escrow_use_cases

第 21 章 固态磁盘部署指南

固态磁盘 (SSD)是使用 NAND 闪存芯片来持久存储数据的存储设备。这样可将其与之前生成的磁盘外设置，后者将数据存储在旋转、大体上手中。在 SSD 中，完整逻辑块地址(LBA)范围的数据访问时间是恒定的；而对于使用旋转介质的旧磁盘，跨越大型地址范围的访问模式会导致成本。因此，SSD 设备具有更好的延迟和吞吐量。

性能降低，因为已用块的数量接近磁盘容量。性能影响的程度因供应商而异。但是，所有设备都出现某种降级。

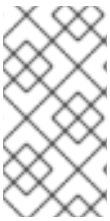
要解决降级的问题，主机系统（如 Linux 内核）可以使用丢弃的请求来告知存储给定范围的块不再使用。SSD 可以使用此信息在内部释放空间，使用空闲块进行磨损均衡。只有存储宣传支持其存储协议（ATA 或 SCSI）时，才会发出丢弃。使用特定于存储协议的协商 discard 命令（用于 ATA 的TRIM 命令和带有 UNMAP 集的 WRITE SAME 或 SCSI 的 UNMAP 命令）向存储发出丢弃请求。

当以下点为 true 时启用 丢弃 支持最有用：

- 可用空间仍可在文件系统中可用。
- 底层存储设备上的大多数逻辑块已被写入。

有关 TRIM 的更多信息，请参阅 数据集管理 T13 规范。

有关 UNMAP 的更多信息，请参阅 SCSI 块命令 3 T10 规范 的 4.7.3.4 部分。



注意

不是市场中的所有固态设备 都支持。要确定您的固态设备是否支持 丢弃，请检查 /sys/block/sda/queue/discard_granularity，这是设备的内部分配单元的大小。

部署注意事项

由于 SSD 的内部布局和操作，最好在内部 擦除块边界 上对设备进行分区。如果 SSD 导出拓扑信息，Red Hat Enterprise Linux 7 中的分区工具会选择 sane 默认值。但是，如果设备 没有 导出拓扑信息，红帽建议在 1MB 边界下创建第一个分区。

SSD 有不同的 TRIM 机制，具体取决于供应商选择。早期版本的磁盘通过在 read 命令后影响数据泄漏来提高性能。

以下是 TRIM 机制的类型：

- 非确定的 TRIM
- 确定性 TRIM (DRAT)
- TRIM (RZAT)后确定性读零

在 TRIM 返回不同或相同数据后，前两种 TRIM 机制可能会导致数据泄漏，因为 read 命令会泄漏到 LBA。RZAT 在 read 命令后返回零，红帽建议这种 TRIM 机制以避免数据泄露。它只在 SSD 中受到影响。选择支持 RZAT 机制的磁盘。

使用的 TRIM 机制类型取决于硬件实现。要在 ATA 上查找 TRIM 机制的类型，请使用 hdparm 命令。请参阅以下示例来查找 TRIM 机制的类型：

```
# hdparm -I /dev/sda | grep TRIM
Data Set Management TRIM supported (limit 8 block)
Deterministic read data after TRIM
```

如需更多信息，请参阅 man hdparm。

LVM 使用的逻辑卷管理器(LVM)、设备映射器(DM)目标和 MD （软件 raid）目标。不支持丢弃的唯一 DM 目标是 dm-snapshot、dm-crypt 和 dm-raid45。在 Red Hat Enterprise Linux 6.1 中添加了对 dm-mirror 的丢弃支持，在 7.0 MD 中支持丢弃。

如果 SSD 无法正确处理 丢弃，则使用 RAID 5 与 SSD 的性能会较低。您可以在 raid456.conf 文件中设置 discard，或者在 GRUB2 配置中设置 discard。具体步骤请查看以下步骤。

过程 21.1. 在 raid456.conf 中设置 discard

devices_handle_discard_safely 模块参数在 raid456 模块中设置。在 raid456.conf 文件中启用丢弃：

1. 验证您的硬件支持丢弃：

```
# cat /sys/block/disk-name/queue/discard_zeroes_data
```

如果返回的值是 1，则支持丢弃。如果命令返回 0， RAID 代码必须归零磁盘，这需要更长的时间。

2. 创建 /etc/modprobe.d/raid456.conf 文件，并包含以下行：

```
options raid456 devices_handle_discard_safely=Y
```

3. 使用 dracut -f 命令重建初始 ramdisk (initrd)。

4. 重启系统以使更改生效。

过程 21.2. 在 GRUB2 配置中设置 discard

devices_handle_discard_safely 模块参数在 raid456 模块中设置。在 GRUB2 配置中启用丢弃：

1. 验证您的硬件支持丢弃：

```
# cat /sys/block/disk-name/queue/discard_zeroes_data
```

如果返回的值是 1，则支持丢弃。如果命令返回 0， RAID 代码必须归零磁盘，这需要更长的时间。

2. 在 /etc/default/grub 文件中添加以下行：

```
raid456.devices_handle_discard_safely=Y
```

3. 在使用 BIOS 固件以及带有 UEFI 的系统中，GRUB2 配置文件的位置有所不同。使用以下命令之一重新创建 GRUB2 配置文件。

- 在带有 BIOS 固件的系统中，使用：

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- 在具有 UEFI 固件的系统中，使用：

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

4. 重启系统以使更改生效。

注意

在 Red Hat Enterprise Linux 7 中，仅 ext4 和 XFS 文件系统完全支持 discard。

在 Red Hat Enterprise Linux 6.3 及更早版本中，只有 ext4 文件系统完全支持 discard。从 Red Hat Enterprise Linux 6.4 开始，ext4 和 XFS 文件系统都完全支持 discard。要在设备上启用 discard 命令，请使用 mount 命令的 discard 选项。例如：要将 /dev/sda2 挂载到启用了 discard 的 /mnt 中，请使用：

```
# mount -t ext4 -o discard /dev/sda2 /mnt
```

默认情况下，ext4 不会发出 discard 命令，主要避免了可能无法正确实现丢弃的设备出现问题。Linux swap 代码为启用丢弃的设备发出丢弃命令，没有控制此行为的选项。

性能调优注意事项

有关固态磁盘性能调优注意事项的详情，请查看 Red Hat Enterprise Linux 7 性能调节指南 中的 Solid-State 磁盘 部分。

第 22 章 写屏障

中获取的内容

写屏障 是一种内核机制，用于确保正确写入文件系统元数据，并在持久存储上排序，即使具有易失性写缓存的存储设备断电。启用写屏障的文件系统可确保通过 fsync （） 传输的数据在电源丢失过程中持久保留。

启用写屏障会给某些应用程序带来大量性能损失。具体来说，大量使用 fsync （） 或创建和删除许多小文件的应用程序可能会运行得非常慢。

22.1. 写屏障的重要性

文件系统安全地更新元数据，确保一致性。Journalled 文件系统将元数据更新捆绑到事务中，并以以下方式将它们发送到持久性存储：

1. 文件系统将事务正文发送到存储设备。
2. 文件系统发送提交块。
3. 如果事务及其相应的提交块写入了磁盘，则文件系统假定事务将在任何电源失败后都能生存。

但是，对于带有额外缓存的存储设备而言，电源失败期间文件系统的完整性会变得更加复杂。存储目标设备（如本地 S-ATA 或 SAS 驱动器）可能具有大小为 32MB 到 64MB 的写缓存（与现代驱动器）。硬件 RAID 控制器通常包含内部写缓存。此外，像 NetApp、IBM、Hitachi 和 EMC（其他人）的高端阵列也具有大型缓存。

当数据处于缓存中时，带有写缓存的存储设备会报告 I/O 为"完成"；如果缓存断电，也会丢失数据。更糟糕的情况是，当缓存降级到持久存储时，它可能会更改原始元数据排序。发生这种情况时，提交块可能会出现在磁盘上，而没有完整的、相关的事务。因此，日志可能会在断电后的恢复期间将这些未初始化的事务块重新运行到文件系统中，这会导致数据不一致和损坏。

写障碍是如何工作的

在 Linux 内核中，写障碍是通过 I/O 前后的存储写缓存刷新实现的，这是 顺序关键的。写事务后，将刷新存储缓存，提交块被写入，然后再次刷新缓存。这样可确保：

- 磁盘包含所有数据。
- 未发生重新排序。

启用障碍后，fsync （） 调用也会发出存储缓存清除。这样可保证文件数据在磁盘上持久保留，即使 fsync （） 返回后不久发生断电。

22.2. 启用和禁用写障碍

为了降低掉电期间数据损坏的风险，一些存储设备使用电池支持的写缓存。通常，高端阵列和一些硬件控制器使用电池支持的写缓存。但是，由于缓存的剧剧性对内核不可见，因此 Red Hat Enterprise Linux 7 在所有支持的日志记录文件系统中默认启用写障碍。

注意

写入缓存旨在提高 I/O 性能。但是，启用写屏障意味着持续清除这些缓存，这可以显著降低性能。

对于具有非易失性、支持电池支持的写缓存的设备以及禁用写缓存的设备，您可以在挂载时使用 mount 的 -o nobarrier 选项安全地禁用写障碍。但是，有些设备不支持写屏障，此类设备会将错误消息记录到 /var/log/messages。如需更多信息，请参阅 表 22.1 “每个文件系统的写障碍错误消息”。

表 22.1. 每个文件系统的写障碍错误消息

文件系统	错误消息
ext3/ext4	JBD: barrier-based sync failed on device - disabling barriers
XFS	Filesystem device - Disabling barriers, trial barrier write failed
btrfs	btrfs: disabling barriers on dev device

22.3. 写障碍注意事项

有些系统配置不需要写屏障来保护数据。在大多数情况下，其他方法比写屏障更可取，因为启用写障碍会导致严重的性能损失。

禁用写缓存

避免数据完整性问题的另一种方法是确保在电源失败时不丢失任何写缓存数据。在可能的情况下，配置此功能的最佳方法是禁用写缓存。在使用一个或多个 SATA 驱动器的简单服务器或桌面上（关闭本地 SATA 控制器 Intel AHCI 部分），您可以使用以下命令在目标 SATA 驱动器中禁用写缓存：

```
# hdparm -W0 /device/
```

电池支持的写缓存

当系统使用带有电池支持的写缓存的硬件 RAID 控制器时，写障碍也是不必要的。如果系统带有此类控制器，并且其组件驱动器禁用了写缓存，则控制器将充当直写缓存，这会告知内核写缓存数据在电源丢失后仍存在。

大多数控制器使用特定于供应商的工具来查询和操作目标驱动器。例如，LSI Megaraid SAS 控制器使用电池支持的写缓存；这种类型的控制器需要 MegaCli64 工具来管理目标驱动器。要显示 LSI Megaraid SAS 的所有后端驱动器的状态，请使用：

```
# MegaCli64 -LDGetProp -DskCache -LAll -aALL
```

要禁用 LSI Megaraid SAS 的所有后端驱动器的写缓存，请使用：

```
# MegaCli64 -LDSetProp -DisDskCache -LaII -aALL
```

注意

在系统运行时，硬件 RAID 卡会对电池充电。如果系统关机的时间较长，则电池将不会充电，从而在电源故障期间存储的数据会丢失。

高端阵列

高端阵列在电源故障时有多种保护数据的方法。因此，不需要验证外部 RAID 存储中的内部驱动器的状态。

NFS

NFS 客户端不需要启用写障碍，因为数据完整性是由 NFS 服务器端处理的。因此，应配置 NFS 服务器，以确保断电过程中数据的持久性（无论是通过写障碍还是通过其他方法）。

第 23 章 存储 I/O 校准和大小

最近对 SCSI 和 ATA 标准的增强允许存储设备指明其首选的（在某些情况下需要的） I/O 校准 和 I/O 大小。对于将物理扇区大小从 512 字节增加到 4k 字节的较新的磁盘驱动器，此信息特别有用。此信息可能对 RAID 设备有好处，其中块大小和条带大小可能会影响性能。

Linux I/O 堆栈已被改进，可处理供应商提供的 I/O 校准和 I/O 大小信息，允许存储管理工具 (parted、lvm、mkfs. X 等)来优化数据放置和访问。如果传统设备没有导出 I/O 校准和大小数据，Red Hat Enterprise Linux 7 中的存储管理工具会更保守地在 4k （或更大 2 的指数 2）边界上校准 I/O。这可以确保 4k 扇区设备正确运行，即使它们没有指出任何需要/首选的 I/O 校准和大小。

有关确定操作系统从该设备中获取的信息，请参考 第 23.2 节 “用户空间访问”。存储管理工具随后将使用此数据来确定数据放置。

Red Hat Enterprise Linux 7 的 IO 调度程序已更改。默认的 IO 调度程序现在为 Deadline，但 SATA 驱动器除外。CFQ 是 SATA 驱动器的默认 IO 调度程序。为实现更快的存储，Deadline 优于 CFQ 的性能，在使用时可以提高性能，无需特殊调优。

如果默认值不适合某些磁盘（例如，SAS 轮转磁盘），则将 IO 调度程序更改为 CFQ。此实例将取决于工作负载。

23.1. 存储访问参数

操作系统使用以下信息来确定 I/O 校准和大小：

physical_block_size

设备可操作的最小内部单位

logical_block_size

用于外部定位设备上的位置

alignment_offset

Linux 块设备开始的字节数（分区/MD/LVM 设备）是底层物理校准的偏移量

minimum_io_size

设备用于随机 I/O 的首选最小单位

optimal_io_size

设备用于流 I/O 的首选单位

例如，某些 4K 扇区设备可以在内部使用 4K physical_block_size，但向 Linux 公开更精细的 512 字节 logical_block_size。这种差异可能会导致 I/O 不准。为解决这个问题，Red Hat Enterprise Linux 7 I/O 堆栈将尝试启动自然对齐边界(physical_block_size)上的所有数据区域，如果块设备的开头是底层物理校准的偏移量。

存储供应商还可以提供有关设备的随机 I/O (minimum_io_size)和流 I/O (optimal_io_size)的首选最小单元的 I/O 提示。例如： minimum_io_size 和 optimal_io_size 分别对应于 RAID 设备的块大小和条带大小。

23.2. 用户空间访问

始终要小心使用正确校准和大小的 I/O。这对于直接 I/O 访问尤为重要。直接 I/O 应在 logical_block_size 边界上对齐，并在 logical_block_size 的倍数上保持一致。

使用原生 4K 设备（即 logical_block_size 为 4K）现在对于在设备的 logical_block_size 的倍数中执行直接 I/O 至关重要。这意味着，如果带有原生 4k 设备的应用程序执行 512 字节校准的 I/O，而不是 4k 校准的 I/O 时将失败。

为避免这种情况，应用程序应参考设备的 I/O 参数来确保使用了正确的 I/O 校准和大小。如前文所述，I/O 参数通过 sysfs 和块设备 ioctl 接口公开。

如需更多信息，请参阅 man libblkid。此 man page 由 libblkid-devel 软件包提供。

sysfs 接口

- /sys/block/disk/alignment_offset

/sys/block/disk/alignment_offset

或者

/sys/block/disk/partition/alignment_offset

注意

文件位置取决于磁盘是物理磁盘（即本地磁盘、本地 RAID 或者多路径 LUN）还是虚拟磁盘。第一个文件位置适用于物理磁盘，第二个文件位置适用于虚拟磁盘。其原因是 virtio-blk 始终会报告分区的对齐值。物理磁盘可能也可能不会报告对齐值。

- /sys/block/disk/queue/physical_block_size
- /sys/block/disk/queue/logical_block_size
- /sys/block/disk/queue/minimum_io_size
- /sys/block/disk/queue/optimal_io_size

内核仍会为不提供 I/O 参数信息的"传统"设备导出这些 sysfs 属性，例如：

例 23.1. sysfs 接口

```
alignment_offset:    0
physical_block_size: 512
logical_block_size:  512
minimum_io_size:     512
optimal_io_size:     0
```

块设备 ioctls

- *BLKALIGNOFF: alignment_offset*
- *BLKPBSZGET: physical_block_size*
- *BLKSSZGET: logical_block_size*
- *BLKIOMIN: minimum_io_size*
- *BLKIOOPT ： optimal_io_size*

23.3. I/O 标准

这部分描述了 ATA 和 SCSI 设备所使用的 I/O 标准。

ATA

ATA 设备必须通过 IDENTIFY DEVICE 命令报告合适的信息。ATA 设备仅报告 physical_block_size、logical_block_size 和 alignment_offset 的 I/O 参数。其他 I/O 提示超出了 ATA 命令集的范围。

SCSI

Red Hat Enterprise Linux 7 中的 I/O 参数支持至少需要 SCSI 主命令 (SPC-3) 协议的版本 3。内核只会向声明符合 SPC-3 的设备发送 扩展查询 （其可以访问 BLOCK LIMITS VPD 页面）和 READ CAPACITY (16) 命令。

READ CAPACITY (16) 命令提供块大小和校准偏移量：

- *LOGICAL BLOCK LENGTH IN BYTES 用于派生 /sys/block/磁盘/queue/physical_block_size*

- LOGICAL BLOCKS PER PHYSICAL BLOCK EXPONENT 用于派生 /sys/block/磁盘/queue/logical_block_size

- LOWEST ALIGNED LOGICAL BLOCK ADDRESS 用于派生：

  - /sys/block/disk/alignment_offset

  - /sys/block/disk/partition/alignment_offset

BLOCK LIMITS VPD 页面(0xb0)提供 I/O 提示。它还使用 OPTIMAL TRANSFER LENGTH GRANULARITY 和 OPTIMAL TRANSFER LENGTH 派生：

- /sys/block/disk/queue/minimum_io_size

- /sys/block/disk/queue/optimal_io_size

sg3_utils 软件包提供 sg_inq 工具，可用于访问 BLOCK LIMITS VPD 页面。为此，请运行：

```
# sg_inq -p 0xb0 disk
```

23.4. 堆栈 I/O 参数

Linux I/O 堆栈的所有层都经过了设计，可以在堆栈之上传播各种 I/O 参数。当层消耗属性或聚合多个设备时，该层必须公开适当的 I/O 参数，以便上层设备或工具在转换后能够准确地查看存储。一些实际的示例如下：

- 对于非零 alignment_offset，仅应调整 I/O 堆栈中的一个层；当层相应地调整后，它将导出具有 alignment_offset 为 0 的设备。

- 使用 LVM 创建的条状设备映射器(DM)设备必须导出相对于条带数（磁盘数）和用户提供的块大小的 minimum_io_size 和 optimal_io_size。

在 Red Hat Enterprise Linux 7 中，设备映射器和软件 Raid (MD)设备驱动程序可用于任意组合具有不同 I/O 参数的设备。内核的块层将尝试合理组合各个设备的 I/O 参数。内核不会阻止组合异构设备；但是，请注意这样做所带来的风险。

例如，512 字节设备和 4K 设备可以合并为一个逻辑 DM 设备，其将有 4K 的 logical_block_size。在这种混合设备上分层的文件系统假定 4K 会以原子方式写入，但实际上，在发布到 512 字节设备时，该文件将跨越 8 个逻辑块地址。将 4K logical_block_size 用于更高级别的 DM 设备会增加系统崩溃时部分写入 512 字节设备的可能性。

如果组合多个设备的 I/O 参数会导致冲突，则块层可以发出一个警告，指出设备易受部分写入和/或未对齐的影响。

23.5. 逻辑卷管理器

LVM 提供用于管理内核 DM 设备的用户空间工具。LVM 将转换数据区域的开头（给定的 DM 设备将使用）来考虑与 LVM 管理的任何设备关联的非零 alignment_offset。这意味着逻辑卷将正确对齐 (alignment_offset=0)。

默认情况下，LVM 将针对任何 alignment_offset 进行调整，但可以通过在 /etc/lvm/lvm.conf 中将 data_alignment_offset_detection 设置为 0 来禁用此行为。不建议禁用此功能。

LVM 还将检测设备的 I/O 提示。设备数据区的开头将是 sysfs 中公开的 minimum_io_size 或 optimal_io_size 的倍数。如果 optimal_io_size 未定义（例如 0），LVM 将使用 minimum_io_size。

默认情况下，LVM 将自动决定这些 I/O 提示，但可以通过在 /etc/lvm/lvm.conf 中将 data_alignment_detection 设置为 0 来禁用此行为。不建议禁用此功能。

23.6. 分区和文件系统工具

这部分论述了不同的分区和文件系统管理工具如何与设备的 I/O 参数进行交互。

util-linux-ng 的 libblkid 和 fdisk

util-linux-ng 软件包提供的 libblkid 库包括用于访问设备的 I/O 参数的编程 API。libblkid 允许应用程序（特别是那些使用直接 I/O 的应用程序）正确调整其 I/O 请求的大小。util-linux-ng 的 fdisk 工具使用 libblkid 来确定设备的 I/O 参数，以优化所有分区的位置。fdisk 实用程序将在 1MB 边界上校准所有分区。

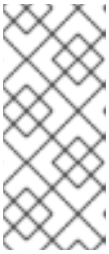
parted 和 libparted

parted 的 libparted 库也使用 libblkid 的 I/O 参数 API。Red Hat Enterprise Linux 7 安装程序 Anaconda 使用 libparted，这意味着由安装程序或 parted 创建的所有分区都会被正确对齐。对于在似乎不提供 I/O 参数的设备上创建的所有分区，默认校准为 1MB。

Heuristics parted 使用如下：

- *始终使用报告的 alignment_offset 作为第一个主分区开头的偏移量。*
- *如果定义了 optimal_io_size （即不是 0），在 optimal_io_size 边界上校准所有分区。*
- *如果 optimal_io_size 未定义（即 0），则 alignment_offset 为 0，而 minimum_io_size 是 2 的指数，使用 1MB 默认对齐。*

这是对“遗留”设备的总称，这些设备似乎不提供 I/O 提示。因此，默认情况下，所有分区将在 1MB 边界上校准。



注意

Red Hat Enterprise Linux 7 无法区分不提供 I/O 提示的设备和使用 alignment_offset=0 和 optimal_io_size=0 的设备。此类设备可能是单个 SAS 4K 设备；因此，最坏情况下，磁盘开始时会丢失 1MB 空间。

文件系统工具

不同的 mkfs. 文件系统工具已被改进，以使用设备的 I/O 参数。这些工具不允许格式化文件系统以使用比底层存储设备的 logical_block_size 小的块大小。

除 mkfs.gfs2 外，所有其他 mkfs.filesystem 工具也使用 I/O 提示来布局与底层存储设备的 minimum_io_size 和 optimal_io_size 相关的磁盘上数据结构和数据区域。这使得文件系统可以针对各种 RAID（条带）布局进行最佳格式化。

要设置通过 PXE 引导的基本得远程无盘系统，您需要以下软件包：

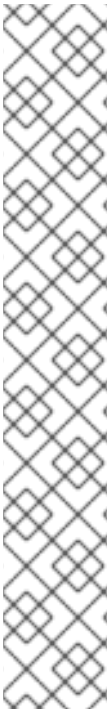
- tftp-server
- xinetd
- dhcp
- syslinux
- dracut-network

注意

安装 dracut-network 软件包后，请在 /etc/dracut.conf 中添加以下行：

```
add_dracutmodules+="nfs"
```

远程无盘系统引导需要 tftp 服务（由 tftp-server提供）和 DHCP 服务（由 dhcp提供）。tftp 服务用于通过 PXE 加载程序通过网络检索内核镜像和 initrd。



注意

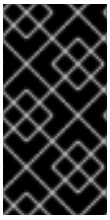
SELinux 只在 NFSv4.2 上被支持。要使用 SELinux，必须通过添加以下行在 /etc/sysconfig/nfs 中明确启用 NFS：

RPCNFSDARGS="-V 4.2"

然后，在 /var/lib/tftpboot/pxelinux.cfg/default 中，将 root=nfs:server-ip:/exported/root/directory 改为 root=nfs:server-ip:/exported/root/directory,vers=4.2。

最后，重启 NFS 服务器。

以下小节概述了在网络环境中部署远程无盘系统所需的步骤。



重要

有些 RPM 软件包开始使用文件功能（如 setcap 和 getcap）。但是，NFS 目前不支持它们，因此尝试安装或更新任何使用文件功能的软件包都会失败。

24.1. 为无盘客户端配置 TFTP 服务

先决条件

- 安装所需软件包。请查看 第 24 章 设置远程无盘系统

流程

要配置 tftp，请执行以下步骤：

过程 24.1. 配置 tftp

1. 通过网络启用 PXE 引导：

```
# systemctl enable --now tftp
```

2. tftp 根目录(chroot)位于 /var/lib/tftpboot 中。将 /usr/share/syslinux/pxelinux.0 复制到

/var/lib/tftpboot/ ：

```
# cp /usr/share/syslinux/pxelinux.0 /var/lib/tftpboot/
```

3. 在 tftp 根目录中创建 pxelinux.cfg 目录：

```
# mkdir -p /var/lib/tftpboot/pxelinux.cfg/
```

4. 配置防火墙规则以允许 tftp 流量。

因为 tftp 支持 TCP 包装器，您可以在 /etc/hosts.allow 配置文件中配置主机对 tftp 的访问。有关配置 TCP 包装器和 /etc/hosts.allow 配置文件的更多信息，请参阅 Red Hat Enterprise Linux 7 安全指南。hosts_access(5) 也提供有关 /etc/hosts.allow 的信息。

后续步骤

为无盘客户端配置 tftp 后，相应地配置 DHCP、NFS 和导出的文件系统。有关配置 DHCP、NFS 和导出的文件系统的步骤，请参考 第 24.2 节 “为无盘客户端配置 DHCP” 和 第 24.3 节 “为无盘客户端配置导出的文件系统”。

24.2. 为无盘客户端配置 DHCP

先决条件

- 安装所需软件包。请查看 第 24 章 设置远程无盘系统
- 配置 tftp 服务。请参阅 第 24.1 节 “为无盘客户端配置 tftp 服务”。

流程

1. 配置 tftp 服务器后，您需要在同一主机上设置 DHCP 服务。有关设置 DHCP 服务器的步骤，请参阅 配置 DHCP 服务器。

2. 通过在 /etc/dhcp/dhcpd.conf 中添加以下配置，在 DHCP 服务器上启用 PXE 引导：

```
allow booting;
allow bootp;
class "pxeclients" {
  match if substring(option vendor-class-identifier, 0, 9) = "PXEClient";
```

```
    next-server server-ip;
    filename "pxelinux.0";
}
```

- 使用 tftp 和 DHCP 服务所在的主机的 IP 地址替换 server-ip。

注意

当 libvirt 虚拟机用作无盘客户端时，libvirt 提供 DHCP 服务，并且不使用独立 DHCP 服务器。在这种情况下，必须使用 libvirt 网络配置中的 bootp file='filename' 选项启用网络引导，virsh net-edit。

后续步骤

现在，tftp 和 DHCP 已配置，请配置 NFS 和导出的文件系统。具体步骤请查看 第 24.3 节 “为无盘客户端配置导出的文件系统”。

24.3. 为无盘客户端配置导出的文件系统

先决条件

- 安装所需软件包。请查看 第 24 章 设置远程无盘系统
- 配置 tftp 服务。请参阅 第 24.1 节 “为无盘客户端配置 tftp 服务”。
- 配置 DHCP。请参阅 第 24.2 节 “为无盘客户端配置 DHCP”。

流程

1. 导出的文件系统的根目录（由网络中的无盘客户端使用）通过 NFS 共享。通过将根目录添加到 /etc/exports，将 NFS 服务配置为导出根目录。有关如何操作的步骤，请查看 第 8.6.1 节 “/etc/exports 配置文件”。

2. 要完全使用无盘客户机，根目录应该包含完整的 Red Hat Enterprise Linux。您可以克隆现有安装或安装新的基本系统：

   - 要与正在运行的系统同步，请使用 rsync 工具：

```
# rsync -a -e ssh --exclude='/proc/*' --exclude='/sys/*' \
    hostname.com:/exported-root-directory
```

- 使用要通过 rsync 进行同步的正在运行的系统的主机名替换 hostname.com。
- 使用导出的文件系统的路径替换 exported-root-directory。

- 要将 Red Hat Enterprise Linux 安装到导出的位置，请使用带有 --installroot 选项的 yum 工具：

```
# yum install @Base kernel dracut-network nfs-utils \
    --installroot=exported-root-directory --releasever=/
```

在被无盘客户端使用之前，要导出的文件系统仍然需要做进一步的配置。要做到这一点，请执行以下步骤：

过程 24.2. 配置文件系统

1. 选择无盘客户端应使用的内核(vmlinuz-kernel-version)并将其复制到 tftp 引导目录中：

```
# cp /boot/vmlinuz-kernel-version /var/lib/tftpboot/
```

2. 使用 NFS 支持创建 initrd （即 initramfs-kernel-version.img）：

```
# dracut --add nfs initramfs-kernel-version.img kernel-version
```

3. 使用以下命令将 initrd 的文件权限改为 644：

```
# chmod 644 initramfs-kernel-version.img
```

警告

如果没有更改 initrd 的文件权限，pxelinux.0 引导装载程序将失败，并显示 "file not found" 错误。

4. 同时，将生成的 initramfs-kernel-version.img 复制到 tftp 引导目录中。

5. 编辑默认启动配置，以使用 /var/lib/tftpboot/ 目录中的 initrd 和内核。此配置应指示无盘客户端的 root 用户以读写形式挂载导出的文件系统(/exported/root/directory)。在 /var/lib/tftpboot/pxelinux.cfg/default 文件中添加以下配置：

```
default rhel7

label rhel7
  kernel vmlinuz-kernel-version
  append initrd=initramfs-kernel-version.img root=nfs:server-ip:/exported/root/directory rw
```

使用 tftp 和 DHCP 服务所在的主机的 IP 地址替换 server-ip。

NFS 共享现在可以导出到无盘客户端。这些客户端可通过 PXE 通过网络引导。

第 25 章 在线存储管理

在操作系统运行时，通常最好在操作系统运行时添加、删除或重新调整存储设备，而且无需重新启动。本章概述了在系统运行时，在 Red Hat Enterprise Linux 7 主机系统中重新配置存储设备的步骤。它涵盖了 iSCSI 和光纤通道存储互连；以后可能会添加其他互连类型。

本章重点介绍添加、删除、修改和监控存储设备。它不会详细讨论光纤通道或 iSCSI 协议。有关这些协议的更多信息，请参阅其他文档。

本章引用各种 sysfs 对象。红帽建议 sysfs 对象名称和目录结构在主 Red Hat Enterprise Linux 版本中有所变化。这是因为上游 Linux 内核不提供稳定的内部 API。有关如何以传输的方式引用 sysfs 对象的说明，请参阅内核源树中的文档 /usr/share/doc/kernel-doc-版本/Documentation/sysfs-rules.txt。

警告

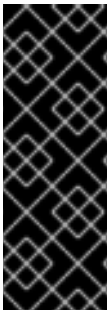
必须谨慎进行在线存储重新配置。处理过程中的系统故障或中断可能会导致意外的结果。红帽建议在更改操作期间最大程度地减少系统负载。这将减少配置更改期间发生 I/O 错误、内存不足或类似错误的可能性。以下章节提供了关于此方面的更具体的指南。

另外，红帽建议您在重新配置在线存储前备份所有数据。

25.1. 目标设置

Red Hat Enterprise Linux 7 使用 targetcli shell 作为前端来查看、编辑和保存 Linux-IO 目标配置，而无需直接操作内核目标的配置文件。targetcli 工具是一个命令行界面，允许管理员将本地存储资源导出，这些资源由文件、卷、本地 SCSI 设备或 RAM 磁盘支持。targetcli 工具具有一个基于树形的布局，包含内置 tab 自动完成功能，并提供完整的自动完成支持和内联文档。

targetcli 的层次结构并不总是与内核接口完全匹配，因为 targetcli 尽可能简化。



重要

要确保 targetcli 中所做的更改是持久的，请启动并启用 target 服务：

```
# systemctl start target
# systemctl enable target
```

25.1.1. 安装并运行 targetcli

要安装 targetcli，请使用：

```
# yum install targetcli
```

启动 目标 服务：

```
# systemctl start target
```

将目标 配置为在引导时启动：

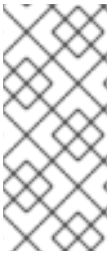
```
# systemctl enable target
```

在防火墙中打开端口 3260，并重新载入防火墙配置：

```
# firewall-cmd --permanent --add-port=3260/tcp
Success
# firewall-cmd --reload
Success
```

使用 targetcli 命令，然后使用 ls 命令获取树接口的布局：

```
# targetcli
:
/> ls
o- /.....[...]
  o- backstores.....[...]
  | o- block.....[Storage Objects: 0]
  | o- fileio.....[Storage Objects: 0]
  | o- pscsi.....[Storage Objects: 0]
  | o- ramdisk.....[Storage Ojbects: 0]
  o- iscsi.....[Targets: 0]
  o- loopback.....[Targets: 0]
```



注意

在 Red Hat Enterprise Linux 7.0 中，使用 Bash 中的 targetcli 命令（例如 targetcli iscsi/ create ）无法正常工作，且不会返回错误。从 Red Hat Enterprise Linux 7.1 开始，提供了一个错误状态代码，以便使用 targetcli 和 shell 脚本。

25.1.2. 创建后端存储

后端存储支持在本地计算机上存储导出的 LUN 数据的不同方法。创建存储对象定义了后端存储使用的资源。

注意

在 Red Hat Enterprise Linux 6 中，术语"backing-store"用于引用创建的映射。但是，为了避免在 Red Hat Enterprise Linux 7 中使用 'backstores' 的不同方法混淆，在 Red Hat Enterprise Linux 7 术语 'storage objects' 指的是所创建的映射和"backstores"来描述不同类型的后备设备。

LIO 支持的后端存储设备有：

FILEIO (Linux 文件支持的存储)

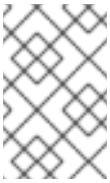
FILEIO 存储对象可以支持 write_back 或 write_thru 操作。write_back 启用本地文件系统缓存。这提高了性能，但会增加数据丢失的风险。建议您使用 write_back=false 禁用 write_back，而是使用 write_thru。

要创建 fileio 存储对象，请运行命令 /backstores/fileio create file_name file_location file_size write_back=false。例如：

```
/> /backstores/fileio create file1 /tmp/disk1.img 200M write_back=false
Created fileio file1 with size 209715200
```

BLOCK (Linux BLOCK 设备)

块驱动程序允许使用 /sys/block 中出现的任何块设备与 LIO 一起使用。这包括物理设备（如 HDD、SSD、CD、DVD）和逻辑设备（如软件或硬件 RAID 卷或 LVM 卷）。



注意

BLOCK backstores 通常提供最佳性能。

要使用任何块设备创建 BLOCK 后端存储，请使用以下命令：

```
# fdisk /dev/vdb
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x39dc48fb.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): *Enter*
Using default response p
Partition number (1-4, default 1): *Enter*
First sector (2048-2097151, default 2048): *Enter*
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151): +250M
Partition 1 of type Linux and of size 250 MiB is set

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

```
/> /backstores/block create name=block_backend dev=/dev/vdb
Generating a wwn serial.
Created block storage object block_backend using /dev/vdb.
```

注意

您还可以在逻辑卷中创建 BLOCK 后端存储。

PSCSI (Linux 直通 SCSI 设备)

任何支持直接传递 SCSI 命令且无 SCSI 模拟的存储对象，并且使用 /proc/scsi/scsi （如 SAS 硬盘驱动器）中显示的底层 SCSI 设备可以配置为后备存储。这个子系统支持 SCSI-3 及更高系统。

警告

PSCSI 应该只供高级用户使用。高级 SCSI 命令（如 Aysmmetric Logical Unit Assignment (ALUAs)或 Persistent Reservations (ALUAs)）或 Persistent Reservations （如 VMware ESX 和 vSphere 使用）通常不会在设备固件中实施，并可能导致故障或崩溃。如果有疑问，请对生产环境设置使用 BLOCK。

要为物理 SCSI 设备创建 PSCSI 后端存储，本例中使用 /dev/sr0 的 TYPE_ROM 设备，请使用：

```
/> backstores/pscsi/ create name=pscsi_backend dev=/dev/sr0
Generating a wwn serial.
Created pscsi storage object pscsi_backend using /dev/sr0
```

内存复制 RAM 磁盘(Linux RAMDISK_MCP)

Memory Copy RAM 磁盘(ramdisk)为 RAM 磁盘提供完整的 SCSI 模拟，并使用启动器的内存副本来分隔内存映射。这为多会话提供了功能，对于生产环境的快速易失性存储特别有用。

要创建 1GB RAM 磁盘后备存储，请使用以下命令：

```
/> backstores/ramdisk/ create name=rd_backend size=1GB
Generating a wwn serial.
Created rd_mcp ramdisk rd_backend with size 1GB.
```

25.1.3. 创建 iSCSI 目标

创建 iSCSI 目标：

过程 25.1. 创建 iSCSI 目标

1. *运行 targetcli。*

2. 移动到 iSCSI 配置路径：

```
/> iscsi/
```

注意

cd 命令还可接受更改目录，并只是列出要移动到的路径。

3. 使用默认目标名称创建 iSCSI 目标。

```
/iscsi> create
Created target
iqn.2003-01.org.linux-iscsi.hostname.x8664:sn.78b473f296ff
Created TPG1
```

或者使用指定名称创建 iSCSI 目标。

```
/iscsi > create iqn.2006-04.com.example:444
Created target iqn.2006-04.com.example:444
Created TPG1
```

4. 使用 ls 列出目标时，验证新创建的目标是否可见。

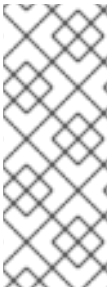
```
/iscsi > ls
o- iscsi.....[1 Target]
  o- iqn.2006-04.com.example:444.....[1 TPG]
    o- tpg1.....[enabled, auth]
      o- acls.....[0 ACL]
      o- luns.....[0 LUN]
      o- portals.....[0 Portal]
```

注意

从 Red Hat Enterprise Linux 7.1 开始，每当创建目标时，也会创建一个默认门户。

25.1.4. 配置 iSCSI 门户

要配置 iSCSI 门户，必须首先创建 iSCSI 目标并与 TPG 关联。有关如何进行此操作的步骤，请参考 第 25.1.3 节 “创建 iSCSI 目标”。



注意

从 Red Hat Enterprise Linux 7.1 创建 iSCSI 目标时，也会创建一个默认门户。此门户设置为使用默认端口号(00:3260)侦听所有 IP 地址。要删除此功能并只添加指定的门户，请使用 /iscsi/iqn-name/tpg1/portals delete ip_address=0.0.0.0 ip_port=3260，然后使用所需信息创建一个新门户。

过程 25.2. 创建 iSCSI 门户

1. 移至 TPG。

```
/iscsi> iqn.2006-04.example:444/tpg1/
```

2. 创建门户的方法有两种：创建一个默认门户，或者创建一个门户，指定要侦听的 IP 地址。

创建默认门户使用默认 iSCSI 端口 3260，并允许目标侦听该端口上的所有 IP 地址。

```
/iscsi/iqn.20...mple:444/tpg1> portals/ create
Using default IP port 3260
Binding to INADDR_Any (0.0.0.0)
Created network portal 0.0.0.0:3260
```

要创建门户，请指定要侦听的 IP 地址，请使用以下命令：

```
/iscsi/iqn.20...mple:444/tpg1> portals/ create 192.168.122.137
Using default IP port 3260
Created network portal 192.168.122.137:3260
```

3. 使用 ls 命令，验证新创建的门户是否可见。

```
/iscsi/iqn.20...mple:444/tpg1> ls
o- tpg..................................... [enambled, auth]
   o- acls ....................................[0 ACL]
   o- luns ....................................[0 LUN]
   o- portals .................................[1 Portal]
      o- 192.168.122.137:3260.....................[OK]
```

25.1.5. 配置 LUN

要配置 LUN，首先请创建存储对象。请参阅 第 25.1.2 节 “创建后端存储” 了解更多信息。

过程 25.3. 配置 LUN

1. *创建已创建的存储对象的 LUN。*

```
/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/ramdisk/rd_backend
Created LUN 0.

/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/block/block_backend
Created LUN 1.

/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/fileio/file1
Created LUN 2.
```

2. *显示更改。*

```
/iscsi/iqn.20...mple:444/tpg1> ls
o- tpg..................................... [enambled, auth]
   o- acls ....................................[0 ACL]
   o- luns ....................................[3 LUNs]
   |  o- lun0.....................[ramdisk/ramdisk1]
   |  o- lun1.................[block/block1 (/dev/vdb1)]
   |  o- lun2.................[fileio/file1 (/foo.img)]
   o- portals ................................[1 Portal]
      o- 192.168.122.137:3260.....................[OK]
```

注意

请注意，默认的 LUN 名称从 0 开始，而不是在 Red Hat Enterprise Linux 6 中使用 tgtd 时为 1。

3. *配置 ACL。如需更多信息，请参阅 第 25.1.6 节 “配置 ACL”。*

重要

默认情况下，使用读写权限创建 LUN。如果在创建 ACL 后添加了新的 LUN，则 LUN 将自动映射到所有可用 ACL。这可能导致安全风险。使用以下步骤以只读方式创建 LUN。

过程 25.4. 创建一个只读 LUN

1. *要创建具有只读权限的 LUN，首先使用以下命令：*

```
/> set global auto_add_mapped_luns=false
Parameter auto_add_mapped_luns is now 'false'.
```

这样可防止 LUN 自动映射到现有 ACL，从而允许手动映射 LUN。

2. 接下来，使用命令 iscsi/target_iqn_name/tpg1/acls/initiator_iqn_name/ create mapped_lun=next_sequential_LUN_number tpg_lun_or_backstore=backstore write_protect=1 来手动创建 LUN。

```
/> iscsi/iqn.2015-06.com.redhat:target/tpg1/acls/iqn.2015-06.com.redhat:initiator/ create
mapped_lun=1 tpg_lun_or_backstore=/backstores/block/block2 write_protect=1
Created LUN 1.
Created Mapped LUN 1.
/> ls
o- / ..... [...]
  o- backstores ..... [...]
  <snip>
  o- iscsi ..... [Targets: 1]
  | o- iqn.2015-06.com.redhat:target ..... [TPGs: 1]
  |   o- tpg1 ..... [no-gen-acls, no-auth]
  |     o- acls ..... [ACLs: 2]
  |     | o- iqn.2015-06.com.redhat:initiator .. [Mapped LUNs: 2]
  |     | | o- mapped_lun0 ..... [lun0 block/disk1 (rw)]
  |     | | o- mapped_lun1 ..... [lun1 block/disk2 (ro)]
  |     o- luns ..... [LUNs: 2]
  |     | o- lun0 ..... [block/disk1 (/dev/vdb)]
  |     | o- lun1 ..... [block/disk2 (/dev/vdc)]
  <snip>
```

mapping_lun1 行现在在结尾有(ro)（不像 mapping_lun0's (rw)），表示它是只读的。

3. 配置 ACL。如需更多信息，请参阅 第 25.1.6 节 “配置 ACL”。

25.1.6. 配置 ACL

为要连接的每个启动器创建 ACL。这会在启动器连接时强制身份验证，仅允许将 LUN 公开给每个启动器。通常，每个 initator 都具有对 LUN 的独占访问权限。目标和发起方都有唯一的标识名称。必须知道启动器的唯一名称来配置 ACL。对于 open-iscsi 启动器，可在 /etc/iscsi/initiatorname.iscsi 中找到。

过程 25.5. 配置 ACL

1. 移动到 acls 目录。

```
/iscsi/iqn.20...mple:444/tpg1> acls/
```

2. 创建 ACL。使用启动器上 /etc/iscsi/initiatorname.iscsi 中找到的启动器名称，或者使用更易于记住的名称，请参阅 第 25.2 节 “创建 iSCSI 启动器” 来确保 ACL 与启动器匹配。例如：

```
/iscsi/iqn.20...444/tpg1/acls> create iqn.2006-04.com.example.foo:888
Created Node ACL for iqn.2006-04.com.example.foo:888
Created mapped LUN 2.
Created mapped LUN 1.
Created mapped LUN 0.
```

注意

给定示例的行为取决于所使用的设置。在这种情况下，使用全局设置 auto_add_mapped_luns。这会自动将 LUN 映射到任何创建的 ACL。

您可以在目标服务器上的 TPG 节点中设置用户创建的 ACL：

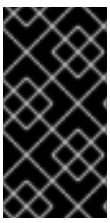
```
/iscsi/iqn.20...scsi:444/tpg1> set attribute generate_node_acls=1
```

3. 显示更改。

```
/iscsi/iqn.20...444/tpg1/acls> ls
o- acls ..................................................[1 ACL]
  o- iqn.2006-04.com.example.foo:888 ....[3 Mapped LUNs, auth]
    o- mapped_lun0 .............[lun0 ramdisk/ramdisk1 (rw)]
    o- mapped_lun1 .............[lun1 block/block1 (rw)]
    o- mapped_lun2 .............[lun2 fileio/file1 (rw)]
```

25.1.7. 通过以太网配置光纤通道(FCoE)目标

除了通过 FCoE 挂载 LUN 外，如 第 25.5 节 “通过以太网接口配置光纤通道” 所述，还支持通过 FCoE 将 LUN 导出到其它机器，并帮助 targetcli。



重要

在继续操作前，请参阅 第 25.5 节 “通过以太网接口配置光纤通道” 并验证基本 FCoE 设置已完成，并且 fcoeadm -i 显示配置的 FCoE 接口。

过程 25.6. 配置 FCoE 目标

1. 设置 FCoE 目标需要安装 targetcli 软件包，及其依赖项。有关 targetcli 基础知识和设置的详情，请参考 第 25.1 节 “目标设置”。

2. 在 FCoE 接口上创建 FCoE 目标实例。

```
/> tcm_fc/ create 00:11:22:33:44:55:66:77
```

   如果系统上存在 FCoE 接口，则创建后相应的选项卡将列出可用的接口。如果没有，请确保 fcoeadm -i 显示活动的接口。

3. 将后端存储映射到目标实例。

   例 25.1. 将后端存储映射到目标实例的示例

```
/> tcm_fc/00:11:22:33:44:55:66:77
/> luns/ create /backstores/fileio/example2
```

4. 允许从 FCoE 启动器访问 LUN。

```
/> acls/ create 00:99:88:77:66:55:44:33
```

   现在，LUN 应该可以被该启动器访问。

5. 要使更改在重启后保留，请使用 saveconfig 命令，并在提示时输入 yes。如果没有这样做，则配置将在重新引导后丢失。

6. 通过键入 exit 或输入 ctrl+D 退出 targetcli。

25.1.8. 使用 targetcli删除对象

要删除后端存储，请使用以下命令：

```
/> /backstores/backstore-type/backstore-name
```

要删除 iSCSI 目标（如 ACL）的部分，请使用以下命令：

```
/> /iscsi/iqn-name/tpg/acls/ delete iqn-name
```

要删除整个目标，包括所有 ACL、LUN 和门户，请使用以下命令：

```
/> /iscsi delete iqn-name
```

25.1.9. targetcli 参考

有关 targetcli 的更多信息，请参阅以下资源：

man targetcli

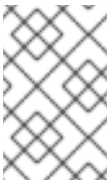
targetcli 手册页。它包括示例步骤。

Linux SCSI Target Wiki

http://linux-iscsi.org/wiki/Targetcli

Andy Grover 的录屏

https://www.youtube.com/watch?v=BkBGTBadOO8



注意

这于 2012 年 2 月 28 日上传。因此，服务名称已从 targetcli 改为 target。

25.2. 创建 ISCSI 启动器

在 Red Hat Enterprise Linux 7 中，iSCSI 服务默认是 lazily 启动：服务会在运行 iscsiadm 命令后启动。

过程 25.7. 创建 iSCSI 启动器

1. 安装 iscsi-initiator-utils：

```
# yum install iscsi-initiator-utils -y
```

2. 如果 ACL 在 第 25.1.6 节 “配置 ACL” 中被授予自定义名称，请相应地修改 /etc/iscsi/initiatorname.iscsi 文件。例如：

```
# cat /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2006-04.com.example.node1

# vi /etc/iscsi/initiatorname.iscsi
```

3. 发现目标：

```
# iscsiadm -m discovery -t st -p target-ip-address
10.64.24.179:3260,1 iqn.2006-04.com.example:3260
```

4. 使用在第 3 步中发现的目标 IQN 登录到目标：

```
# iscsiadm -m node -T iqn.2006-04.com.example:3260 -l
Logging in to [iface: default, target: iqn.2006-04.com.example:3260, portal:
10.64.24.179,3260] (multiple)
Login to [iface: default, target: iqn.2006-04.com.example:3260, portal: 10.64.24.179,3260]
successful.
```

对于连接到同一 LUN 的任意数量的 initator，只要将其特定启动器名称添加到 ACL 中，就可以遵循这个过程，如 第 25.1.6 节 “配置 ACL” 所述。

5. 找到 iSCSI 磁盘名称并在这个 iSCSI 磁盘中创建文件系统：

```
# grep "Attached SCSI" /var/log/messages

# mkfs.ext4 /dev/disk_name
```

使用 /var/log/messages 中显示的 iSCSI 磁盘名称替换 disk_name。

6. 挂载文件系统：

```
# mkdir /mount/point
# mount /dev/disk_name /mount/point
```

使用分区的挂载点替换 **/mount/point**。

7. 编辑 **/etc/fstab** 以在系统引导时自动挂载文件系统：

```
# vim /etc/fstab
/dev/disk_name /mount/point ext4 _netdev 0 0
```

使用 **iSCSI** 磁盘名称替换 **disk_name**。

8. 从目标登出：

```
# iscsiadm -m node -T iqn.2006-04.com.example:3260 -u
```

25.3. 设置 CHALLENGE-HANDSHAKE 身份验证协议

配置 **ACL** 并创建 **iSCSI** 启动器后，设置 **Challenge-Handshake Authentication Protocol (CHAP)**。有关配置 **ACL** 和创建 **iSCSI** 启动器的详情，请参考 第 25.1.6 节 “配置 ACL” 和 第 25.2 节 “创建 iSCSI 启动器”。

CHAP 允许用户使用密码保护目标。发起方必须了解这个密码才能连接到目标。

过程 25.8. 为目标设置 CHAP

1. 设置属性身份验证：

```
/iscsi/iqn.20...mple:444/tpg1> set attribute authentication=1

Parameter authentication is now '1'.
```

2. 设置 **userid** 和密码：

```
/iscsi/iqn.20...mple:444/tpg1> set auth userid=redhat

Parameter userid is now 'redhat'.
```

```
/iscsi/iqn.20...mple:444/tpg1> set auth password=redhat_passwd

Parameter password is now 'redhat_passwd'.
```

过程 25.9. 为发起方设置 CHAP

1. 编辑 iscsid.conf 文件：

   - 在 iscsid.conf 文件中启用 CHAP 身份验证：

     ```
     # vi /etc/iscsi/iscsid.conf

     node.session.auth.authmethod = CHAP
     ```

     默认情况下，node.session.auth.authmethod 选项被设置为 None。

   - 在 iscsid.conf 文件中添加目标用户名和密码：

     ```
     node.session.auth.username = redhat
     node.session.auth.password = redhat_passwd
     ```

2. 重启 iscsid 服务：

   ```
   # systemctl restart iscsid.service
   ```

如需更多信息，请参阅 targetcli 和 iscsiadm man page。

25.4. FIBRE CHANNEL

本节讨论光纤通道 API、原生 Red Hat Enterprise Linux 7 光纤通道驱动程序以及这些驱动程序的光纤通道功能。

25.4.1. 光纤通道 API

以下是 /sys/class/ 目录的列表，其中包含用于提供用户空间 API 的文件。在每个项目中，主机号由 H 指定，总线号为 B，目标为 T，逻辑单元号(LUN)为 L，远程端口号为 R。

重要的

如果您的系统使用多路径软件，红帽建议您在更改本部分中所描述的值之前咨询您的硬件厂商。

Transport: /sys/class/fc_transport/targetH:B:T/

- port_id - 24 位端口 ID/地址
- node_name - 64 位节点名称
- port_name - 64 位端口名称

Remote Port: /sys/class/fc_remote_ports/rport-H:B-R/

- port_id
- node_name
- port_name
- dev_loss_tmo ：控制 scsi 设备从系统中删除的时间。在 dev_loss_tmo 触发后，scsi 设备被删除。

  在 multipath.conf 中，您可以将 dev_loss_tmo 设置为 infinity，它将它的值设为 2,147,483,647 秒, 或 68 年，它是 dev_loss_tmo 的最大值。

  在 Red Hat Enterprise Linux 7 中，如果您没有设置 fast_io_fail_tmo 选项，dev_loss_tmo 的上限为 600 秒。默认情况下，如果 multipathd 服务正在运行，fast_io_fail_tmo 会在 Red Hat Enterprise Linux 7 中被设置为 5 秒；否则，它被设置为 off。

- fast_io_fail_tmo: 指定在将链接标记为"bad"前要等待的秒数。链接被标记为坏的后，现有正在运行的 I/O 或相应路径上的任何新 I/O 都将失败。

如果 I/O 处于阻塞队列中，则在 dev_loss_tmo 到期前和队列未阻塞前，它不会失败。

如果 fast_io_fail_tmo 设置为除 off 以外的任何值，则会取消dev_loss_tmo。如果 fast_io_fail_tmo 设为 off，则在从系统中删除该设备前不会出现 I/O 失败。如果 fast_io_fail_tmo 设置为一个数字，则在达到 fast_io_fail_tmo 设置的超时会立即触发 I/O 失败。

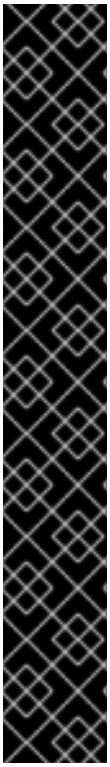
Host: /sys/class/fc_host/hostH/

- port_id
- issue_lip ：指示驱动程序重新发现远程端口。

25.4.2. 原生光纤通道驱动程序和能力

Red Hat Enterprise Linux 7 附带以下原生 Fibre Channel 驱动程序：

- lpfc
- qla2xxx
- zfcp
- bfa



重要

默认情况下，qla2xxx 驱动程序在 initiator 模式下运行。要将 qla2xxx 与 Linux-IO 搭配使用，请使用对应的 qlini_mode 模块参数启用光纤通道目标模式。

首先，确保安装了 qla 设备（如 ql2200-firmware 或类似）的固件软件包。

要启用目标模式，请在 /usr/lib/modprobe.d/qla2xxx.conf qla2xxx 模块配置文件中添加以下参数：

```
options qla2xxx qlini_mode=disabled
```

然后，使用 dracut -f 命令重建初始 ramdisk (initrd)，并重启系统以使更改生效。

表 25.1 “Fibre Channel API 功能” 描述每个原生 Red Hat Enterprise Linux 7 驱动程序的不同光纤通道 API 功能。X 表示对该功能的支持。

表 25.1. Fibre Channel API 功能

	lpfc	qla2xxx	zfcp	bfa
Transport port_id	X	X	X	X
Transport node_name	X	X	X	X
Transport port_name	X	X	X	X
Remote Port dev_loss_tmo	X	X	X	X
Remote Port fast_io_fail_tmo	X	X [a]	X [b]	X
Host port_id	X	X	X	X
Host issue_lip	X	X		X

lpfc	qla2xxx	zfcp	bfa

[a] 从 Red Hat Enterprise Linux 5.4 开始支持

[b] 从 Red Hat Enterprise Linux 6.0 开始支持

25.5. 通过以太网接口配置光纤通道

设置和部署以太网光纤通道(FCoE)接口需要两个软件包：

- fcoe-utils
- lldpad

安装这些软件包后，请执行以下流程来启用虚拟 LAN(VLAN) FCoE：

过程 25.10. 将以太网接口配置为使用 FCoE

1. 要配置新的 VLAN，请制作现有网络脚本的副本，如 /etc/fcoe/cfg-eth0，并将名称改为支持 FCoE 的以太网设备。这为您提供了一个要配置的默认文件。假设 FCoE 设备是 ethX，请运行：

```
# cp /etc/fcoe/cfg-ethx  /etc/fcoe/cfg-ethX
```

根据需要修改 cfg-ethX 的内容。值得注意的是，对于实现硬件数据中心桥接交换(DCBX)协议客户端的网络接口，将 DCB_REQUIRED 设置为 no。

2. 如果您希望设备在引导时自动载入，请在相应的 /etc/sysconfig/network-scripts/ifcfg-ethX 文件中设置 ONBOOT=yes。例如，如果 FCoE 设备是 eth2，请相应地编辑 /etc/sysconfig/network-scripts/ifcfg-eth2。

3. 运行以下命令启动数据中心桥接守护进程(dcbd)：

```
# systemctl start lldpad
```

4. 对于实现硬件 DCBX 客户端的网络接口，请跳过这一步。

   对于需要软件 DCBX 客户端的接口，请运行以下命令在以太网接口上启用数据中心桥接：

   ```
   # dcbtool sc ethX dcb on
   ```

   然后，运行以下命令在以太网接口上启用 FCoE：

   ```
   # dcbtool sc ethX app:fcoe e:1
   ```

   请注意，只有在以太网接口的 dcbd 设置没有改变时，这些命令才起作用。

5. 现在使用以下方法载入 FCoE 设备：

   ```
   # ip link set dev ethX up
   ```

6. 使用以下方法启动 FCoE：

   ```
   # systemctl start fcoe
   ```

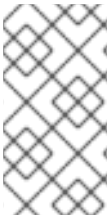
   如果光纤上的所有其他设置都正确，则 FCoE 设备很快出现。要查看配置的 FCoE 设备，请运行：

   ```
   # fcoeadm -i
   ```

在正确配置以太网接口以使用 FCoE 后，红帽建议您将 FCoE 和 lldpad 服务设置为在启动时运行。要做到这一点，请使用 systemctl 工具：

```
# systemctl enable lldpad
```

```
# systemctl enable fcoe
```

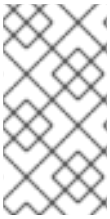


注意

运行 **# systemctl stop fcoe** 命令将停止守护进程，但不会重置 **FCoE** 接口的配置。为此，请运行 **# systemctl -s SIGHUP kill fcoe** 命令。

从 **Red Hat Enterprise Linux 7** 开始，网络管理器能够查询和设置具有 **DCB** 的以太网接口的 **DCB** 设置。

25.6. 将 FCOE 接口配置为在引导时自动挂载



注意

从 **Red Hat Enterprise Linux 6.1** 开始，本节中的说明包括在 **/usr/share/doc/fcoe-utils-版本/README** 中。有关小版本的任何可能的更改，请参阅该文档。

您可以通过 **udev** 规则、**autofs** 和其他类似方法挂载新发现的磁盘。但有时候，特定的服务可能会要求在启动时挂载 **FCoE** 磁盘。在这种情况下，应在 **fcoe** 服务运行时以及需要 **FCoE** 磁盘的任何服务启动前 立即 挂载 **FCoE** 磁盘。

要将 **FCoE** 磁盘配置为在引导时自动挂载，请将正确的 **FCoE** 挂载代码添加到 **fcoe** 服务的启动脚本中。**fcoe** 启动脚本为 **/lib/systemd/system/fcoe.service**。

FCoE 挂载代码在每个系统配置中都有所不同，无论您使用的是简单格式化的 **FCoE** 磁盘、**LVM** 还是多路径设备节点。

例 25.2. FCoE 挂载代码

以下是通过 **/etc/fstab** 中通配符指定的挂载文件系统的 **FCoE** 挂载代码示例：

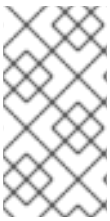
```
mount_fcoe_disks_from_fstab()
{
    local timeout=20
    local done=1
    local fcoe_disks=($(egrep 'by-path\/fc-.*_netdev' /etc/fstab | cut -d ' ' -f1))

    test -z $fcoe_disks && return 0

    echo -n "Waiting for fcoe disks . "
    while [ $timeout -gt 0 ]; do
  for disk in ${fcoe_disks[*]}; do
   if ! test -b $disk; then
```

```
  done=0
  break
 fi
done

test $done -eq 1 && break;
sleep 1
echo -n ". "
done=1
let timeout--
   done

   if test $timeout -eq 0; then
echo "timeout!"
   else
echo "done!"
   fi

   # mount any newly discovered disk
   mount -a 2>/dev/null
}
```

在 fcoe 服务脚本启动 fcoemon 守护进程后，应调用 mount_fcoe_disks_from_fstab 功能。这将挂载由 /etc/fstab 中的以下路径指定的 FCoE 磁盘：

```
/dev/disk/by-path/fc-0xXX:0xXX /mnt/fcoe-disk1 ext3  defaults,_netdev    0 0
/dev/disk/by-path/fc-0xYY:0xYY /mnt/fcoe-disk2 ext3  defaults,_netdev    0 0
```

带有 fc- 和 _netdev 子字符串的条目启用 mount_fcoe_disks_from_fstab 功能来识别 FCoE 磁盘挂载条目。有关 /etc/fstab 条目的更多信息，请参阅 man 5 fstab。



注意

fcoe 服务不为 FCoE 磁盘发现实现超时。因此，FCoE 挂载代码应该实现它自己的超时时间。

25.7. ISCSI

本节描述了 iSCSI API 和 iscsiadm 工具。在使用 iscsiadm 工具前，请先运行 yum install iscsi-initiator-utils 软件包来安装 iscsi-initiator-utils 软件包。

在 Red Hat Enterprise Linux 7 中，iSCSI 服务默认为 lazily 启动。如果 root 不在 iSCSI 设备上，或者没有标记为 node.startup = automatic 的节点，则 iSCSI 服务将不会启动，直到运行 iscsiadm 命令

需要 iscsid 或 iscsi 内核模块启动。例如，运行发现命令 iscsiadm -m discovery -t st -p ip:port 将导致 iscsiadm 启动 iSCSI 服务。

要强制 iscsid 守护进程运行和 iSCSI 内核模块进行加载，请运行 systemctl start iscsid.service。

25.7.1. iSCSI API

要获取有关正在运行的会话的信息，请运行：

```
# iscsiadm -m session -P 3
```

这个命令显示会话/设备状态、会话 ID (sid)、一些协商的参数以及可通过会话访问的 SCSI 设备。

对于较短的输出（例如，只显示 sid-to-node 映射），请运行：

```
# iscsiadm -m session -P 0
```

或者

```
# iscsiadm -m session
```

这些命令使用格式打印正在运行的会话列表：

```
driver [sid] target_ip:port,target_portal_group_tag proper_target_name
```

例 25.3. iscsisadm -m session 命令的输出

例如：

```
# iscsiadm -m session

tcp [2] 10.15.84.19:3260,2 iqn.1992-08.com.netapp:sn.33615311
tcp [3] 10.15.85.19:3260,3 iqn.1992-08.com.netapp:sn.33615311
```

有关 iSCSI API 的更多信息，请参阅 /usr/share/doc/iscsi-initiator-utils-版本/README。

25.8. 持久性命名

Red Hat Enterprise Linux 提供了多种方法来识别存储设备。在使用时，务必要使用正确的选项来识别每个设备，以避免意外访问错误的设备，特别是在安装到或重新格式化驱动器时。

25.8.1. 存储设备的主和次号

由 sd 驱动程序管理的存储设备由一组主要设备号及其关联的副号码来标识。用于此目的的主要设备号不在连续范围内。每个存储设备都由一个主号和一系列次要号表示，用于标识整个设备或设备中的分区。以 sd <letter (s)> 号的形式分配给设备和[号码在主号和次号之间进行直接关联]。每当 sd 驱动程序检测到新设备时，都会分配一个可用的主号码和次号范围。每当从操作系统中删除设备时，主号码和次号范围都会被释放，以便稍后重复使用。

当检测到设备时，会为每个设备分配主号和次号范围，以及相关的 sd 名称。这意味着，如果设备检测顺序发生了变化，主号和次号范围之间的关联以及相关的 sd 名称可能会发生变化。虽然这对于一些硬件配置（例如，内部 SCSI 控制器和磁盘将其 SCSI 目标 ID 被它们在机箱中的物理位置分配的 SCSI 目标 ID）不同，但它永远不会发生。可能发生这种情况的情况示例如下：

- 磁盘可能无法启动或响应 SCSI 控制器。这会导致正常设备探测不会检测到它。这个磁盘无法被系统访问，后续的设备将具有其主号和次号范围，包括相关的 sd 名称下移。例如，如果没有检测到通常称为 sdb 的磁盘，则通常称为 sdc 的磁盘将显示为 sdb。

- SCSI 控制器（主机总线适配器或 HBA）可能无法初始化，从而导致无法检测到连接到那个 HBA 的所有磁盘。任何连接到随后探测到的 HBA 的磁盘都会被分配不同的主号和次号范围，以及不同的相关 sd 名称。

- 如果系统中存在不同类型的 HBA，则驱动程序初始化顺序可能会改变。这会导致以不同顺序检测到连接到这些 HBA 的磁盘。如果 HBA 移到系统的不同 PCI 插槽中，也会发生这种情况。

- 连接到带有光纤通道、iSCSI 或 FCoE 适配器的系统的磁盘可能在检测存储设备时无法访问，例如，由于存储阵列或中间交换机断电。如果存储阵列需要比系统启动的时间更长，则系统在电源失败后重启时会出现这种情况。虽然某些光纤通道驱动程序支持一种机制来向 WWPN 映射指定持久的 SCSI 目标 ID，但这不会导致主号和次号范围，以及相关的 sd 名称被保留，它只提供一致的 SCSI 目标 ID 号。

这些原因使得在引用设备(例如在 /etc/fstab文件中的)时不希望使用主号和次号范围或相关的 sd 名称。可能挂载错误的设备，并可能导致数据崩溃。

然而，偶尔仍然需要引用 sd 名称，即使使用了其他机制（比如当设备报告错误时）。这是因为 Linux 内核在有关设备的内核消息中使用 sd 名称（以及 SCSI 主机/通道/目标/LUN 元组）。

25.8.2. 全球标识符(WWID)

全球识别符(WWID)可用于可靠地识别设备。它是 SCSI 标准要求所有 SCSI 设备提供的持久的、独立于系统的 ID。保证 WWID 标识符对于每个存储设备都是唯一的，并且独立于用于访问该设备的路径。

可通过发出 SCSI 查询来检索设备识别重要产品数据（第 0x 83 页）或 单元序列号 （第 0x80页）来获取此标识符。在 /dev/disk/by-id/ 目录中维护的符号链接中，可以看到从这些 WWID 到当前 /dev/sd 名称的映射。

例 25.4. WWID

例如，带有页 0x83 标识符的设备应该有：

```
scsi-3600508b400105e210000900000490000 -> ../../sda
```

或者，带有页 0x80 标识符的设备应该有：

```
scsi-SSEAGATE_ST373453LW_3HW1RHM6 -> ../../sda
```

Red Hat Enterprise Linux 自动维护系统上从基于 WWID 的设备名称到当前 /dev/sd 名称的正确映射。应用程序可以使用 /dev/disk/by-id/ 名称来引用磁盘上的数据，即使设备的路径有变化，即使从不同的系统访问该设备也一样。

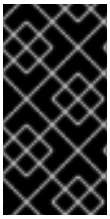
如果系统中存在多个路径到某个设备，DM 多路径 会使用 WWID 来检测这一点。然后，DM 多路径会在 /dev/mapper/wwid 目录中显示一个"pseudo-device"，如 /dev/mapper/3600508b400105df70000e00000ac0000。

命令 multipath -l 显示到非持久性标识符的映射： Host:Channel:Target:LUN, /dev/sd name, 和 major:minor 号。

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=0][active]
 \_ 5:0:1:1 sdc 8:32  [active][undef]
 \_ 6:0:1:1 sdg 8:96  [active][undef]
\_ round-robin 0 [prio=0][enabled]
 \_ 5:0:0:1 sdb 8:16  [active][undef]
 \_ 6:0:0:1 sdf 8:80  [active][undef]
```

- DM 多路径自动维护系统上每个基于 WWID 的设备名称到对应的 /dev/sd 名称的正确映射。这些名称可在路径更改之间保留，在从不同系统访问该设备时会保持一致。

- 当使用 user_friendly_names 功能(DM 多路径)时，WWID 会映射到 /dev/mapper/mpathn 格式的名称。默认情况下，此映射在 /etc/multipath/bindings 文件中维护。只要该文件被维护，这些 mpathn 名称就会持久存在。



重要

如果使用 user_friendly_names，则需要额外的步骤来获得集群中的一致名称。请参阅 DM 多路径 本书 中的集群部分中的 一致性多路径设备名称部分。

- 除了系统提供的这些持久名称外，您还可以使用 udev 规则来实现映射到存储的 WWID 的持久名称。

25.8.3. 由 /dev/disk/byjpeg 中的 udev 机制管理的设备名称

udev 机制由三个主要组件组成：

内核

生成在设备添加、删除或更改时发送到用户空间的事件。

udevd 服务

接收事件。

udev 规则

指定 udev 服务接收内核事件时要执行的操作。

这个机制用于 Linux 中的所有设备，而不仅仅是存储设备。对于存储设备，Red Hat Enterprise Linux 包含 udev 规则，该规则在 /dev/disk/ 目录中创建符号链接，允许存储设备被其内容引用、唯一标识符、序列号或用于访问该设备的硬件路径。

/dev/disk/by-label/

此目录中的条目提供一个符号链接名称，它通过存储在设备上的内容（即数据）中的标签引用存储设备。blkid 实用程序用于从设备读取数据，并确定该设备的名称（即标签）。例如：

```
/dev/disk/by-label/Boot
```

注意

该信息从设备上的内容（即数据）获取，以便在内容复制到另一个设备时，标签将保持不变。

该标签也可用于使用以下语法指向 /etc/fstab 中的设备：

```
LABEL=Boot
```

/dev/disk/by-uuid/

此目录中的条目提供一个符号链接名称，它通过存储在设备上的内容（即数据）中的唯一标识符来引用存储设备。blkid 实用程序用于从设备读取数据，并获取该设备的唯一标识符（即 UUID）。例如：

```
UUID=3e6be9de-8139-11d1-9106-a43f08d823a6
```

/dev/disk/by-id/

这个目录中的条目提供了一个符号链接名称，它通过唯一标识符（与所有其他存储设备不同）指向存储设备。标识符是设备的属性，但不存储在设备上的内容（即数据）中。例如：

```
/dev/disk/by-id/scsi-3600508e000000000ce506dc50ab0ad05
```

```
/dev/disk/by-id/wwn-0x600508e000000000ce506dc50ab0ad05
```

id 从设备的全球 ID 或设备序列号获得。/dev/disk/by-id/ 条目也可以包含分区号。例如：

```
/dev/disk/by-id/scsi-3600508e000000000ce506dc50ab0ad05-part1
```

```
/dev/disk/by-id/wwn-0x600508e000000000ce506dc50ab0ad05-part1
```

/dev/disk/by-path/

这个目录中的条目提供了一个符号链接名称，它通过用于访问该设备的硬件路径引用，从 PCI 层次结构中的存储控制器的引用，包括 SCSI 主机、频道、目标和 LUN 号，以及可选的分区号。虽然这些名称最好使用主号和次号或 sd 名称，但必须小心谨慎，以确保目标号在光纤通道 SAN 环境中不会改变（例如，通过使用持久绑定），如果主机适配器移至不同的 PCI 插槽，则更新名称。另外，如果 HBA 无法探测、如果以不同顺序载入驱动程序，或者系统上安装了新的 HBA，则 SCSI 主机号可能会改变。by-path 列表的示例如下：

```
/dev/disk/by-path/pci-0000:03:00.0-scsi-0:1:0:0
```

/dev/disk/by-path/ 条目可能还包括分区号，例如：

```
/dev/disk/by-path/pci-0000:03:00.0-scsi-0:1:0:0-part1
```

25.8.3.1. udev 设备命名约定的限制

以下是 udev 命名规则的一些限制。

- 执行查询时可能无法访问该设备，因为 udev 机制可能依赖于为 udev 事件处理 udev 规则时查询 存储设备的能力。当设备不在服务器机箱中时，这更可能会在光纤频道、iSCSI 或者 FCoE 存储设备中发生。
- 内核也可以随时发送 udev 事件，从而导致规则被处理，并可能导致 /dev/disk/by KafkaUser/ 链接在设备无法访问时被删除。
- 当 udev 事件生成和处理时（如检测到大量设备且用户空间 udevd 服务需要一定时间来处理每个事件）之间可能会有延迟。这可能会导致内核检测到该设备和 /dev/disk/byjpeg/ 名称可用之间延迟。
- 规则调用的 blkid 等外部程序可以在短时间内打开该设备，使设备无法被其他用途访问。

25.8.3.2. 修改持久性命名属性

虽然 udev 命名属性是持久的，但它们在系统重启后不会自行更改，但有一部分也是可以配置的。您可以为以下持久性命名属性设置自定义值：

- UUID ：文件系统 UUID

- LABEL: 文件系统标签

由于 UUID 和 LABEL 属性与文件系统相关，因此您需要使用的工具取决于该分区上的文件系统。

- 要更改 XFS 文件系统的 UUID 或 LABEL 属性，请卸载文件系统，然后使用 xfs_admin 工具更改属性：

```
# umount /dev/device
# xfs_admin [-U new_uuid] [-L new_label] /dev/device
# udevadm settle
```

- 要更改 ext4、ext3 或 ext2 文件系统的 UUID 或 LABEL 属性，请使用 tune2fs 工具：

```
# tune2fs [-U new_uuid] [-L new_label] /dev/device
# udevadm settle
```

使用您要设置的 UUID 替换 new_uuid ；例如，1cdfbc07-1c90-4984-b5ec-f61943f5ea50。使用标签替换 new_label，例如 backup_data。

注意

更改 udev 属性在后台进行，可能需要很长时间。udevadm settle 命令一直等待直到更改完全注册，这样可确保您的下一个命令能够正确使用新属性。

您还应在创建新设备后使用 命令；例如，在使用 parted 工具创建带有自定义 PARTUUID 或 PARTLABEL 属性的分区后，或者在创建新文件系统后使用。

25.9. 删除存储设备

在删除对存储设备本身的访问前，建议先备份设备上的数据。之后，刷新 I/O ，并删除对该设备的所有操作系统引用（如下所述）。如果该设备使用多路径，则对多路径"伪设备"(第 25.8.2 节 “全球标识符 (WWID)”)以及代表该设备路径的每个标识符进行此操作。如果您只删除多路径设备的路径，则其他路径会保留，那么这个流程会很简单，如 第 25.11 节 “添加存储设备或路径” 所述。

当系统面临内存压力时，不建议移除存储设备，因为 I/O 刷新会增加负载。要确定内存压力的级别，请运行命令 vmstat 1 100; 不建议删除设备，如果：

- 可用内存小于每 100 超过 10 个样本的 5%(可用 命令也可用于显示总内存)。
- 交换处于活跃状态(vmstat 输出中的非零 si 和 so 列)。

删除对设备的所有访问的一般流程如下：

过程 25.11. 确保彻底的设备删除

1. 根据需要，关闭设备的所有用户，并备份设备数据。
2. 使用 umount 卸载挂载该设备的任何文件系统。
3. 使用它的任何 md 和 LVM 卷中删除该设备。如果设备是 LVM 卷组的成员，那么可能需要使用 pvmove 命令将数据移出设备，然后使用 vgreduce 命令删除物理卷，以及（可选） pvremove 从磁盘中删除 LVM 元数据。
4. 运行 multipath -l 命令来查找配置为多路径设备 的设备列表。如果该设备被配置为多路径设备，请运行 multipath -f device 命令来清除任何未完成的 I/O 并删除多路径设备。
5. 将未完成的 I/O 刷新到已用路径。这对 raw 设备非常重要，其中没有 umount 或 vgreduce 操作来导致 I/O 刷新。只有在出现以下情况时，才需要执行此步骤：

   - 该设备没有配置为多路径设备，或者
   - 该设备被配置为多路径设备，I/O 是在过去某个点直接发布到其独立路径。

   使用以下命令刷新任何未完成的 I/O：

   ```
   # blockdev --flushbufs device
   ```

6.

删除对设备的基于路径名称的任何引用，如 /dev/sd、/dev/disk/by-path 或系统上的 major:minor 号、应用程序、脚本或工具。这对于确保以后添加的不同设备不会被误认为是当前设备非常重要。

7.

最后，从 SCSI 子系统中删除该设备的每个路径。为此，请使用命令 echo 1 > /sys/block/device-name/device/delete，例如，其中 device-name 可以是 sde。

此操作的另一个变化是 echo 1 > /sys/class/scsi_device/h:c:t:l/device/delete，其中 h 是 HBA 号，c 是 HBA 上的频道，t 是 SCSI 目标 ID，l 是 LUN。

注意

这些命令的旧形式 echo "scsi remove-single-device 0 0 0 0" > /proc/scsi/scsi 已被弃用。

您可以从各种命令（如 lsscsi、scsi_id、multipath -l 和 ls -l ）确定设备的 device-name、HBA 号、HBA 通道、SCSI 目标 ID 和 LUN。

执行 过程 25.11, “确保彻底的设备删除” 后，可以从正在运行的系统中安全地删除设备。在这样做的同时，不需要停止到其他设备的 I/O。

不建议其它流程，如物理删除设备，然后重新扫描 SCSI 总线（如 第 25.12 节 “扫描存储互连” 中所述），来更新操作系统状态以反映更改。这会导致因为 I/O 超时造成的延迟，设备可能会被意外删除。如果需要对互连执行重新扫描，必须在 I/O 暂停时完成，如 第 25.12 节 “扫描存储互连” 所述。

25.10. 删除存储设备的路径

如果您要删除使用多路径的设备的路径（不会影响该设备的其他路径），则常规流程如下：

过程 25.12. 删除存储设备的路径

1.

删除对设备的基于路径名称的任何引用，如 /dev/sd 或 /dev/disk/by-path 或系统上的 major:minor 号、应用程序、脚本或工具。这对于确保以后添加的不同设备不会被误认为是当前设备非常重要。

2.
使用 echo offline > /sys/block/sda/device/state 使路径离线。

这将导致发送至该路径上设备的任何后续 I/O 立即失败。device-mapper-multipath 将继续使用设备的剩余路径。

3.
从 SCSI 子系统中删除路径。为此，请使用命令 echo 1 > /sys/block/device-name/device/delete，其中 device-name 可以是 sde （如 过程 25.11, “确保彻底的设备删除”所述）。

执行 过程 25.12, “删除存储设备的路径” 后，可以从正在运行的系统中安全地删除路径。这样做时不需要停止 I/O，因为 device-mapper-multipath 将根据配置的路径分组和故障转移策略将 I/O 重新路由到剩余的路径。

不建议其他流程，如网线的物理移除，然后再重新扫描 SCSI 总线，来更新操作系统状态以反映更改。这会导致因为 I/O 超时造成的延迟，设备可能会被意外删除。如果需要对互连执行重新扫描，必须在 I/O 暂停时完成，如 第 25.12 节 “扫描存储互连” 所述。

25.11. 添加存储设备或路径

添加设备时，请注意系统分配给新设备的设备名称(/dev/sd 名称、main :minor 号和 /dev/disk/by-path 名称)。因此，请确保已删除所有对基于路径的设备名称的旧引用。否则，新设备可能会被误认为是旧设备。

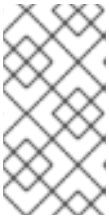
过程 25.13. 添加存储设备或路径

1.
添加存储设备或路径的第一步是物理启用对新存储设备的访问，或对现有设备的新路径的访问。这是通过对光纤通道或 iSCSI 存储服务器使用特定于供应商的命令来完成的。执行此操作时，请注意将呈现给您的主机的新存储的 LUN 值。如果存储服务器是光纤通道，可记下存储服务器的 全球节点名称 (WWNN)，并确定存储服务器上的所有端口是否有单个 WWNN。否则，请注意将用于访问新 LUN 的每个端口的 全球端口名称 (WWPN)。

2.
接下来，使操作系统知道新的存储设备，或现有设备的路径。建议使用的命令有：

```
$ echo "c t l" > /sys/class/scsi_host/hosth/scan
```

在上一命令中，h 是 HBA 号，c 是 HBA 上的频道，t 是 SCSI 目标 ID，l 是 LUN。

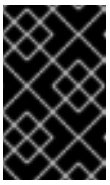


注意

此命令的旧形式 echo "scsi add-single-device 0 0 0 0" > /proc/scsi/scsi 已被弃用。

a.

在某些光纤通道硬件中，RAID 阵列上新创建的 LUN 可能在执行 循环初始化协议（LIP)操作之前无法被操作系统看到。有关如何操作的说明，请参阅 第 25.12 节 “扫描存储互连”。



重要

如果需要 LIP，则需要在执行此操作时需要停止 I/O。

b.

如果在 RAID 阵列中添加了新的 LUN，但仍然没有被操作系统配置，请使用 sg_luns 命令（这是 sg3_utils 软件包的一部分）确认阵列导出的 LUN 列表。这会将 SCSI REPORT LUNS 命令发布到 RAID 阵列，并返回存在的 LUN 列表。

对于为所有端口实现单个 WWNN 的光纤通道存储服务器，您可以通过搜索 sysfs 中的 WWNN 来确定正确的 h、c 和 t 值（如 HBA 号、HBA 通道和 SCSI 目标 ID）。

例 25.5. 确定正确的 h、c 和 t 值

例如，如果存储服务器的 WWNN 是 0x5006016090203181，请使用：

```
$ grep 5006016090203181 /sys/class/fc_transport/*/node_name
```

这应该显示类似于如下的输出：

```
/sys/class/fc_transport/target5:0:2/node_name:0x5006016090203181
/sys/class/fc_transport/target5:0:3/node_name:0x5006016090203181
/sys/class/fc_transport/target6:0:2/node_name:0x5006016090203181
/sys/class/fc_transport/target6:0:3/node_name:0x5006016090203181
```

这表示有四个光纤通道路由到这个目标（两个单通道 HBA，各自指向两个存储端口）。假设 LUN 值为 56，以下命令将配置第一个路径：

```
$ echo "0 2 56" >  /sys/class/scsi_host/host5/scan
```

-

必须为新设备的每个路径完成此操作。

对于不为所有端口实现单个 WWN 的光纤通道存储服务器，您可以通过在 sysfs 中搜索每个 WWPN 来确定正确的 HBA 号、HBA 通道和 SCSI 目标 ID。

确定 HBA 号、HBA 通道和 SCSI 目标 ID 的另一种方法是参考已在与新设备相同的路径上配置的另一个设备。这可以通过各种命令完成，如 lsscsi、scsi_id、multipath -l 和 ls -l /dev/disk/byrew。此信息以及新设备的 LUN 号，可以探测并配置新设备的路径，如上所示。

3.

向设备添加所有 SCSI 路径后，执行 multipath 命令，并检查该设备是否已正确配置。此时，可以将设备添加到 md、LVM、mkfs 或 mount 中。

如果遵循上述步骤，则可以安全地将设备添加到正在运行的系统中。在完成此操作的时，不需要停止到其他设备的 I/O 。在存储 I/O 进行的过程中，不建议采用其他涉及 SCSI 总线的重新扫描（或重置）的流程，这会导致操作系统更新其状态，以反映当前的设备连接。

25.12. 扫描存储互连

某些命令允许您重置、扫描或重置并扫描一个或多个互连，这些互连可能在一次操作中添加和删除了多个设备。这种类型的扫描可能会造成混乱，因为它可能会在 I/O 操作超时时造成延迟，并意外删除设备。红帽建议 仅在需要时 使用互连扫描。扫描存储互连时，请观察以下限制：

- 在执行此流程前，必须暂停并刷新受影响的互连上的所有 I/O，并在 I/O 恢复前检查扫描的结果。

- 与删除设备一样，当系统面临内存压力时，不建议进行互连扫描。要确定内存压力的级别，请运行 vmstat 1 100 命令。如果每 100 个样本中超过 10 个样本的空闲内存小于总内存的 5%，则不建议进行互连扫描。另外，如果交换处于活跃状态(vmstat 输出中的非零 si 和 so 列)，则不建议进行互连扫描。free 命令还可以显示总内存。

以下命令可用于扫描存储互连：

```
echo "1" > /sys/class/fc_host/hostN/issue_lip
```

（用主机号替换 N。）

此操作执行 循环初始化协议 (LIP)，扫描互连，并导致更新 SCSI 层以反映总线上当前的设备。基本上，LIP 是总线重置，导致设备添加和删除。为了在光纤通道互连上配置新的 SCSI 目标，这个流程是必需的。

请注意，issue_lip 是异步操作。该命令可以在整个扫描完成之前完成。您必须监控 /var/log/messages，以确定 issue_lip 何时完成。

lpfc、qla2xxx 和 bnx2fc 驱动程序支持 issue_lip。有关 Red Hat Enterprise Linux 中每个驱动程序支持的 API 功能的更多信息，请参阅 表 25.1 “Fibre Channel API 功能”。

/usr/bin/rescan-scsi-bus.sh

Red Hat Enterprise Linux 5.4 中引入了 /usr/bin/rescan-scsi-bus.sh 脚本。默认情况下，此脚本会扫描系统上的所有 SCSI 总线，并更新 SCSI 层以反映总线上的新设备。脚本提供了允许移除设备和发布 LIP 的附加选项。有关此脚本的详情，包括已知问题，请参阅 第 25.18 节 “通过 rescan-scsi-bus.sh 添加/删除逻辑单元”。

echo "- - -" > /sys/class/scsi_host/hosth/scan

这与 第 25.11 节 “添加存储设备或路径” 中所述的添加存储设备或路径的命令相同。然而，在这种情况下，通道号、SCSI 目标 ID 和 LUN 值会被通配符替代。允许标识符和通配符的任意组合，以便您可以根据需要将命令设置为特定的或广泛的。这个流程添加了 LUN，但不会删除它们。

modprobe --remove driver-name, modprobe driver-name

运行 modprobe --remove driver-name 命令，后跟 modprobe driver-name 命令完全重新初始化由驱动程序控制的所有互连的状态。尽管非常极端，但在某些情况下，使用上述命令可能是合适的。例如，命令可用于使用不同的模块参数值重新启动驱动程序。

25.13. iSCSI 发现配置

默认 iSCSI 配置文件为 /etc/iscsi/iscsid.conf。此文件包含 iscsid 和 iscsiadm 使用的 iSCSI 设置。

在目标发现过程中，iscsiadm 工具使用 /etc/iscsi/iscsid.conf 中的设置来创建两种类型的记录：

/var/lib/iscsi/nodes中的节点记录

登录到目标时，iscsiadm 会使用此文件中的设置。

/var/lib/iscsi/discovery_type中的发现记录

向同一目标执行发现时，iscsiadm 会使用此文件中的设置。

在对发现使用不同的设置之前，请先删除当前的发现记录（例如 /var/lib/iscsi/discovery_type）。要做到这一点，请使用以下命令：[5]

```
# iscsiadm -m discovery -t discovery_type -p target_IP:port -o delete
```

在这里，discovery_type 可以是 sendtargets、isns 或 fw。

有关不同类型的发现的详情，请参考 iscsiadm(8) man page 中的 DISCOVERY TYPES 部分。

重新配置发现记录设置的方法有两种：

- 在执行发现前，直接编辑 /etc/iscsi/iscsid.conf 文件。发现设置使用前缀 发现; 查看它们，请运行：

  ```
  # iscsiadm -m discovery -t discovery_type -p target_IP:port
  ```

- 或者，iscsiadm 也可用于直接更改发现记录设置，如下所示：

  ```
  # iscsiadm -m discovery -t discovery_type -p target_IP:port -o update -n setting -v %value
  ```

  有关可用 设置选项 和有效值选项的详情，请参考 iscsiadm(8) man page。

配置发现设置后，任何后续尝试发现新目标都将使用新设置。有关如何扫描新 iSCSI 目标的详情，请参考 第 25.15 节 “扫描 iSCSI Interconnects”。

有关配置 iSCSI 目标发现的更多信息，请参阅 iscsiadm 和 iscsid 的 man page。/etc/iscsi/iscsid.conf 文件还包含正确的配置语法示例。

25.14. 配置 ISCSI 卸载和接口绑定

本章论述了如何在使用软件 iSCSI 时设置 iSCSI 接口，以便将会话绑定到 NIC 端口。它还描述了如何设置接口以用于支持卸载的网络设备。

可以配置网络子系统来确定 iSCSI 接口应该用于绑定的路径/NIC。例如，如果在不同子网中设置了端口和 NIC，则不需要为绑定手动配置 iSCSI 接口。

在尝试为绑定配置 iSCSI 接口前，请先运行以下命令：

```
$ ping -I ethX target_IP
```

如果 ping 失败，您将无法将会话绑定到 NIC。如果是这种情况，请先检查网络设置。

25.14.1. 查看可用的 iface 配置

以下 iSCSI 启动器实现支持 iSCSI 卸载和接口绑定：

软件 iSCSI

此堆栈为每个会话分配一个 iSCSI 主机实例（即 scsi_host），每个会话都有一个连接。因此，/sys/class_scsi_host 和 /proc/scsi 将为您登录的每个连接/会话报告一个 scsi_host。

卸载 iSCSI

此堆栈为每个 PCI 设备分配一个 scsi_host。因此，主机总线适配器中的每个端口都显示为不同的 PCI 设备，每个 HBA 端口都有不同的 scsi_host。

要管理两种类型的启动器，iscsiadm 使用 iface 结构。使用这种结构时，必须在用于绑定会话的每个 HBA 端口、软件 iSCSI 或网络设备(ethX)的 /var/lib/iscsi/ifaces 中输入 iface 配置。

要查看可用的 iface 配置，请运行 iscsiadm -m iface。这将以以下格式显示 iface 信息：

```
iface_name transport_name,hardware_address,ip_address,net_ifacename,initiator_name
```

对于每个值/设置的解释，请参考下表。

表 25.2. iface 设置

设置	描述
iface_name	iface 配置名称。
transport_name	驱动程序的名称
hardware_address	MAC 地址
ip_address	此端口使用的 IP 地址
net_iface_name	用于软件 iSCSI 会话的 vlan 或别名绑定的名称。对于 iSCSI 卸载，net_iface_name 将是 <empty>，因为这个值在重启后不会保留。
initiator_name	此设置用于覆盖启动器的默认名称，该名称在 /etc/iscsi/initiatorname.iscsi中定义

例 25.6. iscsiadm -m iface 命令的输出示例

以下是 iscsiadm -m iface 命令的输出示例：

```
iface0 qla4xxx,00:c0:dd:08:63:e8,20.15.0.7,default,iqn.2005-06.com.redhat:madmax
iface1 qla4xxx,00:c0:dd:08:63:ea,20.15.0.9,default,iqn.2005-06.com.redhat:madmax
```

对于软件 iSCSI，每个 iface 配置都必须具有唯一的名称（少于 65 个字符）。支持卸载的网络设备的 iface_name 显示为 transport_name 格式。hardware_name.

例 25.7. 带有 Chelsio 网卡的 iscsiadm -m iface 输出

例如，使用 Chelsio 网卡的系统上的 iscsiadm -m iface 输出示例可能显示为：

```
default tcp,<empty>,<empty>,<empty>,<empty>
iser iser,<empty>,<empty>,<empty>,<empty>
cxgb3i.00:07:43:05:97:07 cxgb3i,00:07:43:05:97:07,<empty>,<empty>,<empty>
```

也可以以更友好的方式显示特定 iface 配置的设置。为此，请使用选项 -I iface_name。这将以以下格式显示设置：

```
iface.setting = value
```

例 25.8. 使用带有 Chelsio 融合网络适配器的 iface 设置

使用前面的示例，同一 Chelsio 聚合网络适配器的 iface 设置（即 iscsiadm -m iface -I cxgb3i.00:07:43:05:97:07）将显示为：

```
# BEGIN RECORD 2.0-871
iface.iscsi_ifacename = cxgb3i.00:07:43:05:97:07
iface.net_ifacename = <empty>
iface.ipaddress = <empty>
iface.hwaddress = 00:07:43:05:97:07
iface.transport_name = cxgb3i
iface.initiatorname = <empty>
# END RECORD
```

25.14.2. 为软件 iSCSI 配置 iface

如前文所述，每个网络对象都需要一个 iface 配置，用于绑定会话。

之前

要为软件 iSCSI 创建 iface 配置，请运行以下命令：

```
# iscsiadm -m iface -I iface_name --op=new
```

这将创建一个带有指定 iface _name 的新 空 iface 配置。如果现有的 iface 配置已经具有相同的 iface_name，那么它将被一个新的空 iface_name 覆盖。

要配置 iface 配置的特定设置，请使用以下命令：

```
# iscsiadm -m iface -I iface_name --op=update -n iface.setting -v hw_address
```

例 25.9. 设置 MAC 地址 iface0

例如，要将 iface0 的 MAC 地址(hardware_address)设置为 00:0F:1F:92:6B:BF，请运行：

```
# iscsiadm -m iface -I iface0 --op=update -n iface.hwaddress -v 00:0F:1F:92:6B:BF
```

警告

不要使用 default 或 iser 作为 iface 名称。这两个字符串都是 iscsiadm 使用的特殊值，用于向后兼容。任何手动创建的名为 default 或 iser 的 iface 配置都会禁用向后兼容。

25.14.3. 为 iSCSI 卸载配置 iface

默认情况下，iscsiadm 为每个端口创建一个 iface 配置。要查看可用的 iface 配置，请使用软件 iSCSI 中的相同命令： iscsiadm -m iface。

在对 iSCSI 卸载使用网卡的 iface 之前，首先将卸载接口的 iface.ipaddress 值设置为接口应使用的启动器 IP 地址：

- 对于使用 be2iscsi 驱动程序的设备，IP 地址在 BIOS 设置屏幕中配置。
- 对于所有其他设备，若要配置 iface 的 IP 地址，请使用：

```
# iscsiadm -m iface -I iface_name -o update -n iface.ipaddress -v initiator_ip_address
```

例 25.10. 设置 Chelsio 卡的 iface IP 地址

例如，在使用 iface 名称 cxgb3i.00:07:43:05:97:07 时，要将 iface IP 地址设置为 20.15.0.66，请使用：

```
# iscsiadm -m iface -I cxgb3i.00:07:43:05:97:07 -o update -n iface.ipaddress -v 20.15.0.66
```

25.14.4. 绑定/解绑到端口的 iface

每当使用 iscsiadm 扫描互连时，它将首先检查 /var/lib/iscsi/ifaces 中每个 iface 配置的 iface.transport 设置。然后，iscsiadm 工具将发现的端口绑定到其 iface.transport 为 tcp 的任何 iface。

实现此行为是出于兼容的原因。要覆盖此功能，请使用 **-I iface_name** 指定哪个端口绑定到 iface，如下所示：

```
# iscsiadm -m discovery -t st -p target_IP:port -I iface_name -P 1
```
[5]

默认情况下，iscsiadm 工具不会自动将任何端口绑定到使用卸载的 iface 配置。这是因为此类 iface 配置不会将 iface.transport 设置为 tcp。因此，iface 配置需要手动绑定到发现的门户。

也可以防止端口绑定到任何现有的 iface。要做到这一点，使用 **default** 作为 **iface_name**，如下所示：

```
# iscsiadm -m discovery -t st -p IP:port -I default -P 1
```

要删除目标和 iface 之间的绑定，请使用：

```
# iscsiadm -m node -targetname proper_target_name -I iface0 --op=delete
```
[6]

要删除特定 iface 的所有绑定，请使用：

```
# iscsiadm -m node -I iface_name --op=delete
```

要删除特定端口（例如，Equalogic 目标）的绑定，请使用：

```
# iscsiadm -m node -p IP:port -I iface_name --op=delete
```

注意

如果没有在 /var/lib/iscsi/ iface 中定义的 iface 配置，并且没有使用 **-I** 选项，iscsiadm 将允许网络子系统决定特定门户应使用哪个设备。

25.15. 扫描 ISCSI INTERCONNECTS

对于 iSCSI，如果目标发送指示新存储的 iSCSI async 事件，则会自动执行扫描。

但是，如果目标没有发送 iSCSI async 事件，您需要使用 iscsiadm 程序手动扫描它们。但是，在这样做前，您需要首先检索正确的 --targetname 和 --portal 值。如果您的设备模型只支持每个目标的单一逻辑单元和门户，请使用 iscsiadm 向主机发出 sendtargets 命令，如下所示：

```
# iscsiadm -m discovery -t sendtargets -p target_IP:port
[5]
```

输出将以以下格式显示：

```
target_IP:port,target_portal_group_tag proper_target_name
```

例 25.11. 使用 iscsiadm 发出 sendtargets 命令

例如，在一个带有 proper_target_name 为 iqn.1992-08.com.netapp:sn.33615311 且 target_IP:port 为 10.15.85.19:3260 的目标中，输出可能显示为：

```
10.15.84.19:3260,2 iqn.1992-08.com.netapp:sn.33615311
10.15.85.19:3260,3 iqn.1992-08.com.netapp:sn.33615311
```

在这个示例中，目标有两个门户，每个门户都使用 target_ip:ports 为 10.15.84.19:3260 和 10.15.85.19:3260。

要查看每个会话将使用哪个 iface 配置，请添加 -P 1 选项。这个选项还将以树形格式打印会话信息，如下所示：

```
Target: proper_target_name
    Portal: target_IP:port,target_portal_group_tag
        Iface Name: iface_name
```

例 25.12. 查看 iface 配置

例如，使用 iscsiadm -m discovery -t sendtargets -p 10.15.85.19:3260 -P 1，输出可能显示为：

```
Target: iqn.1992-08.com.netapp:sn.33615311
   Portal: 10.15.84.19:3260,2
      Iface Name: iface2
   Portal: 10.15.85.19:3260,3
      Iface Name: iface2
```

这意味着目标 iqn.1992-08.com.netapp:sn.33615311 将使用 iface2 作为其 iface 配置。

对于某些设备模型，一个目标可能有多个逻辑单元和门户。在这种情况下，首先向主机发出 sendtargets 命令，以查找目标上的新门户。然后，使用以下命令重新扫描现有会话：

```
# iscsiadm -m session --rescan
```

您还可以通过指定会话的 SID 值来重新扫描特定的会话，如下所示：

```
# iscsiadm -m session -r SID --rescan[7]
```

如果您的设备支持多个目标，则需要向主机发出 sendtargets 命令，以查找每个目标的新门户。重新扫描现有会话，以使用 --rescan 选项发现现有会话上的新逻辑单元。

重要

用于检索 **--targetname** 和 **--portal** 值的 **sendtargets** 命令会覆盖 **/var/lib/iscsi/nodes** 数据库的内容。然后，将使用 **/etc/iscsi/iscsid.conf** 中的设置重新填充此数据库。但是，如果会话当前已登录并在使用，则不会发生这种情况。

要安全地添加新目标/端口或删除旧目标/端口，请分别使用 **-o new** 或 **-o delete** 选项。例如，要在不覆盖 **/var/lib/iscsi/nodes** 的情况下添加新目标/端口，请使用以下命令：

```
iscsiadm -m discovery -t st -p target_IP -o new
```

要删除在发现过程中没有显示目标的 **/var/lib/iscsi/nodes** 条目，请使用：

```
iscsiadm -m discovery -t st -p target_IP -o delete
```

您还可以同时执行这两个任务，如下所示：

```
iscsiadm -m discovery -t st -p target_IP -o delete -o new
```

sendtargets 命令将生成以下输出：

```
ip:port,target_portal_group_tag proper_target_name
```

例 25.13. sendtargets 命令的输出

例如，如果一个具有单个目标、逻辑单元和门户的设备，具有 **equallogic-iscsi1** 作为您的 **target_name**，输出应类似于如下：

```
10.16.41.155:3260,0 iqn.2001-05.com.equallogic:6-8a0900-ac3fe0101-63aff113e344a4a2-dl585-03-1
```

请注意，**proper_target_name** 和 **ip:port,target_portal_group_tag** 与 第 25.7.1 节 “iSCSI API” 中相同名称的值相同。

此时，您现在具有手动扫描 **iSCSI** 设备所需的正确 **--targetname** 和 **--portal** 值。为此，请运行以下命令：

```
# iscsiadm --mode node --targetname proper_target_name --portal ip:port,target_portal_group_tag \ --login
[8]
```

例 25.14. 完整的 iscsiadm 命令

使用我们前面的示例（其中 proper_target_name 是 equallogic-iscsi1），完整的命令将是：

```
# iscsiadm --mode node --targetname  \ iqn.2001-05.com.equallogic:6-8a0900-ac3fe0101-63aff113e344a4a2-dl585-03-1  \ --portal 10.16.41.155:3260,0 --login[8]
```

25.16. 登录到 ISCSI 目标

如 第 25.7 节 “iSCSI” 所述，必须运行 iSCSI 服务才能发现或登录到目标。要启动 iSCSI 服务，请运行：

```
# systemctl start iscsi
```

当执行这个命令时，iSCSI 初始化脚本 将自动登录到目标，其中 node.startup 设置被配置为 automatic。这是所有目标 node.startup 的默认值。

要防止自动登录到目标，将 node.startup 设置为 manual。要做到这一点，请运行以下命令：

```
# iscsiadm -m node --targetname proper_target_name -p target_IP:port -o update -n node.startup -v manual
```

删除整个记录也会阻止自动登录。为此，请运行：

```
# iscsiadm -m node --targetname proper_target_name -p target_IP:port -o delete
```

要从网络上的 iSCSI 设备自动挂载文件系统，请使用 _netdev 选项在 /etc/fstab 中为挂载添加一个分区条目。例如：要在启动时自动将 iSCSI 设备 sdb 挂载到 /mount/iscsi，请在 /etc/fstab 中添加以下行：

```
/dev/sdb /mnt/iscsi ext3 _netdev 0 0
```

要手动登录到 iSCSI 目标，请使用以下命令：

```
# iscsiadm -m node --targetname proper_target_name -p target_IP:port -l
```

注意

proper_target_name 和 target_IP:port 是指目标的完整名称和 IP 地址/端口组合。如需更多信息，请参阅 第 25.7.1 节 “iSCSI API” 和 第 25.15 节 “扫描 iSCSI Interconnects”。

25.17. 重新调整在线逻辑单元的大小

在大多数情况下，全面调整在线 逻辑单元 的大小涉及到两件事：调整逻辑单元本身的大小，并在相应的多路径设备中反映大小的变化（如果系统上启用了多路径）。

要调整在线逻辑单元的大小，请首先通过存储设备的阵列管理界面修改逻辑单元的大小。这个流程因每个阵列而有所不同，因此，请查阅您的存储阵列厂商文档来了解详情。

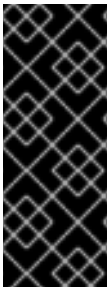
注意

为了调整在线文件系统的大小，文件系统不能在分区的设备上。

25.17.1. 重新调整光纤通道逻辑单元的大小

修改在线逻辑单元的大小后，重新扫描逻辑单元以确保系统可以检测到更新的大小。要为光纤通道逻辑单元执行此操作，请使用以下命令：

```
$ echo 1 > /sys/block/sdX/device/rescan
```



重要

要在使用多路径的系统上重新扫描光纤通道逻辑单元，请对代表多路径逻辑单元的路径的每个 sd 设备（如 sd1、sd2 等）执行上述命令。要确定哪个设备是多路径逻辑单元的路径，请使用 multipath -ll ；然后，找到与正在调整大小的逻辑单元匹配的条目。建议您参考每个条目的 WWID，以便更轻松地找到哪一个与正在调整大小的逻辑单元相匹配。

25.17.2. 重新调整 iSCSI 逻辑单元的大小

修改在线逻辑单元的大小后，重新扫描逻辑单元以确保系统可以检测到更新的大小。要为 iSCSI 设备执行此操作，请使用以下命令：

```
# iscsiadm -m node --targetname target_name -R
```
[5]

使用设备所在的目标的名称替换 target_name。

备注

您还可以使用以下命令重新扫描 iSCSI 逻辑单元：

```
# iscsiadm -m node -R -I interface
```

使用调整大小的逻辑单元的对应接口名称（如 iface0）替换 interface。此命令执行两个操作：

- 它使用与命令 echo "- - -" > /sys/class/scsi_host/host/scan 相同的方式扫描新设备（请参阅 第 25.15 节 “扫描 iSCSI Interconnects”）。
- 它使用与命令 echo 1 > /sys/block/sdX/device/rescan 相同的方式重新扫描新的/ 修改逻辑单元。请注意，这个命令与重新扫描光纤通道逻辑单元的命令相同。

25.17.3. 更新多路径设备的大小

如果系统上启用了多路径，您还需要将逻辑单元大小的变化反映到逻辑单元对应的多路径设备上（在调整逻辑单元的大小之后）。这可以通过 multipathd 完成。为此，首先使用 service multipathd status 确保 multipathd 正在运行。验证 multipathd 正常工作后，运行以下命令：

```
# multipathd -k"resize map multipath_device"
```

multipath_device 变量是 /dev/mapper 中设备的对应多路径条目。根据多路径在您的系统上是如何设置的，multipath_device 可以是两种格式之一：

- mpathX，其中 X 是设备的对应条目（例如 mpath0）
- WWID；例如 3600508b400105e210000900000490000

要确定哪个多路径条目对应于您调整了大小的逻辑单元，请运行 multipath -ll。这将显示系统中所有现有的多路径条目的列表，以及对应设备的主号和次号。

重要的

如果有任何命令排队到 multipath_device，则不要使用 multipathd -k"resize map multipath_device "。也就是说，当 no_path_retry 参数（在 /etc/multipath.conf中）设置为 "queue" 时，不要使用此命令，且设备没有主动路径。

有关多路径的更多信息，请参阅 Red Hat Enterprise Linux 7 DM 多路径 指南。

25.17.4. 更改在线逻辑单元的读/写状态

某些存储设备允许用户将设备状态从读/写(R/W)更改为只读(RO)，从 RO 更改为 R/W。这通常是通过存储设备上的管理界面来完成的。在进行更改时，操作系统不会自动更新其关于设备状态的视图。按照本章中所述的流程，让操作系统知道这些更改。

运行以下命令，将 XYZ 替换为所需的设备指示器，以确定操作系统对设备的 R/W 状态的当前视图：

```
# blockdev --getro /dev/sdXYZ
```

以下命令可用于 Red Hat Enterprise Linux 7：

```
# cat /sys/block/sdXYZ/ro 1 = read-only 0 = read-write
```

使用多路径时，请参考 multipath -ll 命令的输出的第二行中的 ro 或 rw 字段。例如：

```
36001438005deb4710000500000640000 dm-8 GZ,GZ500
[size=20G][features=0][hwhandler=0][ro]
\_ round-robin 0 [prio=200][active]
 \_ 6:0:4:1  sdax 67:16  [active][ready]
 \_ 6:0:5:1  sday 67:32  [active][ready]
\_ round-robin 0 [prio=40][enabled]
 \_ 6:0:6:1  sdaz 67:48  [active][ready]
 \_ 6:0:7:1  sdba 67:64  [active][ready]
```

要更改 R/W 状态，请使用以下流程：

过程 25.14. 更改 R/W 状态

1. 要将设备从 RO 移到 R/W，请参阅第 2 步。

   要将设备从 R/W 移到 RO，请确保不再发出写操作。通过停止应用程序，或使用合适的、特定于应用的操作来完成此操作。

   使用以下命令确保所有未完成的写 I/O 都完成了：

   ```
   # blockdev --flushbufs /dev/device
   ```

   使用所需的设计器替换 device ；对于设备映射器多路径，这是 dev/mapper 中您设备的条目。例如： /dev/mapper/mpath3。

2. 使用存储设备的管理界面，将逻辑单元的状态从 R/W 改为 RO，或者从 RO 改为 R/W。其流程因每个阵列而异。如需更多信息，请参阅适用的存储阵列供应商文档。

3. 执行设备的重新扫描，以更新操作系统的设备 R/W 状态视图。如果使用设备映射器多路径，请对设备的每个路径执行此重新扫描，然后再发出命令告诉多路径重新加载其设备映射。

   这一过程在 第 25.17.4.1 节 “重新扫描逻辑单元” 中进行了更详细的说明。

25.17.4.1. 重新扫描逻辑单元

修改在线逻辑单元读/写状态后，如 第 25.17.4 节 “更改在线逻辑单元的读/写状态” 所述，重新扫描逻辑单元以确保系统使用以下命令检测到更新的状态：

```
# echo 1 > /sys/block/sdX/device/rescan
```

要在使用多路径的系统上重新扫描逻辑单元，请对代表多路径逻辑单元路径的每个 sd 设备执行上述命令。例如，在 sd1、sd2 和所有其他 sd 设备上运行命令。要确定哪个设备是多路径单元的路径，请使用 multipath -11，然后找到与要更改的逻辑单元匹配的条目。

例 25.15. 使用 multipath -11 命令

例如，上面的 多路径 -11 显示 WWID 为 36001438005deb4710000500000640000 的 LUN 的路径。在这种情况下，请输入：

```
# echo 1 > /sys/block/sdax/device/rescan
# echo 1 > /sys/block/sday/device/rescan
# echo 1 > /sys/block/sdaz/device/rescan
# echo 1 > /sys/block/sdba/device/rescan
```

25.17.4.2. 更新多路径设备的 R/W 状态

如果启用了多路径，则重新扫描逻辑单元后，其状态的变化将需要反映在逻辑单元相应的多路径驱动器中。为此，请使用以下命令重新载入多路径设备映射：

```
# multipath -r
```

然后，可使用 multipath -11 命令确认更改。

25.17.4.3. Documentation

如需更多信息，请参阅红帽知识库。若要对此进行访问，请导航到 https://www.redhat.com/wapps/sso/login.html?redirect=https://access.redhat.com/knowledge/，并登录。然后在 https://access.redhat.com/kb/docs/DOC-32850 处访问文章。

25.18. 通过 RESCAN-SCSI-BUS.SH 添加/删除逻辑单元

sg3_utils 软件包提供 rescan-scsi-bus.sh 脚本，该脚本可以根据需要自动更新主机的逻辑单元配置（在设备添加到系统后）。rescan-scsi-bus.sh 脚本还可以对支持的设备执行 issue_lip。有关如何使用此脚本的更多信息，请参阅 rescan-scsi-bus.sh --help。

要安装 sg3_utils 软件包，请运行 yum install sg3_utils。

rescan-scsi-bus.sh 的已知问题

使用 rescan-scsi-bus.sh 脚本时，请注意以下已知问题：

- 要使 rescan-scsi-bus.sh 正常工作，LUN0 必须是第一个映射的逻辑单元。rescan-scsi-bus.sh 只能检测第一个映射的逻辑单元（如果它是 LUN0 ）。rescan-scsi-bus.sh 将无法扫描任何其他逻辑单元，除非它检测到第一个映射的逻辑单元，即使您使用了 --nooptscan 选项。
-

如果第一次映射逻辑单元，则竞争条件要求运行 rescan-scsi-bus.sh 两次。在第一次扫描过程中，rescan-scsi-bus.sh 仅添加 LUN0 ；在第二次扫描中添加所有其他逻辑单元。

- 当使用 --remove 选项时，rescan-scsi-bus.sh 脚本中的一个错误会错误地执行识别逻辑单元大小变化的功能。

- rescan-scsi-bus.sh 脚本无法识别 ISCSI 逻辑单元删除。

25.19. 修改链接丢失行为

这部分论述了如何修改使用光纤通道或 iSCSI 协议的设备的链接丢失行为。

25.19.1. Fibre Channel

如果驱动程序实现了传输 dev_loss_tmo 回调，当检测到传输问题时，通过链接访问设备的尝试将被阻止。要验证设备是否被阻止，请运行以下命令：

```
$ cat /sys/block/device/device/state
```

如果设备被阻止，这个命令将返回 blocked。如果设备正常运行，这个命令会返回 running。

过程 25.15. 确定远程端口的状态

1. 要确定远程端口的状态，请运行以下命令：

```
$ cat
/sys/class/fc_remote_port/rport-H:B:R/port_state
```

2. 当远程端口（以及通过它访问的设备）被阻止时，这个命令将返回 Blocked。如果远程端口正常运行，命令将返回 Online。

3. 如果没有在 dev_loss_tmo 秒内解决问题，则 rport 和设备将被取消阻塞，且在该设备上运行的所有 I/O （以及发送到该设备的任何新的 I/O）都将失败。

过程 25.16. 更改 dev_loss_tmo

- 要更改 dev_loss_tmo 值，请将所需值 echo 到该文件。例如，要将 dev_loss_tmo 设置为

30 秒，请运行：

```
$ echo 30 >
/sys/class/fc_remote_port/rport-H:B:R/dev_loss_tmo
```

有关 dev_loss_tmo 的更多信息，请参阅 第 25.4.1 节 “光纤通道 API”。

当链接丢失超过 dev_loss_tmo 时，会删除 scsi_device 和 sdN 设备。通常，光纤通道类会将设备保留原样；例如 /dev/sdx 将保持 /dev/sdx。这是因为，目标绑定由 Fibre Channel 驱动程序保存，因此当目标端口返回时，SCSI 地址会被重新创建。但是，无法保证这一点；只有在未进行 LUN 的存储框中配置时，才会恢复 sdx。

25.19.2. 使用 dm-multipath的 iSCSI 设置

如果实现了 dm-multipath，建议设置 iSCSI 计时器，以立即将命令延迟到多路径层。要配置此功能，请在 /etc/multipath.conf 中的 device { 下嵌套以下行：

```
features  "1 queue_if_no_path"
```

这样可确保在 dm-multipath 层中所有路径都失败时重试 I/O 错误并排队。

您可能需要进一步调整 iSCSI 计时器，以更好地监控 SAN 是否存在问题。您可以配置的可用 iSCSI 计时器为 NOP-Out Interval/Timeouts 和 replacement_timeout，后续章节将对此进行讨论。

25.19.2.1. NOP-Out Interval/Timeout

为了帮助监控 SAN 的问题，iSCSI 层向每个目标发送 NOP-Out 请求。如果 NOP-Out 请求超时，iSCSI 层将通过使任何正在运行的命令失败而做出响应，并指示 SCSI 层尽可能重新排队这些命令。

当使用 dm-multipath 时，SCSI 层将失败那些运行命令并将其延迟到多路径层。然后多路径层会对另一个路径重新尝试这些命令。如果没有使用 dm-multipath，则这些命令会在完全失败前重试五次。

NOP-Out 请求之间的间隔默认为 5 秒。要调整此功能，请打开 /etc/iscsi/iscsid.conf 并编辑以下行：

```
node.conn[0].timeo.noop_out_interval = [interval value]
```

设置之后，iSCSI 层将每隔 [interval value] 秒向每个目标发送 NOP-Out 请求。

默认情况下，NOP-Out 请求超时(5 秒)[9].要调整此功能，请打开 /etc/iscsi/iscsid.conf 并编辑以下行：

```
node.conn[0].timeo.noop_out_timeout = [timeout value]
```

这会将 iSCSI 层设置为在 [timeout value] 秒后使 NOP-Out 请求超时。

SCSI 错误处理程序

如果 SCSI 错误处理程序正在运行，当 NOP-Out 请求在该路径上超时时，在路径上运行命令不会立即失败。相反，这些命令将在 replacement_timeout 秒后 失败。有关 replacement_timeout 的详情，请参考 第 25.19.2.2 节 “replacement_timeout”。

要验证 SCSI 错误处理程序是否正在运行，请运行：

```
# iscsiadm -m session -P 3
```

25.19.2.2. replacement_timeout

replacement_timeout 控制 iSCSI 层应等待超时路径/会话重新建立自己的时间，然后再在其上失败任何命令。默认的 replacement_timeout 值为 120 秒。

要调整 replacement_timeout，请打开 /etc/iscsi/iscsid.conf 并编辑以下行：

```
node.session.timeo.replacement_timeout = [replacement_timeout]
```

/etc/multipath.conf 中的 1 queue_if_no_path 选项设置 iSCSI 计时器，以立即将命令延迟到多路径层（请参考 第 25.19.2 节 “使用 dm-multipath的 iSCSI 设置”）。此设置可防止 I/O 错误传播到应用程序；因此，您可以将 replacement_timeout 设置为 15-20 秒。

通过配置较低 replacement_timeout，I/O 会快速发送到新路径并执行（如果发生 NOP-Out 超时），而 iSCSI 层会尝试重新建立失败的路径/会话。如果所有路径都超时，则多路径和设备映射器层将根据 /etc/multipath.conf 中的设置而不是 /etc/iscsi/iscsid.conf 中的设置在内部排队 I/O。

重要的

无论您的考虑是故障转移速度还是安全性，推荐的 replacement_timeout 值取决于其他因素。这些因素包括网络、目标和系统工作负载。因此，建议您在将其应用到关键任务系统之前，对 replacements_timeout 的任何新配置进行彻底测试。

iSCSI 和 DM 多路径覆盖

restore_tmo sysfs 选项控制一个特定 iSCSI 设备的超时时间。以下选项会在全局范围内覆盖 recovery_tmo 值：

- replacement_timeout 配置选项会全局覆盖所有 iSCSI 设备的 recovery_tmo 值。
- 对于由 DM 多路径管理的所有 iSCSI 设备，DM 多路径中的 fast_io_fail_tmo 选项会全局覆盖 recovery_tmo 值。DM 多路径中的 fast_io_fail_tmo 选项会覆盖光纤通道设备的 fast_io_fail_tmo 选项。

DM 多路径 fast_io_fail_tmo 选项优先于 replacement_timeout。红帽不推荐使用 replacement_timeout 覆盖由 DM 多路径管理的设备中的 recovery_tmo，因为在 multipathd 服务重新载入时 DM 多路径总是重置 recovery_tmo。

25.19.3. iSCSI 根

当直接通过 iSCSI 磁盘访问根分区时，应设置 iSCSI 定时器，以便 iSCSI 层有多个机会尝试重新建立路径/会话。此外，命令不应快速重新排队到 SCSI 层。这与实现 dm-multipath 时应该执行的操作相反。

首先，应禁用 NOP-Outs。您可以通过将 NOP-Out 间隔和超时设置为零来实现此操作。要设置此设置，请打开 /etc/iscsi/iscsid.conf 并编辑如下：

```
node.conn[0].timeo.noop_out_interval = 0
node.conn[0].timeo.noop_out_timeout = 0
```

与此相符，应将 replacement_timeout 设置为高数字。这将指示系统等待很长时间来重新建立路径/会话。要调整 replacement_timeout，请打开 /etc/iscsi/iscsid.conf 并编辑以下行：

```
node.session.timeo.replacement_timeout = replacement_timeout
```

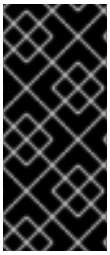
配置 /etc/iscsi/iscsid.conf 后，您必须对受影响的存储执行重新发现。这将允许系统加载和使用 /etc/iscsi/iscsid.conf 中的任何新值。有关如何发现 iSCSI 设备的详情，请参考 第 25.15 节 “扫描 iSCSI

Interconnects”。

为特定会话配置超时

您还可以为特定会话配置超时，并使它们成为非持久性（而不是使用 /etc/iscsi/iscsid.conf）。为此，请运行以下命令（相应地替换变量）：

```
# iscsiadm -m node -T target_name -p target_IP:port -o update -n
node.session.timeo.replacement_timeout -v $timeout_value
```



重要的

对于涉及根分区访问的 iSCSI 会话，建议使用此处描述的配置。对于涉及访问其他类型的存储的 iSCSI 会话（例如，在使用 dm-multipath的系统中），请参阅 第 25.19.2 节 “使用 dm-multipath的 iSCSI 设置”。

25.20. 控制 SCSI 命令计时器和设备状态

Linux SCSI 层对每个命令设置一个计时器。当这个计时器到期时，SCSI 层将停止 主机总线适配器 (HBA)，并等待所有未执行的命令超时或完成。之后，SCSI 层将激活驱动程序的错误处理程序。

触发错误处理程序时，它会按顺序尝试以下操作（直到成功执行了一个操作）：

1. 中止命令。
2. 重置设备。
3. 重置总线。
4. 重置主机。

如果所有这些操作都失败，则该设备将设置为 离线状态。当发生这种情况时，到该设备的所有 I/O 都将失败，直到问题被修正，用户会将设备设置为 运行。

但是，如果设备使用 Fibre Channel 协议，且 rport 被阻止，则此过程会有所不同。在这种情况下，

驱动程序会在激活错误处理程序前等待几秒钟，以使 rport 再次上线。这可防止设备因为临时传输问题而离线。

设备状态

要显示设备的状态，请使用：

```
$ cat /sys/block/device-name/device/state
```

要将设备设置为 running 状态，请使用：

```
# echo running > /sys/block/device-name/device/state
```

命令计时器

要控制命令计时器，请修改 /sys/block/device-name/device/timeout 文件：

```
# echo value > /sys/block/device-name/device/timeout
```

使用您要实现的超时值（以秒为单位）替换命令中的 value。

25.21. 在线存储配置故障排除

本节提供了在线存储重新配置过程中用户遇到的常见问题的解决方案。

逻辑单元删除状态不会在主机上反映。

在配置的文件上删除逻辑单元时，该更改不会在主机上反映。在这种情况下，当使用 dm-multipath 时 lvm 命令会无限期挂起，因为逻辑单元现已过时。

要临时解决这个问题，请执行以下步骤：

过程 25.17. 围绕过时的逻辑单元进行工作

1. 确定 /etc/lvm/cache/.cache 中的哪些 mpath 链接条目特定于过时的逻辑单元。要做到这一点，请运行以下命令：

```
$ ls -l /dev/mpath | grep stale-logical-unit
```

例 25.16. 确定特定 mpath 链接条目

例如，如果 **stale-logical-unit 是 3600d0230003414f30000203a7bc41a00，则可能会出现以下结果：**

```
lrwxrwxrwx 1 root root 7 Aug  2 10:33 /3600d0230003414f30000203a7bc41a00 ->
../dm-4
lrwxrwxrwx 1 root root 7 Aug  2 10:33 /3600d0230003414f30000203a7bc41a00p1 ->
../dm-5
```

这意味着 **3600d0230003414f30000203a7bc41a00 被映射到两个 mpath 链接：dm-4 和 dm-5。**

2.

接下来，打开 /etc/lvm/cache/.cache。删除包含 stale-logical-unit 和 stale-logical-unit 映射到的 mpath 链接的所有行。

例 25.17. 删除相关行

使用上一步中的同一示例，您需要删除的行是：

```
/dev/dm-4
/dev/dm-5
/dev/mapper/3600d0230003414f30000203a7bc41a00
/dev/mapper/3600d0230003414f30000203a7bc41a00p1
/dev/mpath/3600d0230003414f30000203a7bc41a00
/dev/mpath/3600d0230003414f30000203a7bc41a00p1
```

25.22. 使用 EH_DEADLINE 为错误恢复配置最大时间

重要

在大多数情况下，您不需要启用 eh_deadline 参数。在某些特定情况下，使用 eh_deadline 参数很有用，例如，如果在光纤通道交换机和目标端口之间发生链接丢失，主机总线适配器(HBA)不会接收 Registered State Change Notifications (RSCN)。在这种情况下，I/O 请求和错误恢复命令会超时，而不是遇到错误。在这个环境中的 eh_deadline 设置会为恢复时间设置上限，它允许在多路径的另一个可用路径中重试失败的 I/O。

但是，如果启用了 RSCN，则 HBA 不会注册链接不可用，或者 eh_deadline 功能不会提供额外的好处，因为 I/O 和错误恢复命令会立即失败，这允许多路径重试。

SCSI 主机对象 eh_deadline 参数允许您配置 SCSI 错误处理机制在停止和重置整个 HBA 前尝试执行错误恢复的最长时间。

eh_deadline 的值以秒为单位指定。默认设置为 off，它会禁用时间限制并允许进行所有错误恢复。除了使用 sysfs 外，还可使用 scsi_mod.eh_deadline 内核参数为所有 SCSI HBA 设置默认值。

请注意，当 eh_deadline 过期时，HBA 被重置，这会影响那个 HBA 中的所有目标路径，而不仅仅是失败的路径。因此，如果某些冗余路径因其他原因不可用，则可能会出现 I/O 错误。仅在所有目标中有完全冗余的多路径配置时才启用 eh_deadline。

---

[5] *target_IP 和 port 变量分别指向目标门户的 IP 地址和端口组合。如需更多信息，请参阅 第 25.7.1 节 “iSCSI API” 和 第 25.15 节 “扫描 iSCSI Interconnects”。*

[6] *Refer to 第 25.15 节 “扫描 iSCSI Interconnects” for information on proper_target_name.*

[7] *For information on how to retrieve a session's SID value, refer to 第 25.7.1 节 “iSCSI API”.*

[8] *This is a single command split into multiple lines, to accommodate printed and PDF versions of this document. All concatenated lines — preceded by the backslash (\) — should be treated as one command, sans backslashes.*

[9] *在 Red Hat Enterprise Linux 5.4 之前，默认的 NOP-Out 请求超时时间为 15 秒。*

第 26 章 虚拟机的设备映射器多路径(DM 多路径)和存储

Red Hat Enterprise Linux 7 支持虚拟机的 DM 多路径 和存储。

26.1. 虚拟机存储

Red Hat Enterprise Linux 7 支持以下文件系统或虚拟存储的在线存储方法：

- Fibre Channel
- iSCSI
- NFS
- GFS2

Red Hat Enterprise Linux 7 中的虚拟化使用 libvirt 管理虚拟实例。libvirt 工具使用 存储池 的概念来管理虚拟客户机的存储。存储池是一种可以划分为较小的卷或直接分配给客户机的存储。存储池的卷可以分配给虚拟客户机。以下是可用存储池的类别：

本地存储池

本地存储包括直接附加到主机、本地目录、直接附加的磁盘和 LVM 卷组的存储设备、文件或目录。

联网的（共享）存储池

网络存储涵盖使用标准协议通过网络共享的存储设备。它包括使用光纤通道、iSCSI、NFS、GFS2 和 SCSI RDMA 协议的共享存储设备，是在主机之间迁移虚拟客户机的要求。

有关 Red Hat Enterprise Linux 支持的存储池类型的更多信息，请参阅 Red Hat Virtualization 部署

和管理指南。

26.2. DM MULTIPATH

DM 多路径是一种功能，允许您将服务器节点和存储阵列之间的多个 I/O 路径配置为单一设备。这些 I/O 路径是可包含独立电缆、交换机和控制器的物理 SAN 连接。多路径将 I/O 路径聚合在一起，从而创建一个由聚合路径组成的新设备。

DM 多路径主要用于以下原因：

冗余

DM 多路径可在主动/被动（active/passive）配置中提供故障切换。在主动/被动配置中，对于 I/O，任何时候都只会使用一半的路径。如果 I/O 路径的任何元素失败，DM 多路径会切换到备用路径。

改进的性能

可将 DM 多路径配置为主动/主动模式，其中 I/O 以轮循方式分布到路径中。在某些配置中，DM-Multipath 可以检测 I/O 路径上的加载，并动态重新平衡负载。

如需更多信息，请参阅 Red Hat DM 多路径 指南。

第 27 章 外部阵列管理(LIBSTORAGEMGMT)

Red Hat Enterprise Linux 7 附带一个新的外部阵列管理库，名为 libStorageMgmt。

27.1. LIBSTORAGEMGMT 简介

libStorageMgmt 库是一个独立于应用程序编程接口(API)的存储阵列。作为开发者，您可以使用此 API 管理不同的存储阵列，并利用硬件加速功能。

这个库被用作其他更高级别的管理工具和应用程序的一个构建块。最终系统管理员还可以将它用作工具，通过利用脚本手动管理存储并自动执行存储管理任务。

使用 libStorageMgmt 库，您可以执行以下操作：

- 列出存储池、卷、访问组或文件系统。
- 创建和删除卷、访问组、文件系统或 NFS 导出。
- 授予和移除对卷、访问组或启动器的访问权限。
- 使用快照、克隆和副本复制卷。
- 创建和删除访问组，并编辑组成员。

未使用服务器资源，如 CPU 和互连带宽，因为操作都在数组上完成。

libstoragemgmt 软件包提供：

- 用于客户端应用程序和插件开发人员的稳定 C 和 Python API。

- 使用库的命令行界面(lsmcli)。
- 执行插件(lsmd)的守护进程。
- 允许测试客户端应用程序(sim)的模拟插件。
- 用于与数组交互的插件架构。

警告

这个库及其关联的工具能够销毁它所管理的数组上的任何及所有数据。强烈建议您针对存储模拟器插件开发和测试应用程序和脚本，以便在使用生产系统前删除任何逻辑错误。在部署到生产环境前，强烈建议在实际非生产硬件上测试应用程序和脚本（如果可能）。

Red Hat Enterprise Linux 7 中的 libStorageMgmt 库添加了默认的 udev 规则，以处理 REPORTED LUNS DATA HAS CHANGED 单元注意。

当存储配置更改发生时，其中一个单元 Attention ASC/ASCQ 代码会报告更改。然后会生成 uevent，并使用 sysfs 自动扫描。

文件 /lib/udev/rules.d/90-scsi-ua.rules 包含示例规则，用于枚举内核可生成的其他事件。

libStorageMgmt 库使用插件架构来容纳存储阵列的不同。有关 libStorageMgmt 插件以及如何编写它们的更多信息，请参阅 Red Hat Developer Guide。

27.2. LIBSTORAGEMGMT TERMINOLOGY

不同的阵列供应商和存储标准使用不同的术语来指代类似的功能。这个库使用以下术语：

存储阵列

任何通过网络连接存储(NAS)提供块访问(FC、 FCoE、 iSCSI)或文件访问的存储系统。

卷

存储区域网络(SAN)存储阵列可以通过不同的传输（如 FC、 iSCSI 或 FCoE）向主机总线适配器(HBA)公开卷。主机操作系统将其视为块设备。如果启用了 多路径[2]， 一个卷可以公开给多个磁盘。

这也被称为逻辑单元号(LUN)， 带有 SNIA 术语的 StorageVolume 或虚拟磁盘。

pool

一组存储空间。可以从池中创建文件系统或卷。池可以从磁盘、卷和其他池创建。池也可以保存 RAID 设置或精简配置设置。

这也称为带有 SNIA 术语的 StoragePool。

Snapshot

时间点，即只读、空间高效数据副本。

这也称为只读快照。

克隆

时间点、 可读取、空间高效数据副本。

这也被称为可读写快照。

Copy

数据的全位副本。它占据了完整的空间。

mirror

持续更新的副本（同步和异步）。

访问组

被授予对一个或多个存储卷访问权限的 iSCSI、FC 和 FCoE 启动器的集合。这样可确保只有存储卷可以被指定的启动器访问。

这也被称为 initiator 组。

access Grant

将卷公开给指定的访问组或启动器.libStorageMgmt 库目前不支持 LUN 映射，能够选择特定的逻辑单元号。libStorageMgmt 库允许存储阵列选择下一个可用的 LUN 进行分配。如果配置从 SAN 或屏蔽超过 256 个卷的引导，请确定读取 OS、存储阵列或 HBA 文档。

访问权限授权也称为 LUN 掩码。

System

代表存储阵列或直接连接的存储 RAID。

File system

网络附加存储(NAS)存储阵列可使用 NFS 或 CIFS 协议通过 IP 网络公开文件系统来托管操作系统。主机操作系统将其视为挂载点或包含文件的文件夹，具体取决于客户端操作系统。

磁盘

保存数据的物理磁盘。这通常在使用 RAID 设置创建池时使用。

这也称为 DiskDrive，使用 SNIA 术语。

initiator

在光纤通道(FC)或以太网光纤通道(FCoE)中，发起方是全球端口名称(WWPN)或全球节点名称(WWNN)。在 iSCSI 中，启动器是 iSCSI 限定名称(IQN)。在 NFS 或 CIFS 中，启动器是主机名或主机的 IP 地址。

子依赖项

有些阵列在原始卷（父卷或文件系统）和子（如快照或克隆）之间有一个隐式关系。例如，如果父对象有一个或多个依赖的子项，则无法删除父项。API 提供了方法来确定是否存在此类关系，以及

通过复制所需块来删除依赖项的方法。

27.3. INSTALLING LIBSTORAGEMGMT

要安装 libStorageMgmt 以使用命令行，所需的运行时库和 simulator 插件，请使用以下命令：

```
# yum install libstoragemgmt libstoragemgmt-python
```

要开发使用库的 C 应用程序，请使用以下命令安装 libstoragemgmt-devel 软件包：

```
# yum install libstoragemgmt-devel
```

要安装 libStorageMgmt 用于硬件数组，请使用以下命令选择一个或多个适当的插件软件包：

```
# yum install libstoragemgmt-name-plugin
```

可用的插件包括：

libstoragemgmt-smis-plugin

通用 SMI-S 阵列支持。

libstoragemgmt-netapp-plugin

对 NetApp 文件的特定支持。

libstoragemgmt-nstor-plugin

对 NexentaStor 的特定支持。

libstoragemgmt-targetd-plugin

对目标的特定支持。

然后，守护进程会被安装，并配置为在下一次重启后在启动时运行。要在不重新启动的情况下立即使用它，请手动启动守护进程。

管理数组需要通过插件提供支持。基本安装软件包包括多个不同供应商的开源插件。其他插件软件包将单独提供，因为阵列支持有所改进。目前支持的硬件不断变化和改进。

libStorageMgmt 守护进程(lsmd)的行为与系统的任何标准服务类似。

检查 libStorageMgmt 服务的状态：

```
# systemctl status libstoragemgmt
```

停止该服务：

```
# systemctl stop libstoragemgmt
```

启动该服务：

```
# systemctl start libstoragemgmt
```

27.4. 使用 LIBSTORAGEMGMT

要以交互方式使用 libStorageMgmt，请使用 lsmcli 工具。

lsmcli 工具需要两个项才能运行：

- Uniform Resource Identifier (URI)，用于识别插件以连接到该数组以及该阵列所需的任何可配置的选项。
- 数组的有效用户名和密码。

URI 的格式如下：

plugin+optional-transport://user-name@host:port/?query-string-parameters

每个插件对需要的内容有不同的要求。

> *例 27.1. 不同插件要求示例*
>
> *simulator 插件需要没有用户名或密码*
>
> *sim://*
>
> *NetApp 插件使用用户名 root 进行 SSL*
>
> *ontap+ssl://root@filer.company.com/*
>
> *用于 EMC Array 的 SSL 的 SMI-S 插件*
>
> *smis+ssl://admin@provider.com:5989/?namespace=root/emc*

使用 URI 有三个选项：

1. *将 URI 作为命令的一部分传递。*

```
$ lsmcli -u sim://
```

2. *将 URI 存储在环境变量中。*

```
$ export LSMCLI_URI=sim://
```

3. *将 URI 放在文件 ~/.lsmcli 中，其中包含用 "=" 分隔的名称值对。当前唯一支持的配置是 'uri'。*

确定需要按照以下顺序使用哪个 URI。如果提供了所有三个命令，则只会在命令行中使用第一个。

在命令行中指定 -P 选项或将其放在环境变量 LSMCLI_PASSWORD 中提供密码。

例 27.2. lsmcli 示例

例如，使用命令行创建新卷并使其对启动器可见。

列出此连接提供服务的数组：

```
$ lsmcli list --type SYSTEMS
ID     | Name                          | Status
-------+-------------------------------+--------
sim-01 | LSM simulated storage plug-in | OK
```

列出存储池：

```
$ lsmcli list --type POOLS -H
ID   | Name         | Total space          | Free space           | System ID
-----+--------------+----------------------+----------------------+----------
POO2 | Pool 2       | 18446744073709551616 | 18446744073709551616 | sim-01
POO3 | Pool 3       | 18446744073709551616 | 18446744073709551616 | sim-01
POO1 | Pool 1       | 18446744073709551616 | 18446744073709551616 | sim-01
POO4 | lsm_test_aggr | 18446744073709551616 | 18446744073709551616 | sim-01
```

创建卷：

```
$ lsmcli volume-create --name volume_name --size 20G --pool POO1 -H
ID   | Name        | vpd83                            | bs  | #blocks  | status | ...
-----+-------------+----------------------------------+-----+----------+--------+----
Vol1 | volume_name | F7DDF7CA945C66238F593BC38137BD2F | 512 | 41943040 | OK     | ...
```

使用 iSCSI 启动器在其中创建访问组：

```
$ lsmcli --create-access-group example_ag --id iqn.1994-05.com.domain:01.89bd01 --type ISCSI
--system sim-01
ID                               | Name       | Initiator ID                     |SystemID
---------------------------------+------------+----------------------------------+--------
782d00c8ac63819d6cca7069282e03a0 | example_ag | iqn.1994-05.com.domain:01.89bd01 |sim-
01
```

在其中创建一个具有 iSCSI 问题的访问组：

```
$ lsmcli access-group-create --name example_ag --init iqn.1994-05.com.domain:01.89bd01 --init-type ISCSI --sys sim-01
ID                               | Name       | Initiator IDs                    | System ID
---------------------------------+------------+----------------------------------+-----------
782d00c8ac63819d6cca7069282e03a0 | example_ag | iqn.1994-05.com.domain:01.89bd01 | sim-01
```

允许访问组对新创建的卷可见：

```
$ lsmcli access-group-grant --ag 782d00c8ac63819d6cca7069282e03a0 --vol Vol1 --access RW
```

库设计为客户端和插件之间通过进程间通信(IPC)在 进程间通信(IPC)之间的进程分离提供进程 分离。这可防止插件中的错误使客户端应用程序崩溃。它还提供了一种使用自己选择的许可证编写插件的方法。当客户端打开传递 URI 的库时，客户端库将查看 URI 来确定应使用哪个插件。

该插件由技术上独立使用，但它们设计为在命令行上传递了文件描述符。然后，客户端库会打开适当的 Unix 域套接字，这会导致守护进程分叉并执行插件。这可让客户端库指向与插件的通信频道。守护进程可以在不影响现有客户端的情况下重启。当客户端为该插件打开了库时，插件进程正在运行。发送一个或多个命令且插件关闭后，插件过程会清理，然后退出。

lsmcli 的默认行为是等待操作完成。根据请求的操作，这可能需要几小时时间。要允许返回正常使用，可以在命令行中使用 -b 选项。如果退出代码为 0，则代表命令已完成。如果退出代码为 7，命令正在进行，作业标识符将写入标准输出。然后，用户或脚本可以根据需要使用 lsmcli --jobstatus JobID 来查询命令的状态。如果作业现已完成，则退出值为 0，结果将输出到标准输出。如果命令仍在进行中，返回值为 7，且完成百分比将打印到标准输出中。

例 27.3. Asynchronous 示例

通过 -b 选项创建卷，以便命令立即返回。

```
$ lsmcli volume-create --name async_created --size 20G --pool POO1 -b JOB_3
```

检查退出值：

```
$ echo $?
7
```

7 表示作业仍在进行中。

检查作业是否已完成：

```
$ lsmcli job-status --job JOB_3
33
```

检查退出值。7 表示作业仍在进行中，因此标准输出是根据给定屏幕完成的百分比或 33%。

```
$ echo $?
7
```

等待一些时间，然后再次检查退出值：

```
$ lsmcli job-status --job JOB_3
ID   | Name          | vpd83                            | Block Size  | ...
-----+---------------+----------------------------------+-------------+-----
Vol2 | async_created | 855C9BA51991B0CC122A3791996F6B15 | 512         | ...
```

0 表示成功，而标准输出会显示新卷。

对于脚本脚本，请传递 **-t SeparatorCharacters** 选项。这样可以更轻松地解析输出。

例 27.4. 脚本脚本示例

```
$ lsmcli list --type volumes -t#
Vol1#volume_name#049167B5D09EC0A173E92A63F6C3EA2A#512#41943040#21474836480#O
K#sim-01#POO1
Vol2#async_created#3E771A2E807F68A32FA5E15C235B60CC#512#41943040#21474836480#O
K#sim-01#POO1
```

```
$ lsmcli list --type volumes -t " | "
Vol1 | volume_name | 049167B5D09EC0A173E92A63F6C3EA2A | 512 | 41943040 |
21474836480 | OK | 21474836480 | sim-01 | POO1
Vol2 | async_created | 3E771A2E807F68A32FA5E15C235B60CC | 512 | 41943040 |
21474836480 | OK | sim-01 | POO1
```

```
$ lsmcli list --type volumes -s
----------------------------------------
ID         | Vol1
Name       | volume_name
VPD83      | 049167B5D09EC0A173E92A63F6C3EA2A
Block Size | 512
#blocks    | 41943040
```

```
Size       | 21474836480
Status     | OK
System ID  | sim-01
Pool ID    | POO1
----------------------------------------------
ID         | Vol2
Name       | async_created
VPD83      | 3E771A2E807F68A32FA5E15C235B60CC
Block Size | 512
#blocks    | 41943040
Size       | 21474836480
Status     | OK
System ID  | sim-01
Pool ID    | POO1
----------------------------------------------
```

建议您使用 Python 库进行非缓解脚本。

有关 lsmcli 的更多信息，请参阅 lsmcli 手册页或 lsmcli --help。

第 28 章 持久性内存：NVDIMM

持久内存(pmem)也称为存储类内存，是内存和存储的组合。PMEM 将存储的持久性与低访问延迟和动态 RAM (DRAM)的高带宽相结合：

- 持久内存是字节地址的，因此可以使用 CPU 负载和存储指令访问它。除了访问传统的基于块的存储所需的 read() 或 write() 系统调用外，pmem 还支持直接加载和存储编程模型。
- 持久内存的性能特征与具有非常低访问延迟的 DRAM 类似，通常以十到百纳秒为单位。
- 关闭电源（如存储）时会保留持久内存的内容。

对于以下用例，使用持久性内存很有用：

快速启动：数据集已在内存中。

快速开始也被称为 温缓存 效果。文件服务器启动后，内存中没有文件内容。当客户端连接和读取和写入数据时，这些数据会缓存在页面缓存中。最后，缓存包括大多数热数据。重启后，系统必须再次启动该进程。

如果应用设计正确，持久内存允许应用在重启后保留热缓存。在本实例中，不会涉及页面缓存：应用程序会直接在持久内存中缓存数据。

快速写缓存

在数据位于持久介质前，文件服务器通常不会确认客户端的写入请求。将持久内存用作快速写入缓存可让文件服务器快速确认写入请求，因为 pmem 的低延迟。

NVDIMMs Interleaving

非易失性双内存模块(NVDIMM)可以按照与常规 DRAM 相同的方式分组为交集。interleave 集与跨多个 DIMM 的 RAID 0 （条带）类似。

以下是 NVDIMMS 交集的优点：

- 与 DRAM 一样，NVDIMM 会在配置为 interleave 集时从性能提高。

- 它可用于将多个较小的 NVDIMM 组合为一个较大的逻辑设备。

使用系统 BIOS 或 UEFI 固件配置 interleave 集。

在 Linux 中，每个交集都会创建一个区域设备。

以下是区域设备和标签之间的关系：

- 如果您的 NVDIMM 支持标签，则区域设备可以进一步划分为命名空间。
- 如果您的 NVDIMM 不支持标签，则区域设备只能包含一个命名空间。在这种情况下，内核会创建一个覆盖整个区域的默认命名空间。

持久性内存访问模式

您可以使用 扇区、fsdax、devdax （设备直接访问）或 raw 模式中的持久性内存设备：

扇区模式

它将存储显示为一个快速块设备。使用扇区模式对于尚未修改以使用持久内存的传统应用程序或利用完整 I/O 堆栈的应用程序（包括设备映射器）非常有用。

fsdax 模式

它允许持久内存设备支持直接访问编程，如存储网络行业关联(SNIA)非易失性内存(NVM)编程模型规格 中所述。在这个模式中，I/O 会绕过内核的存储堆栈，因此无法使用很多设备映射器驱动程序。

devdax 模式

devdax (device DAX)模式通过使用 DAX 字符设备节点提供对持久内存的原始访问。可以使用 CPU 缓存清除和隔离指令，使 devdax 设备中的数据可用。某些数据库和虚拟机虚拟机监控程序可能

会受益于 devdax 模式。无法在 devdax 实例上创建文件系统。

原始模式

原始模式命名空间有几个限制，不应使用。

28.1. 使用 NDCTL 配置持久内存

使用 ndctl 工具配置持久内存设备。要安装 ndctl 工具，请使用以下命令：

```
# yum install ndctl
```

过程 28.1. 为不支持标签的设备配置持久性内存

1. 列出系统上可用的 pmem 区域。在以下示例中，命令列出了不支持标签的 NVDIMM-N 设备：

```
# ndctl list --regions
[
  {
    "dev":"region1",
    "size":34359738368,
    "available_size":0,
    "type":"pmem"
  },
  {
    "dev":"region0",
    "size":34359738368,
    "available_size":0,
    "type":"pmem"
  }
]
```

Red Hat Enterprise Linux 为每个区域创建一个 default 命名空间，因为这里的 NVDIMM-N 设备不支持标签。因此，可用大小为 0 字节。

2. 列出系统中所有不活跃的命名空间：

```
# ndctl list --namespaces --idle
[
  {
    "dev":"namespace1.0",
```

```
    "mode":"raw",
    "size":34359738368,
    "state":"disabled",
    "numa_node":1
  },
  {
    "dev":"namespace0.0",
    "mode":"raw",
    "size":34359738368,
    "state":"disabled",
    "numa_node":0
  }
]
```

3. ***重新配置不活跃的命名空间，以使用此空间。例如，要将 namespace0.0 用于支持 DAX 的文件系统，请使用以下命令：***

```
# ndctl create-namespace --force --reconfig=namespace0.0 --mode=fsdax --map=mem
{
  "dev":"namespace0.0",
  "mode":"fsdax",
  "size":"32.00 GiB (34.36 GB)",
  "uuid":"ab91cc8f-4c3e-482e-a86f-78d177ac655d",
  "blockdev":"pmem0",
  "numa_node":0
}
```

过程 28.2. 为支持标签的设备配置持久性内存

1. ***列出系统上可用的 pmem 区域。在以下示例中，命令列出了支持标签的 NVDIMM-N 设备：***

```
# ndctl list --regions
[
  {
    "dev":"region5",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-7337419320239190016
  },
  {
    "dev":"region4",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-137289417188962304
  }
]
```

2.

如果 NVDIMM 设备支持标签，则不会创建默认命名空间，您可以在不使用 --force 或 --reconfigure 标志的情况下从区域分配一个或多个命名空间：

```
# ndctl create-namespace --region=region4 --mode=fsdax --map=dev --size=36G
{
  "dev":"namespace4.0",
  "mode":"fsdax",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"9c5330b5-dc90-4f7a-bccd-5b558fa881fe",
  "blockdev":"pmem4",
  "numa_node":0
}
```

现在，您可以从同一区域创建另一个命名空间：

```
# ndctl create-namespace --region=region4 --mode=fsdax --map=dev --size=36G
{
  "dev":"namespace4.1",
  "mode":"fsdax",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"91868e21-830c-4b8f-a472-353bf482a26d",
  "blockdev":"pmem4.1",
  "numa_node":0
}
```

您还可以使用以下命令从同一区域创建不同类型的命名空间：

```
# ndctl create-namespace --region=region4 --mode=devdax --align=2M --size=36G
{
  "dev":"namespace4.2",
  "mode":"devdax",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"a188c847-4153-4477-81bb-7143e32ffc5c",
  "daxregion":
  {
    "id":4,
    "size":"35.44 GiB (38.05 GB)",
    "align":2097152,
    "devices":[
      {
        "chardev":"dax4.2",
        "size":"35.44 GiB (38.05 GB)"
      }]
  },
    "numa_node":0
}
```

有关 ndctl 工具的更多信息，请参阅 man ndctl。

28.2. 配置永久内存以用作块设备(LEGACY 模式)

要将持久内存用作快速块设备，请将命名空间设置为扇区模式。

```
# ndctl create-namespace --force --reconfig=namespace1.0 --mode=sector
{
  "dev":"namespace1.0",
  "mode":"sector",
  "size":17162027008,
  "uuid":"029caa76-7be3-4439-8890-9c2e374bcc76",
  "sector_size":4096,
  "blockdev":"pmem1s"
}
```

在示例中，namespace1.0 被重新配置为扇区模式。请注意，块设备名称从 pmem1 改为 pmem1s。这个设备可以与系统中的其它块设备相同。例如：可以对设备进行分区，您可以在设备中创建一个文件系统，可以将该设备配置为软件 RAID 集的一部分，设备可以是 dm-cache 的缓存设备。

28.3. 为文件系统直接访问配置持久内存

文件系统直接访问要求将命名空间配置为 fsdax 模式。此模式可用于直接访问编程模型。当在 fsdax 模式中配置设备时，可以在其之上创建一个文件系统，然后使用 -o fsdax 挂载选项挂载。然后，对这个文件系统的文件执行 mmap() 操作的任何应用程序都可以直接访问其存储。请参见以下示例：

```
# ndctl create-namespace --force --reconfig=namespace0.0 --mode=fsdax --map=mem
{
  "dev":"namespace0.0",
  "mode":"fsdax",
  "size":17177772032,
  "uuid":"e6944638-46aa-4e06-a722-0b3f16a5acbf",
  "blockdev":"pmem0"
}
```

在示例中，namespace0.0 转换为命名空间 fsdax 模式。使用 --map=mem 参数时，ndctl 会将用于直接内存访问(DMA)的操作系统数据结构放在系统 DRAM 中。

要执行 DMA，内核需要内存区域中的每个页面的数据结构。此数据结构的开销为每 4KiB 页的 64 字节。对于小型设备，开销量足够小，无法适应 DRAM。例如，16-GiB 命名空间只需要 256MiB 进行页面结构。因为 NVDIMM 设备通常比较小且昂贵，所以将内核页面跟踪数据结构放在 DRAM 中是首选的，如 --map=mem 参数所示。

未来，NVDIMM 设备的大小可能是 TB。对于这样的设备，存储页面跟踪数据结构所需的内存量可能会超过系统中的 DRAM 数量。一个 TiB 的持久内存只需要 16 GiB 才能进行页面结构。因此，在这种情

况下首选指定 --map=dev 参数将数据结构存储在持久内存本身中。

在 fsdax 模式中配置命名空间后，命名空间就可以用于文件系统。从 Red Hat Enterprise Linux 7.3 开始，Ext4 和 XFS 文件系统都作为技术预览使用持久内存启用。文件系统创建不需要特殊参数。要获得 DAX 功能，请使用 dax 挂载选项挂载文件系统。例如：

```
# mkfs -t xfs /dev/pmem0
# mount -o dax /dev/pmem0 /mnt/pmem/
```

然后，应用程序可以使用持久内存并在 /mnt/pmem/ 目录中创建文件，打开文件，并使用 mmap 操作来映射文件以进行直接访问。

当在用于直接访问的 pmem 设备中创建分区时，分区必须在页面边界上保持一致。在 Intel 64 和 AMD64 构架上，至少有 4KiB 校准分区的开头和结尾，但 2MiB 是首选的对齐。默认情况下，parted 工具在 1MiB 边界上对齐分区。对于第一个分区，指定 2MiB 作为分区的开头。如果分区的大小是 2MiB 的倍数，则所有其他分区也一致。

28.4. 配置持久内存以用于设备 DAX 模式

设备 DAX (devdax)提供了应用程序直接访问存储的方法，而无需参与文件系统。设备 DAX 的好处是它提供有保证的故障粒度，可以使用 --align 选项与 ndctl 工具进行配置：

```
# ndctl create-namespace --force --reconfig=namespace0.0 --mode=devdax --align=2M
```

给定命令可确保操作系统一次在 2MiB 页面中出现错误。对于 Intel 64 和 AMD64 构架，支持以下故障颗粒度：

- *4KiB*
- *2MiB*
- *1GiB*

设备 DAX 节点(/dev/daxN.M)只支持以下系统调用：

- *open()*
- *close()*
- *mmap()*
- *fallocate()*

read() 不支持 write() 变体，因为用例与持久内存编程相关联。

28.5. NVDIMM 故障排除

28.5.1. 使用 S.M.A.R.T 监控 NVDIMM 健康状况。

有些 NVDIMM 支持自监控、分析和报告技术(S.M.A.R.T.)接口来检索健康信息。

定期监控 NVDIMM 健康状况以防止数据丢失。如果 S.M.A.R.T. 报告 NVDIMM 健康状态的问题，请替换它，如 第 28.5.2 节 “检测和替换 Broken NVDIMM” 所述。

先决条件

- *在有些系统中，必须载入 acpi_ipmi 驱动程序才能检索健康信息：*

```
# modprobe acpi_ipmi
```

流程

- *要访问健康信息，请使用以下命令：*

```
# ndctl list --dimms --health
...
  {
    "dev":"nmem0",
    "id":"802c-01-1513-b3009166",
    "handle":1,
    "phys_id":22,
    "health":
```

```
    {
      "health_state":"ok",
      "temperature_celsius":25.000000,
      "spares_percentage":99,
      "alarm_temperature":false,
      "alarm_spares":false,
      "temperature_threshold":50.000000,
      "spares_threshold":20,
      "life_used_percentage":1,
      "shutdown_state":"clean"
    }
  }
...
```

28.5.2. 检测和替换 Broken NVDIMM

如果您在系统日志或者 S.M.A.R.T. 中发现与 NVDIMM 相关的错误消息，这可能意味着 NVDIMM 设备失败。在这种情况下，需要：

1. 检测哪个 NVDIMM 设备失败，
2. 备份存储的数据，以及
3. 物理替换该设备。

过程 28.3. 检测和替换 Broken NVDIMM

1. 要检测有问题的 DIMM，请使用以下命令：

```
# ndctl list --dimms --regions --health --media-errors --human
```

badblocks 字段显示哪些 NVDIMM 有问题。注意它在 dev 字段中的名称。在以下示例中，名为 nmem0 的 NVDIMM 有问题：

例 28.1. NVDIMM 设备的健康状况

```
# ndctl list --dimms --regions --health --media-errors --human

...
  "regions":[
    {
      "dev":"region0",
      "size":"250.00 GiB (268.44 GB)",
```

```
      "available_size":0,
      "type":"pmem",
      "numa_node":0,
      "iset_id":"0xXXXXXXXXXXXXXXXX",
      "mappings":[
       {
        "dimm":"nmem1",
        "offset":"0x10000000",
        "length":"0x1f40000000",
        "position":1
       },
       {
        "dimm":"nmem0",
        "offset":"0x10000000",
        "length":"0x1f40000000",
        "position":0
       }
      ],
      "badblock_count":1,
      "badblocks":[
       {
        "offset":65536,
        "length":1,
        "dimms":[
         "nmem0"
        ]
       }
      ],
      "persistence_domain":"memory_controller"
     }
    ]
}
```

2. 使用以下命令查找有问题的 NVDIMM 的 phys_id 属性：

```
# ndctl list --dimms --human
```

在上例中，您知道 nmem0 是有问题的 NVDIMM。因此，查找 nmem0 的 phys_id 属性。在以下示例中，phys_id 是 0x10:

例 28.2. NVDIMM 的 phys_id 属性

```
# ndctl list --dimms --human

[
  {
    "dev":"nmem1",
    "id":"XXXX-XX-XXXX-XXXXXXXX",
    "handle":"0x120",
    "phys_id":"0x1c"
```

```
  },
  {
    "dev":"nmem0",
    "id":"XXXX-XX-XXXX-XXXXXXXX",
    "handle":"0x20",
    "phys_id":"0x10",
    "flag_failed_flush":true,
    "flag_smart_event":true
  }
]
```

3. 使用以下命令查找有问题的 NVDIMM 的内存插槽：

```
# dmidecode
```

在输出中，找到 Handle 标识符与有问题的 NVDIMM 的 phys_id 属性匹配的条目。Locator 字段列出了有问题的 NVDIMM 使用的内存插槽。在以下示例中，nmem0 设备与 0x0010 标识符匹配，并使用 DIMM-XXX-YYYY 内存插槽：

例 28.3. NVDIMM 内存插槽列表

```
# dmidecode

...
Handle 0x0010, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x0004
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 125 GB
        Form Factor: DIMM
        Set: 1
        Locator: DIMM-XXX-YYYY
        Bank Locator: Bank0
        Type: Other
        Type Detail: Non-Volatile Registered (Buffered)
...
```

4. 备份 NVDIMM 命名空间中的所有数据。如果您在替换 NVDIMM 前没有备份数据，当您从系统中删除 NVDIMM 时数据将会丢失。

警告

在某些情况下，比如 NVDIMM 完全无法正常工作，备份可能会失败。

要防止这种情况，请使用 S.M.A.R.T 定期监控 NVDIMM 设备，如 第 28.5.1 节 “使用 S.M.A.R.T 监控 NVDIMM 健康状况。” 所述，并在它们中断前替换失败的 NVDIMM。

使用以下命令列出 NVDIMM 上的命名空间：

```
# ndctl list --namespaces --dimm=DIMM-ID-number
```

在以下示例中，nmem0 设备包含 namespace0.0 和 namespace0.2 命名空间，您需要备份：

例 28.4. NVDIMM 命名空间列表

```
# ndctl list --namespaces --dimm=0

[
  {
    "dev":"namespace0.2",
    "mode":"sector",
    "size":67042312192,
    "uuid":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "raw_uuid":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "sector_size":4096,
    "blockdev":"pmem0.2s",
    "numa_node":0
  },
  {
    "dev":"namespace0.0",
    "mode":"sector",
    "size":67042312192,
    "uuid":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "raw_uuid":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "sector_size":4096,
    "blockdev":"pmem0s",
    "numa_node":0
  }
]
```

5. *以物理方式替换失效的 NVDIMM。*

第 29 章 NVME OVER FABRIC 设备概述

Non-volatile Memory Express(NVMe)是一个接口，它允许主机软件实用程序与固态驱动器通信。使用以下类型的光纤传输来通过光纤设备配置 NVMe ：

- 使用 Remote Direct Memory Access(RDMA)的 NVMe over fabrics。有关如何配置 NVMe/RDMA 的详情，请参考 第 29.1 节 “使用 RDMA 的 NVMe over fabrics”。

- 使用光纤通道(FC)的 NVMe over fabrics。有关如何配置 FC-NVMe 的详情，请参考 第 29.2 节 “使用 FC 的光纤的 NVMe over fabrics”。

使用 FC 和 RDMA 时，固态驱动器不必是您的系统的本地驱动器；它可以通过 FC 或 RDMA 控制器远程配置。

29.1. 使用 RDMA 的 NVME OVER FABRICS

以下小节介绍了如何通过 RDMA (NVMe/RDMA)发起程序配置部署 NVMe。

29.1.1. 通过 RDMA 客户端配置 NVMe

使用这个流程使用 NVMe 管理命令行界面(nvme-cli)配置 NVMe/RDMA 客户端。

1. 安装 nvme-cli 软件包：

```
# yum install nvme-cli
```

2. 如果没有加载，则加载 nvme-rdma 模块：

```
# modprobe nvme-rdma
```

3. 在 NVMe 目标中发现可用子系统：

```
# nvme discover -t rdma -a 172.31.0.202 -s 4420

Discovery Log Number of Records 1, Generation counter 2
```

```
=====Discovery Log Entry 0======
trtype:  rdma
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified, sq flow control disable supported
portid:  1
trsvcid: 4420
subnqn:  testnqn
traddr:  172.31.0.202
rdma_prtype: not specified
rdma_qptype: connected
rdma_cms:    rdma-cm
rdma_pkey: 0x0000
```

4. **连接到发现的子系统：**

```
# nvme connect -t rdma -n testnqn -a 172.31.0.202 -s 4420

# lsblk

NAME                      MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                         8:0    0 465.8G  0 disk
├─sda1                      8:1    0     1G  0 part /boot
└─sda2                      8:2    0 464.8G  0 part
  ├─rhel_rdma--virt--03-root 253:0    0    50G  0 lvm  /
  ├─rhel_rdma--virt--03-swap 253:1    0     4G  0 lvm  [SWAP]
  └─rhel_rdma--virt--03-home 253:2    0 410.8G  0 lvm  /home
nvme0n1

# cat /sys/class/nvme/nvme0/transport

rdma
```

使用 NVMe 子系统名称替换 testnqn。

将 172.31.0.202 替换为目标 IP 地址。

使用端口号替换 4420。

5. **列出当前连接的 NVMe 设备：**

```
# nvme list
```

6. 可选：断开与目标的连接：

```
# nvme disconnect -n testnqn

NQN:testnqn disconnected 1 controller(s)

# lsblk

NAME                      MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                         8:0    0 465.8G  0 disk
├─sda1                      8:1    0     1G  0 part /boot
└─sda2                      8:2    0 464.8G  0 part
├─rhel_rdma--virt--03-root 253:0    0    50G  0 lvm  /
├─rhel_rdma--virt--03-swap 253:1    0     4G  0 lvm  [SWAP]
└─rhel_rdma--virt--03-home 253:2    0 410.8G  0 lvm  /home
```

其他资源

- 如需更多信息，请参阅 **nvme man page** 和 **NVMe-cli Github 存储库**。

29.2. 使用 FC 的光纤的 NVME OVER FABRICS

与某些 **Broadcom Emulex** 和 **Marvell Qlogic Fibre Channel** 适配器一起使用时，在 **initiator** 模式中完全支持 **NVMe over Fibre Channel (FC-NVMe)**。作为系统管理员，完成以下部分中的任务来部署 **FC-NVMe**：

- **第 29.2.1 节 “为广播适配器配置 NVMe initiator”**
- **第 29.2.2 节 “为 QLogic 适配器配置 NVMe initiator”**

29.2.1. 为广播适配器配置 NVMe initiator

使用这个流程，使用 **NVMe** 管理命令行界面(**nvme-cli**)工具为 **Broadcom** 适配器客户端配置 **NVMe initiator**。

1. 安装 **nvme-cli** 工具：

```
# yum install nvme-cli
```

这会在 /etc/nvme/ 目录中创建 hostnqn 文件。hostn 文件标识 NVMe 主机。生成一个新的 hostnqn ：

```
# nvme gen-hostnqn
```

2. *使用以下内容创建 /etc/modprobe.d/lpfc.conf 文件：*

```
options lpfc lpfc_enable_fc4_type=3
```

3. *重建 initramfs 镜像：*

```
# dracut --force
```

4. *重启主机系统来重新配置 lpfc 驱动程序：*

```
# systemctl reboot
```

5. *找到本地和远程端口的 WWNN 和 WWPN，并使用输出查找子系统 NQN：*

```
# cat /sys/class/scsi_host/host*/nvme_info

NVME Initiator Enabled
XRI Dist lpfc0 Total 6144 IO 5894 ELS 250
NVME LPORT lpfc0 WWPN x10000090fae0b5f5 WWNN x20000090fae0b5f5 DID x010f00
ONLINE
NVME RPORT       WWPN x204700a098cbcac6 WWNN x204600a098cbcac6 DID x01050e
TARGET DISCSRVC ONLINE

NVME Statistics
LS: Xmt 000000000e Cmpl 000000000e Abort 00000000
LS XMIT: Err 00000000  CMPL: xb 00000000 Err 00000000
Total FCP Cmpl 00000000000008ea Issue 00000000000008ec OutIO 0000000000000002
    abort 00000000 noxri 00000000 nondlp 00000000 qdepth 00000000 wqerr 00000000 err
00000000
FCP CMPL: xb 00000000 Err 00000000
```

```
# nvme discover --transport fc \ --traddr nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 \
--host-traddr nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5

Discovery Log Number of Records 2, Generation counter 49530
=====Discovery Log Entry 0======
trtype:  fc
adrfam:  fibre-channel
subtype: nvme subsystem
```

```
treq:    not specified
portid:  0
trsvcid: none
subnqn: nqn.1992-
08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1
traddr:  nn-0x204600a098cbcac6:pn-0x204700a098cbcac6
```

将 nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 替换为 traddr。

将 nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5 替换为 host_traddr。

6. ***使用 nvme-cli 连接到 NVMe 目标：***

```
# nvme connect --transport fc --traddr nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 --
host-traddr nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5 -n nqn.1992-
08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1
```

将 nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 替换为 traddr。

将 nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5 替换为 host_traddr。

将 nqn.1992-08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1 替换为 subnqn。

7. ***验证 NVMe 设备当前是否已连接：***

```
# nvme list
Node             SN                   Model                                    Namespace Usage
Format           FW Rev
---------------- -------------------- ---------------------------------------- ---------
-------------------------- ---------- --------
/dev/nvme0n1     80BgLFM7xMJbAAAAAAAC NetApp ONTAP Controller                   1
107.37  GB / 107.37  GB      4 KiB +  0 B   FFFFFFFF
# lsblk |grep nvme
  nvme0n1                 259:0    0   100G  0 disk
```

其他资源

- 如需更多信息，请参阅 nvme man page 和 NVMe-cli Github 存储库。

29.2.2. 为 QLogic 适配器配置 NVMe initiator

使用这个流程，使用 NVMe 管理命令行界面 (nvme-cli) 工具为 Qlogic 适配器客户端配置 NVMe initiator。

1. 安装 nvme-cli 工具：

```
# yum install nvme-cli
```

这会在 /etc/nvme/ 目录中创建 hostnqn 文件。hostn 文件标识 NVMe 主机。生成一个新的 hostnqn ：

```
# nvme gen-hostnqn
```

2. 删除并重新载入 qla2xxx 模块：

```
# rmmod qla2xxx
# modprobe qla2xxx
```

3. 查找本地和远程端口的 WWNN 和 WWPN：

```
# dmesg |grep traddr
   [    6.139862] qla2xxx [0000:04:00.0]-ffff:0: register_localport: host-traddr=nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 on portID:10700
[    6.241762] qla2xxx [0000:04:00.0]-2102:0: qla_nvme_register_remote: traddr=nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 PortID:01050d
```

使用这个 host-traddr 和 traddr，找到子系统 NQN：

```
# nvme discover --transport fc --traddr nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 --host-traddr nn-0x20000024ff19bb62:pn-0x21000024ff19bb62

Discovery Log Number of Records 2, Generation counter 49530
=====Discovery Log Entry 0======
trtype:  fc
adrfam:  fibre-channel
subtype: nvme subsystem
treq:    not specified
```

```
portid:  0
trsvcid: none
subnqn:  nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_su
bsystem_468
traddr:  nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6
```

将 nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 替换为 traddr。

将 nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 替换为 host_traddr。

4. **使用 nvme-cli 工具连接到 NVMe 目标：**

```
# nvme connect --transport fc --traddr nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 --
host_traddr nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 -n nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_su
bsystem_468
```

将 nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 替换为 traddr。

将 nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 替换为 host_traddr。

将 nqn.1992-08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_subsystem_468 替换为 subnqn。

5. **验证 NVMe 设备当前是否已连接：**

```
# nvme list
Node             SN                   Model                                    Namespace Usage
Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- ------------------------
- ---------------- --------
/dev/nvme0n1     80BgLFM7xMJbAAAAAAAC NetApp ONTAP Controller                   1
107.37  GB / 107.37  GB      4 KiB +  0 B   FFFFFFFF
# lsblk |grep nvme
nvme0n1                 259:0    0   100G  0 disk
```

其他资源

- *如需更多信息，请参阅 nvme man page 和 NVMe-cli Github 存储库。*

部分 III. 使用 VDO 的数据重复数据删除和压缩

本节论述了如何通过启用数据优化器(VDO)来为现有存储管理应用程序提供重复数据删除的块存储功能。

第 30 章 VDO 集成

30.1. VDO 的理论概述

虚拟数据优化器(VDO)是一种块虚拟化技术，可让您轻松创建块存储的压缩和去除重复数据的池。

- 重复数据删除(Deduplication)是通过删除重复块的多个副本来减少存储资源消耗的技术。

  VDO 检测每个重复块，并将其记录为对原始块的引用，而不是多次写相同的数据。VDO 维护一个从逻辑块地址（由 VDO 上面的存储层使用）到物理块地址（被 VDO 下的存储层使用）的映射。

  去除重复数据后，多个逻辑块地址可以映射到相同的物理块地址；称为 共享块。块共享对存储用户是不可见的，用户会像 VDO 不存在一样读写块。当覆盖共享块时，会为存储新块数据分配一个新的物理块，以确保映射到共享物理块的其他逻辑块地址不会被修改。

- 压缩 是一种数据化技术，其工作方式与文件格式，不一定会表现出块级冗余，如日志文件和数据库。详情请查看 第 30.4.8 节 “使用压缩”。

VDO 解决方案包括以下组件：

kvdo

载入 Linux 设备映射器层的内核模块，以提供重复数据删除、压缩和精简置备的块存储卷

uds

与卷上通用的去除重复数据服务(UDS)索引通信的内核模块，并分析数据的重复内容。

命令行工具

用于配置和管理优化的存储。

30.1.1. UDS 内核模块(uds)

UDS 索引提供了 VDO 产品的基础。对于每个新数据段，它会快速确定该数据段是否与之前存储的任

何数据段相同。如果索引发现匹配项，则存储系统可在内部引用现有项，以避免多次保存相同的信息。

UDS 索引作为 uds 内核模块在内核中运行。

30.1.2. VDO 内核模块(kvdo)

kvdo Linux 内核模块在 Linux 设备映射器层中提供块层 deduplication 服务。在 Linux 内核中，设备映射器充当管理块存储池的通用框架，允许在内核块接口和实际存储设备驱动程序之间插入块处理模块的存储堆栈。

kvdo 模块作为块设备公开，可直接访问块存储，或通过其中一个可用的 Linux 文件系统（如 XFS 或 ext4）提供。当 kvdo 收到从 VDO 卷读取数据的请求（逻辑）块时，它会将请求的逻辑块映射到底层物理块，然后读取并返回请求的数据。

当 kvdo 收到向 VDO 卷写入数据块的请求时，它会首先检查它是 DISCARD 还是 TRIM 请求，或者数据是否统一为零。如果其中任何一个条件包含，则 kvdo 会更新其块映射并确认请求。否则，将分配物理块以供请求使用。

VDO 写策略概述

如果 kvdo 模块以同步模式运行：

1. 它会在请求中临时将数据写入分配块中，然后确认请求。
2. 确认完成后，会尝试通过计算块数据的 MurmurHash-3 签名来去除重复的块，该签名发送给 VDO 索引。
3. 如果 VDO 索引包含具有相同签名的块的条目，kvdo 会读取指定的块，并对两个块进行字节比较，以验证它们是否相同。
4. 如果它们实际上是相同的，则 kvdo 会更新其块映射，以便逻辑块指向对应的物理块，并释放分配的物理块。
5. 如果 VDO 索引不包含要写入的块的签名条目，或者指定的块实际上不包含相同的数据，则 kvdo 会更新其块映射，使临时物理块永久。

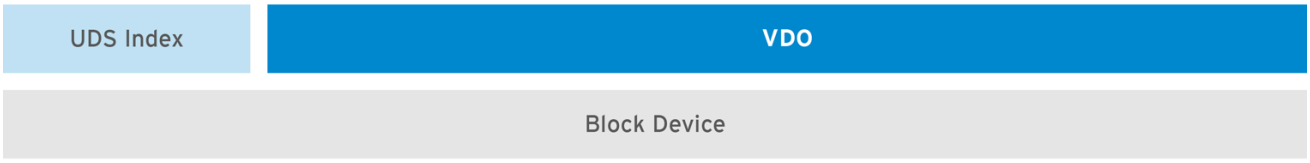
如果 kvdo 以异步模式运行：

1. 它将立即确认请求而不是写数据。
2. 然后它会尝试使用与上述步骤相同的方法来复制块。
3. 如果块变为重复，则 kvdo 将更新其块映射并释放分配的块。否则，它会将请求中的数据写入分配的块，并更新块映射以使物理块永久存在。

30.1.3. VDO 卷

VDO 使用块设备作为后备存储，它包括由一个或多个磁盘、分区甚至平面文件组成的物理存储聚合。当 VDO 卷由存储管理工具创建时，VDO 为 UDS 索引和 VDO 卷保留卷的空间，以便向用户和应用程序提供去除重复数据的块存储。图 30.1 “VDO 磁盘机构” 展示这些部分如何相互兼容。

图 30.1. VDO 磁盘机构



slabs

VDO 卷的物理存储被分成多个 slabs，每个 slab 都是物理空间的连续区域。给定卷的所有 slab 的大小相同，可以是 128 MB 的任何指数，最多为 32 GB。

默认的 slab 大小为 2 GB，用于在较小的测试系统中评估 VDO。单个 VDO 卷最多可能有 8192 个 slabs。因此，在使用 2GB slab 的默认配置中，允许的最大物理存储为 16 TB。当使用 32GB 的 slab 时，允许的最大物理存储为 256 TB。至少一个整个 slab 由 VDO 保留用于元数据，因此不能用于存储用户数据。

slab 大小不影响 VDO 卷的性能。

表 30.1. 根据物理卷大小推荐的 VDO Slab 大小

物理卷大小	推荐的 Slab 大小
10–99 GB	1 GB
100 GB - 1 TB	2 GB
2–256 TB	32 GB

slab 的大小可通过为 --vdoSlabSize=megabytes 命令提供 vdo create 选项来控制。

物理大小和可用物理大小

物理大小和可用物理大小描述了 VDO 可以使用的块设备中的磁盘空间量：

- 物理大小 与底层块设备的大小相同。VDO 使用这个存储用于：
  - 用户数据，这些数据可能会进行重复数据删除和压缩
  - VDO 元数据，如 UDS 索引
- 可用物理大小 是 VDO 可用于用户数据的物理大小的一部分。

  它等同于物理大小减去元数据的大小，再减去将卷根据指定的 slab 的大小分为 slab 后剩余的值。

有关不同大小的块设备中需要多少存储 VDO 元数据的示例，请参阅 第 30.2.3 节 “按物理卷大小划分的 VDO 系统要求示例”。

逻辑大小

如果没有指定 --vdoLogicalSize 选项，则逻辑卷大小默认为可用的物理卷大小。请注意，在 图 30.1 “VDO 磁盘机构” 中，VDO 重复数据删除的存储目标完全位于块设备之上，这意味着 VDO 卷的物理大小与底层块设备的大小相同。

VDO 目前支持任意逻辑卷大小最多为物理卷的 254 倍，但不能超过 4PB。

30.1.4. 命令行工具

VDO 包括以下用于配置和管理的命令行工具：

vdo

创建、配置和控制 VDO 卷

vdostats

提供利用率和性能统计

30.2. 系统要求

处理器架构

需要一个或多个实现 Intel 64 指令集的处理器：即 AMD64 或 Intel 64 架构的处理器。

RAM

每个 VDO 卷有不同的内存要求：

- VDO 模块每 1TB 物理存储管理需要 370 MB 加额外的 268 MB。
- 通用重复数据删除服务(UDS)索引至少需要 250 MB DRAM，这也是去除重复数据使用的默认数量。有关 UDS 内存用量的详情，请参考 第 30.2.1 节 “UDS Index 内存要求”。

存储

VDO 卷是一个精简配置的块设备。要防止物理空间不足，请将卷放在您可以稍后扩展的存储之上。这种可扩展存储示例为 LVM 卷或者 MD RAID 阵列。

可将单个 VDO 卷配置为使用最多 256TB 物理存储。有关从给出 VDO 存储池的物理大小中决定 VDO 管理的卷的可用大小，请参阅 第 30.2.2 节 “VDO 存储空间要求”。

额外的系统软件

VDO 依赖于以下软件：

- *LVM*
- *Python 2.7*

yum 软件包管理器将自动安装所有必要的软件依赖项。

在存储堆栈中放置 VDO

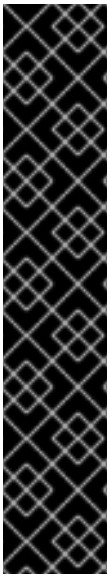
作为常规规则，您应该将某些存储层放在 VDO 下，其他存储层放在 VDO 之上：

- *在 VDO 下：DM-Multipath、DM-Crypt 和软件 RAID (LVM 或 mdraid)。*
- *在 VDO 之上：LVM 缓存、LVM 快照和 LVM Thin Provisioning。*

不支持以下配置：

- *VDO 位于 VDO 卷之上：storage → VDO → LVM → VDO*
- *VDO 位于 LVM 快照之上*
- *VDO 位于 LVM Cache 之上*
- *VDO 位于回送设备之上*
- *VDO 位于 LVM Thin Provisioning 之上*

- 加密的卷（位于 VDO 之上）：storage → VDO → DM-Crypt
- VDO 卷中的分区：fdisk、parted 和类似的分区
- VDO 卷之上的 RAID (LVM、MD 或者任何其他类型)



重要

VDO 支持两种写入模式：sync 和 async。当 VDO 处于 同步 模式时，当底层存储永久写入数据时，会确认对 VDO 设备的写操作。当 VDO 处于 async 模式时，在写入持久性存储前会确认写入。

设置 VDO 写入策略以匹配底层存储的行为非常重要。默认情况下，VDO 写入策略被设置为 auto 选项，该选项会自动选择适当的策略。

如需更多信息，请参阅 第 30.4.2 节 “选择 VDO 写入模式”。

30.2.1. UDS Index 内存要求

UDS 索引由两个部分组成：

- 在内存中使用紧凑表示，每个唯一块最多包含一个条目。
- 记录在索引发生时的相关块名称的磁盘组件，按顺序记录它们。

UDS 在内存中平均使用 4 个字节（包括缓存）。

磁盘上的组件维护传递给 UDS 的数据的相关历史记录。UDS 为属于这个去除重复数据窗口中的数据提供去除重复数据建议，其中包括最近看到的块的名称。去除重复数据窗口允许 UDS 尽可能高效地索引数据，同时限制索引大型数据存储库所需的内存量。虽然去除重复数据窗口的性质，大多数具有高级别的去除重复数据的数据集也表现出高度的时间局部性 - 换句话说，大多数重复数据删除发生在大约同时写入的块集合中。另外，通常要写入的数据通常可能会与最近写入的数据重复。因此，对于给定时间间隔的工作负载，去除重复数据比率通常相同，无论 UDS 仅索引了最新的数据还是所有数据。

由于重复数据往往会表现出时间局部性，因此很少需要对存储系统中的每个块进行索引。否则，索引内存的成本会耗尽导致复制性能降低的存储成本。索引大小要求与数据刷新率紧密相关。例如，假设存储系统的总容量为 100 TB，但每周的摄取率为 1 TB 。UDS 窗口的 deduplication 窗口为 4TB，UDS 可探测到上个月写入的数据的最大冗余度。

UDS 的 Sparse Indexing 功能(VDO 的推荐模式)进一步利用了临时性，方法是尝试在内存中保留最重要的索引条目。UDS 可以维护一个去除重复数据窗口，它在使用相同数量的内存时的十倍。稀疏索引提供了最大的覆盖范围，但密度索引提供了更多建议。对于大多数工作负载，如果内存量相同，则密度和稀疏索引间的重复数据删除率的不同会微不足道。

索引所需的内存由重复数据删除窗口所需的大小决定：

- 对于密度索引，UDS 将每 1 GB RAM 提供 1 TB 的去除重复数据窗口。对于最多 4 TB 的存储系统，1 GB 索引通常就足够了。
- 对于稀疏索引，UDS 将每 1 GB RAM 提供 10 TB 的去除重复数据窗口。1 GB 稀疏索引一般足以满足40TB 物理存储空间。

有关 UDS Index 内存要求的具体示例，请参阅 第 30.2.3 节 “按物理卷大小划分的 VDO 系统要求示例”

30.2.2. VDO 存储空间要求

VDO 需要 VDO 元数据和实际 UDS 重复数据删除索引的存储空间：

- VDO 将两种类型的元数据写入其底层物理存储：
  - 第一个类型使用 VDO 卷的物理大小进行扩展，并为每个 4 GB 物理存储使用大约 1 MB，再加上每个 slab 的额外 1 MB。
  - 第二种类型使用 VDO 卷的逻辑大小进行扩展，并为每个 1 GB 逻辑存储消耗大约 1.25 MB，舍入到最接近的 slab。

  有关 slabs 的描述，请参阅 第 30.1.3 节 “VDO 卷”。

- UDS 索引存储在 VDO 卷组中，并由关联的 VDO 实例管理。所需的存储量取决于索引类型以及分配给索引的 RAM 量。对于每 1 GB RAM，密度 UDS 索引将使用 17 GB 存储，稀疏 UDS 索引将使用 170 GB 存储。

有关 VDO 存储要求的具体示例，请参阅 第 30.2.3 节 “按物理卷大小划分的 VDO 系统要求示例”

30.2.3. 按物理卷大小划分的 VDO 系统要求示例

下表根据基础物理卷的大小提供 VDO 大约系统要求。每个表列出了适合预期部署的需求，如主存储或备份存储。

具体数量取决于您的 VDO 卷的配置。

主存储部署

在主存储情形中，UDS 索引是物理卷大小的 0.01% 到 25%。

表 30.2. 主存储的 VDO 存储和内存要求

物理卷大小	10 GB - 1-TB	2-10 TB	11-50 TB	51–100 TB	101–256 TB
RAM 使用量	250 MB	密度：1 GB 稀疏： 250 MB	2 GB	3 GB	12 GB
磁盘用量	2.5 GB	密度： 10 GB 稀疏：22 GB	170 GB	255 GB	1020 GB
索引类型	密度	dense 或 Sparse	稀疏	稀疏	稀疏

备份存储部署

在备份存储中，UDS 索引覆盖了备份集的大小，但不大于物理卷。如果您预期备份集或物理大小在以后会增大，则需要把这个值加到索引大小中。

表 30.3. 备份存储的 VDO 存储和内存要求

物理卷大小	10 GB - 1 TB	2-10 TB	11-50 TB	51–100 TB	101–256 TB
RAM 使用量	250 MB	2 GB	10 GB	20 GB	26 GB
磁盘用量	2.5 GB	170 GB	850 GB	1700 GB	3400 GB
索引类型	密度	稀疏	稀疏	稀疏	稀疏

30.3. VDO 入门

30.3.1. 简介

Virtual Data Optimizer(VDO)以重复数据删除（deduplication）、压缩和精简置备的形式为 Linux 提供内联数据降低。当您设置 VDO 卷时，您可以指定一个块设备来构建 VDO 卷以及您要存在的逻辑存储量。

- 当托管活跃的虚拟机或容器时，红帽建议使用 10:1 逻辑与物理比例置备存储：也就是说，如果您使用 1TB 物理存储，您将把它显示为 10TB 逻辑存储。
- 对于对象存储，如 Ceph 提供的类型，红帽建议使用 3:1 逻辑与物理比例：1TB 的物理存储将显示为 3TB 逻辑存储。

在这两种情况下，您只需将文件系统放在 VDO 提供的逻辑设备之上，然后直接使用它，或将其作为分布式云存储架构的一部分。

本章论述了 VDO 部署的以下用例：

- 虚拟化服务器的直接连接用例，比如使用 Red Hat Virtualization 构建的用户，以及
- 基于对象的分布式存储集群的云存储用例，如使用 Ceph Storage 构建的云存储用例。

注意

目前不支持使用 Ceph 进行 VDO 部署。

本章提供了配置 VDO 以与标准 Linux 文件系统搭配使用的示例，该系统可针对任一用例轻松部署；请参阅 第 30.3.5 节 “部署示例” 中的图。

30.3.2. 安装 VDO

VDO 使用以下 RPM 软件包部署：

- vdo
- kmod-kvdo

要安装 VDO，请使用 yum 软件包管理器安装 RPM 软件包：

```
# yum install vdo kmod-kvdo
```

30.3.3. 创建 VDO 卷

为您的块设备创建 VDO 卷。请注意，可以为同一机器上的独立设备创建多个 VDO 卷。如果选择这个方法，必须为系统上的每个 VDO 实例提供不同的名称和设备。

重要

使用可扩展存储作为后备块设备。如需更多信息，请参阅 第 30.2 节 “系统要求”。

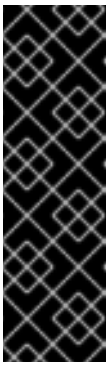
在以下步骤中，将 vdo_name 替换为您要用于 VDO 卷的标识符，例如 vdo1。

1. 使用 VDO Manager 创建 VDO 卷：

```
# vdo create \
      --name=vdo_name \
      --device=block_device \
      --vdoLogicalSize=logical_size \
      [--vdoSlabSize=slab_size]
```

- 使用您要创建 VDO 卷的块设备的持久性名称替换 block_device。例如：

/dev/disk/by-id/scsi-3600508b1001c264ad2af21e903ad031f。



重要

使用持久的设备名称。如果您使用非持久性设备名称，则如果设备名称改变了，VDO 将来可能无法正常启动。

有关持久性名称的更多信息，请参阅 第 25.8 节 “持久性命名”。

- 将 logical_size 替换为 VDO 卷应该存在的逻辑存储量：
  - 对于活跃的虚拟机或容器存储，逻辑大小为您的块设备物理大小的 十 倍。例如，如果您的块设备大小为 1 TB，请在此处使用 10T。
  - 对于对象存储，使用逻辑大小，即您的块设备物理大小的 三倍。例如，如果您的块设备大小为 1 TB，请在此处使用 3T。
- 如果块设备大于 16 TiB，请添加 --vdoSlabSize=32G，以将卷的 slab 大小增加到 32 GiB。

在大于 16 TiB 的块设备上使用 2 GiB 的默认 slab 大小会导致 vdo create 命令失败，并显示以下错误：

```
vdo: ERROR - vdoformat: formatVDO failed on '/dev/device': VDO Status: Exceeds
maximum number of slabs supported
```

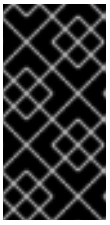
如需更多信息，请参阅 第 30.1.3 节 “VDO 卷”。

例 30.1. 为容器存储创建 VDO

例如，要为 1TB 块设备上的容器存储创建 VDO 卷，您可以使用：

```
# vdo create \
      --name=vdo1 \
      --device=/dev/disk/by-id/scsi-3600508b1001c264ad2af21e903ad031f \
      --vdoLogicalSize=10T
```

创建 VDO 卷时，VDO 会在 /etc/vdoconf.yml 配置文件中添加一个条目。然后，vdo.service systemd 单元使用该条目来默认启动卷。



重要

如果在创建 VDO 卷时发生故障，请删除要清理的卷。详情请查看 第 30.4.3.1 节 “删除 Unsuccessfully Created 卷”。

2. 创建文件系统：

- 对于 XFS 文件系统：

```
# mkfs.xfs -K /dev/mapper/vdo_name
```

- 对于 ext4 文件系统：

```
# mkfs.ext4 -E nodiscard /dev/mapper/vdo_name
```

3. 挂载文件系统：

```
# mkdir -m 1777 /mnt/vdo_name
# mount /dev/mapper/vdo_name /mnt/vdo_name
```

4. 要将文件系统配置为自动挂载，请使用 /etc/fstab 文件或 systemd 挂载单元：

- 如果您决定使用 /etc/fstab 配置文件，请在文件中添加以下行：
  - 对于 XFS 文件系统：

```
/dev/mapper/vdo_name /mnt/vdo_name xfs defaults,_netdev,x-systemd.device-timeout=0,x-systemd.requires=vdo.service 0 0
```

- 对于 **ext4** 文件系统：

```
/dev/mapper/vdo_name /mnt/vdo_name ext4 defaults,_netdev,x-systemd.device-timeout=0,x-systemd.requires=vdo.service 0 0
```

- 或者，如果您决定使用 **systemd** 单元，请使用适当的文件名创建一个 **systemd** 挂载单元文件。对于 **VDO** 卷的挂载点，使用以下内容创建 **/etc/systemd/system/mnt-vdo_name.mount** 文件：

```
[Unit]
Description = VDO unit file to mount file system
name = vdo_name.mount
Requires = vdo.service
After = multi-user.target
Conflicts = umount.target

[Mount]
What = /dev/mapper/vdo_name
Where = /mnt/vdo_name
Type = xfs

[Install]
WantedBy = multi-user.target
```

systemd 单元文件示例也安装在 **/usr/share/doc/vdo/examples/systemd/VDO.mount.example** 中。

5. 为 **VDO** 设备中的文件系统启用 **discard** 功能。批处理和在线操作都可用于 **VDO**。

有关如何设置 **discard** 功能的详情，请参考 **第 2.4 节 “丢弃未使用的块”**。

30.3.4. 监控 VDO

因为 **VDO** 是精简置备的，所以文件系统和应用程序只会看到使用的逻辑空间，并不知道可用的实际物理空间。

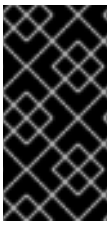
VDO 空间使用量和效率可使用 **vdostats** 工具监控：

```
# vdostats --human-readable

Device                 1K-blocks    Used     Available    Use%    Space saving%
```

```
/dev/mapper/node1osd1   926.5G   21.0G   905.5G   2%   73%
/dev/mapper/node1osd2   926.5G   28.2G   898.3G   3%   64%
```

当 VDO 卷的物理存储容量接近满时，VDO 在系统日志中报告警告，如下所示：

```
Oct  2 17:13:39 system lvm[13863]: Monitoring VDO pool vdo_name.
Oct  2 17:27:39 system lvm[13863]: WARNING: VDO pool vdo_name is now 80.69% full.
Oct  2 17:28:19 system lvm[13863]: WARNING: VDO pool vdo_name is now 85.25% full.
Oct  2 17:29:39 system lvm[13863]: WARNING: VDO pool vdo_name is now 90.64% full.
Oct  2 17:30:29 system lvm[13863]: WARNING: VDO pool vdo_name is now 96.07% full.
```



重要

监控 VDO 卷的物理空间，以防止出现空间不足的情况。物理块不足可能会导致 VDO 卷中最近写入的数据丢失。

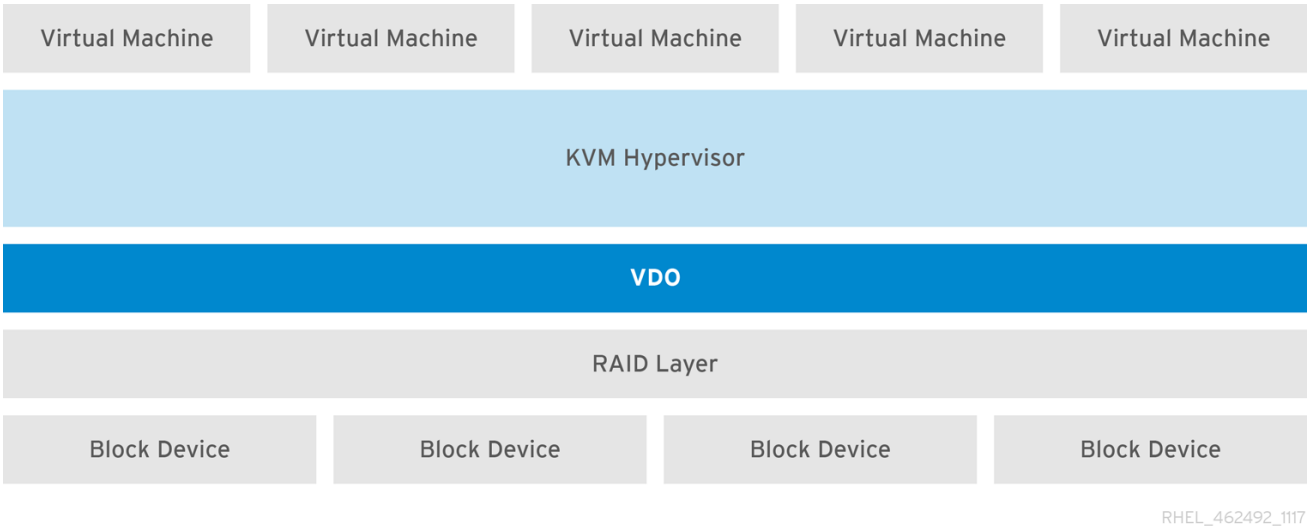
30.3.5. 部署示例

以下示例演示了如何在 KVM 和其他部署中使用 VDO。

使用 KVM 部署 VDO

要了解如何在配置了直接附加存储的 KVM 服务器中成功部署 VDO，请参阅 图 30.2 “使用 KVM 部署 VDO”。

图 30.2. 使用 KVM 部署 VDO



更多部署场景

有关 VDO 部署的更多信息，请参阅 第 30.5 节 “部署场景”。

30.4. 管理 VDO

30.4.1. 启动或停止 VDO

要启动给定的 VDO 卷或所有 VDO 卷以及关联的 UDS 索引，存储管理工具应调用其中一个命令：

```
# vdo start --name=my_vdo
# vdo start --all
```

安装 vdo 软件包时，默认安装并启用 VDO systemd 单元。这个单元会在系统启动时自动运行 vdo start --all 命令，以启动所有 激活的 VDO 卷。请参阅 第 30.4.6 节 “在系统引导时自动启动 VDO 卷” 了解更多信息。

要停止给定的 VDO 卷或所有 VDO 卷，以及相关的 UDS 索引，请使用以下命令之一：

```
# vdo stop --name=my_vdo
# vdo stop --all
```

停止 VDO 卷需要根据存储设备速度以及卷需要写入的数据量而有所不同：

- 卷总是为每 1GiB UDS 索引写入大约 1GiB。
- 使用稀疏 UDS 索引，卷还会写入与块映射缓存大小相等的数据量，再加上每个 slab 的 8MiB。

如果在未完全关闭后重启，VDO 将执行重建以验证其元数据的一致性，并在需要时进行修复。重建是自动的，不需要用户干预。有关重建过程的更多信息，请参阅 第 30.4.5 节 “在 Unclean Shutdown 后恢复 VDO 卷”。

VDO 可能会重建不同的写入模式，具体取决于写入模式：

- 在同步模式下，在关闭前被 VDO 确认的所有写入都会被重建。

- 在异步模式中，在最后一次确认刷新请求之前确认的所有写入都会被重新构建。

在任一模式中，一些未确认或未刷新的写入也可能被重建。

有关 VDO 写入模式的详情，请参考 第 30.4.2 节 “选择 VDO 写入模式”。

30.4.2. 选择 VDO 写入模式

VDO 支持三种写入模式、同步、 sync 和 auto ：

- 当 VDO 处于 同步 模式时，它假定写入命令将数据写入持久性存储。因此，文件系统或应用程序不需要发出 FLUSH 或 Force unit Access (FUA)请求，从而导致数据在关键点变为持久。

  只有当底层存储保证数据在 write 命令完成后写入持久性存储时，才必须将 VDO 设置为 sync 模式。也就是说，存储必须没有易变的写缓存，或者不通过缓存进行写入操作。

- 当 VDO 处于 async 模式时，无法保证在确认写命令时写入持久性存储。文件系统或应用程序必须发出 FLUSH 或 FUA 请求，来确保每次事务中数据在关键点上的持久性。

  如果底层存储无法保证在写命令完成后写入持久性存储，则必须将 VDO 设置为 async 模式；也就是说，当存储具有易失性写回缓存时。

  有关如何查找设备是否使用易失性缓存或未易失性的详情，请参考 “检查 Volatile Cache”一节。

警告

当 VDO 以 async 模式运行时，它与 Atomicity, Consistency, Isolation, Durability (ACID)不兼容。当 VDO 卷之上假设 ACID 合规性的应用程序或文件系统时，sync mode 可能会导致意外的数据丢失。

- auto 模式根据每个设备的特性自动选择 sync 或 async。这是默认选项。

有关写入策略如何操作的详情，请参考 “VDO 写策略概述”一节。

要设置写入策略，请使用 --writePolicy 选项。这可以在创建 VDO 卷时指定为 第 30.3.3 节 “创建 VDO 卷”，或使用 changeWritePolicy 子命令修改现有 VDO 卷时：

```
# vdo changeWritePolicy --writePolicy=sync|async|auto --name=vdo_name
```

重要

使用不正确的写入策略可能会导致数据丢失。

检查 Volatile Cache

要查看设备是否具有回写缓存，请阅读 /sys/block/block_device/device/scsi_disk/标识符/cache_type sysfs 文件。例如：

- 设备 sda 表示 它有一个 回写缓存：

```
$ cat '/sys/block/sda/device/scsi_disk/7:0:0:0/cache_type'
write back
```

- 设备 sdb 表示 它没有 回写缓存：

```
$ cat '/sys/block/sdb/device/scsi_disk/1:2:0:0/cache_type'
None
```

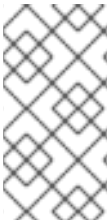
另外，在内核引导日志中，您可以找到上述设备是否有写缓存：

```
sd 7:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
sd 1:2:0:0: [sdb] Write cache: disabled, read cache: disabled, supports DPO and FUA
```

有关读取 系统日志的更多信息，请参阅系统管理员指南中的 查看和管理 日志文件章节。

在这些示例中，为 VDO 使用以下写入策略：

- sda 设备的 async 模式
- sdb 设备的 同步 模式



注意

如果 cache_type 值是 none 或通过 写入，则您应该将 VDO 配置为使用 同步 写入策略。

30.4.3. 删除 VDO 卷

可运行以下命令来从系统中删除 VDO 卷：

```
# vdo remove --name=my_vdo
```

在删除 VDO 卷前，卸载文件系统并停止使用存储的应用程序。vdo remove 命令删除 VDO 卷及其关联的 UDS 索引，以及它们所在的逻辑卷。

30.4.3.1. 删除 Unsuccessfully Created 卷

如果在 vdo 工具创建 VDO 卷时失败，则该卷将处于中间状态。例如，当系统崩溃、电源失败或者管理员中断了运行的 vdo create 命令时会出现这种情况。

要从这种情况中清理，请使用 --force 选项删除创建失败的卷：

```
# vdo remove --force --name=my_vdo
```

需要 --force 选项，因为因为卷创建失败，管理员可能会由于更改系统配置而导致冲突。如果没有 --force 选项，vdo remove 命令会失败，并显示以下信息：

```
[...]
A previous operation failed.
Recovery from the failure either failed or was interrupted.
Add '--force' to 'remove' to perform the following cleanup.
Steps to clean up VDO my_vdo:
umount -f /dev/mapper/my_vdo
udevadm settle
dmsetup remove my_vdo
vdo: ERROR - VDO volume my_vdo previous operation (create) is incomplete
```

30.4.4. 配置 UDS Index

VDO 使用名为 UDS 的高性能 deduplication 索引来检测正在存储的数据重复块。去除重复数据窗口是索引记住之前写入的块的数量。可配置去除重复数据窗口窗口的大小。对于给定的窗口大小，索引需要特定数量的 RAM 以及特定数量的磁盘空间。窗口的大小通常通过使用 **--indexMem=size** 选项指定索引内存的大小来确定。然后会自动决定要使用的磁盘空间量。

通常，红帽建议在所有生产用例中使用 稀疏 UDS 索引。这是一个非常高效的索引数据结构，在其去除重复数据窗口中每个块大约需要一字节 DRAM。在磁盘上，每个块大约需要 72 字节磁盘空间。这个索引的最小配置使用 256 MB 的 DRAM 和大约 25 GB 空间。要使用此配置，请在 **vdo create** 命令中指定 **--sparseIndex=enabled --indexMem=0.25** 选项。这个配置会导致一个 deduplication 窗口 2.5 TB（也就是说它会记住 2.5 TB 的历史记录）。在大多数用例中，2.5 TB 的去除重复数据窗口适合用于大小高达 10 TB 的去除重复数据存储池。

但是，索引的默认配置是使用 密度 索引。DRAM 中的这个索引效率要低得多(10 倍)，但它的最少需要磁盘空间（也是 10 倍）最少需要磁盘空间，使其更便于在受限环境中进行评估。

通常，建议配置一个 VDO 卷的物理大小的 deduplication 窗口。但这不是实际要求。即便是小的去除重复数据窗口（与物理存储量相比）也可以在许多用例中发现大量重复的数据。也可以使用较大的窗口，但多数情况下，这样做将无用。

请联系您的红帽大客户经理代表以获得有关调整此重要系统参数的其他指南。

30.4.5. 在 Unclean Shutdown 后恢复 VDO 卷

如果在不完全关闭的情况下重启卷，VDO 将需要重建一部分元数据才能继续操作，这在卷启动时会自动发生。（另请参阅 第 30.4.5.2 节 “强制重新构建” 在完全关闭的卷中调用此过程。）

数据恢复取决于设备的写入策略：

-

如果 VDO 在同步存储上运行，并且写入策略被设置为 同步，那么写入卷的所有数据都将完全恢复。

- 如果写入策略是 async 的，则在通过发送 VDO 命令或带有 FLUSH 标志（强制单元访问）标记的写入 I/O 时可能无法恢复一些写入。FUA这可以通过调用 fsync、fdatasync、sync 或 umount 等数据完整性操作来实现。

30.4.5.1. 在线恢复

在大多数情况下，重建未清理的 VDO 卷的大多数工作可以在 VDO 卷恢复在线后执行，同时为读取和写入请求提供服务。最初，写入请求的可用空间量可能会限制。当恢复卷的元数据的更多信息，更多可用空间可能会变得可用。另外，如果数据位于尚未恢复的卷的一部分，则在 VDO 恢复时写入的数据可能无法对崩溃前写入的数据进行重复数据删除。在卷被恢复时，可以压缩数据。之前压缩的块可能仍然被读或覆盖。

在线恢复期间，将无法使用很多统计数据：例如，使用中的 块和 块 free。重建完成后，这些统计数据将可用。

30.4.5.2. 强制重新构建

VDO 可以从大多数硬件和软件错误中恢复。如果 VDO 卷无法成功恢复，则会将其置于在卷重启后保留的只读模式。当卷处于只读模式后，就无法保证数据没有丢失或损坏。在这种情况下，红帽建议从只读卷中复制数据，并可能从备份中恢复卷。（ vdostats 的操作模式 属性指示 VDO 卷是否处于只读模式。）

如果数据崩溃的风险可以接受，则可以强制离线重新构建 VDO 卷元数据，以便将该卷重新在线并可用。同样，无法保证重建数据的完整性。

要强制重建只读 VDO 卷，如果卷正在运行，首先停止该卷：

```
# vdo stop --name=my_vdo
```

然后，使用 --forceRebuild 选项重启卷：

```
# vdo start --name=my_vdo --forceRebuild
```

30.4.6. 在系统引导时自动启动 VDO 卷

在系统引导过程中，vdo systemd 单元会自动启动所有配置为 激活 的 VDO 设备。

要防止某些现有卷自动启动，请运行以下命令来 停用 这些卷：

- 取消激活一个特定卷：

```
# vdo deactivate --name=my_vdo
```

- 取消激活所有卷：

```
# vdo deactivate --all
```

相反，要激活卷，请使用以下命令之一：

- 激活一个特定卷：

```
# vdo activate --name=my_vdo
```

- 激活所有卷：

```
# vdo activate --all
```

您还可以通过在 vdo create 命令中添加 --activate=disabled 选项来创建不自动启动的 VDO 卷。

对于将 LVM 卷放在 VDO 卷之上以及它们下的系统（例如 图 30.5 “去除重复数据的统一存储”），以正确顺序启动服务非常重要：

1. 必须首先启动 LVM 的下层（在大多数系统中，当安装 LVM2 软件包时，会自动配置这个层）。

2. 然后，必须启动 vdo systemd 单元。

3. 最后，必须运行其他脚本，以便在现在运行 VDO 卷之上启动 LVM 卷或其他服务。

30.4.7. 禁用和重新启用重复数据删除

在某些情况下，可能需要临时禁用写入 VDO 卷的去除重复数据，同时仍保留从卷读取和写入的功能。禁用 deduplication 可防止后续写入操作被重复数据删除，但已经删除重复数据的数据仍会保留。

- 要在 VDO 卷上停止 deduplication，请使用以下命令：

```
# vdo disableDeduplication --name=my_vdo
```

这会停止关联的 UDS 索引，并通知 VDO 卷 deduplication 不再活跃。

- 要在 VDO 卷上重启 deduplication，请使用以下命令：

```
# vdo enableDeduplication --name=my_vdo
```

这会重启关联的 UDS 索引，并通知 VDO 卷再次激活 deduplication。

您还可以通过在 vdo create 命令中添加 --deduplication=disabled 选项，在创建新 VDO 卷时禁用 deduplication。

30.4.8. 使用压缩

30.4.8.1. 简介

除了块级 deduplication 外，VDO 还使用 HIOPS Compression™ 技术提供内联块级压缩。虽然 deduplication 对虚拟机环境和备份应用程序是最佳解决方案，但压缩非常适合结构化和非结构化的文件格式，这些文件格式通常不会展示块级冗余，如日志文件和数据库。

压缩对未识别为重复的块进行操作。第一次看到唯一数据时，它将被压缩。已存储的数据的后续副本会复制，而无需额外的压缩步骤。压缩功能基于一种基于并行的打包算法，其允许一次处理许多压缩操作。在首先存储块并响应请求者后，最佳打包算法会找到多个块，当压缩时，这些块可以放入一个物理块

中。确定特定的物理块不太可能保存其他压缩块后，它将被写入存储，并且未压缩块的被释放并被重复使用。在已经响应请求者后执行压缩和打包操作，使用压缩会带来最小的延迟损失。

30.4.8.2. 启用和禁用压缩

VDO 卷压缩默认是 on。

在创建卷时，您可以通过在 vdo create 命令中添加 --compression=disabled 选项来禁用压缩。

如果需要，可以在现有 VDO 卷上停止压缩，也可以加快对性能的处理速度，不太可能压缩。

- 要停止 VDO 卷上的压缩，请使用以下命令：

```
# vdo disableCompression --name=my_vdo
```

- 要再次启动它，请使用以下命令：

```
# vdo enableCompression --name=my_vdo
```

30.4.9. 管理可用空间

因为 VDO 是一个精简配置的块存储目标，所以 VDO 使用的物理空间量可能与提供给存储用户的卷大小不同。整合商和系统管理员可以利用这种差异化，以节省存储成本，但如果写入的数据无法达到预期去除重复数据率，则必须小心地避免意外耗尽存储空间。

每当逻辑块（虚拟存储）的数量超过物理块（实际存储）的数量时，文件系统和应用程序可能会意外地遇到没有存储空间的问题。因此，使用 VDO 的存储系统必须为存储管理员提供监控 VDO 空闲池大小的方法。这个可用池的大小可以通过使用 **vdostats** 工具来决定；详情请查看 第 30.7.2 节 “vdostats”。此工具的默认输出列出所有运行 VDO 卷的信息，其格式与 Linux df 实用程序类似。例如：

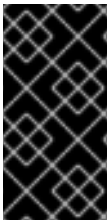
```
Device            1K-blocks   Used       Available   Use%
/dev/mapper/my_vdo  211812352   105906176   105906176     50%
```

当 VDO 卷的物理存储容量接近满时，VDO 在系统日志中报告警告，如下所示：

```
Oct  2 17:13:39 system lvm[13863]: Monitoring VDO pool my_vdo.
Oct  2 17:27:39 system lvm[13863]: WARNING: VDO pool my_vdo is now 80.69% full.
```

```
Oct  2 17:28:19 system lvm[13863]: WARNING: VDO pool my_vdo is now 85.25% full.
Oct  2 17:29:39 system lvm[13863]: WARNING: VDO pool my_vdo is now 90.64% full.
Oct  2 17:30:29 system lvm[13863]: WARNING: VDO pool my_vdo is now 96.07% full.
```

如果 VDO 的可用池大小低于某个级别，则存储管理员可以通过删除数据来采取行动（当删除的数据没有重复时回收空间）、添加物理存储甚至删除 LUN。



重要

监控 VDO 卷的物理空间，以防止出现空间不足的情况。物理块不足可能会导致 VDO 卷中最近写入的数据丢失。

在文件系统上重新声明空间

除非文件系统使用 DISCARD、TRIM 或 UNMAP 命令告知块是空闲的，否则 VDO 无法回收空间。对于不使用 DISCARD、TRIM 或 UNMAP 的文件系统，可以通过存储由二进制零组成的文件手动回收空闲空间，然后删除该文件。

文件系统通常配置为以以下两种方式之一发出 DISCARD 请求：

实时丢弃（也在线丢弃或内联丢弃）

启用实时丢弃时，当用户删除文件和释放空间时，文件系统会将 REQ_DISCARD 请求发送到块层。VDO 会检索这些请求，并将空间返回到其空闲池，假设块没有共享。

对于支持在线丢弃的文件系统，您可以在挂载时设置 discard 选项来启用它。

批量丢弃

批量丢弃是一种用户发起的操作，可导致文件系统通知块层(VDO)任何未使用的块。这可以通过向文件系统发送名为 FITRIM 的 ioctl 请求来完成。

您可以使用 fstrim 工具（例如从 cron）将此 ioctl 发送到文件系统。

有关丢弃功能的详情，请参考 第 2.4 节 “丢弃未使用的块”。

在没有文件系统的情况下重新声明空间

当存储被用作没有文件系统的块存储目标时，也可以管理可用空间。例如，可以通过在其上安装逻辑卷管理器(LVM)，将单个 VDO 卷划分为多个子卷。在取消置备卷前，可以使用 blkdiscard 命令来释放该逻辑卷之前使用的空间。LVM 支持 REQ_DISCARD 命令，并在适当的逻辑块地址上将请求转发到 VDO，以释放空间。如果使用其他卷管理器，它们还需要支持 REQ_DISCARD，或者等效地支持 SCSI 设备的 UNMAP 或用于 ATA 设备的 TRIM。

在光纤通道或以太网网络中回收空间

VDO 卷（或卷部分）也可以置备到光纤通道存储结构上的主机，或使用 SCSI 目标框架（如 LIO 或 SCST）的主机。SCSI 启动器可以使用 UNMAP 命令在精简配置的存储目标上释放空间，但需要配置 SCSI 目标框架来公告对这个命令的支持。这通常是通过在这些卷上启用 精简配置 来完成的。运行以下命令，可以在基于 Linux 的 SCSI 启动器上验证对 UNMAP 的支持：

```
# sg_vpd --page=0xb0 /dev/device
```

在输出中，验证 "Maximum unmap LBA count" 值是否大于零。

30.4.10. 增加逻辑卷大小

管理应用程序可以使用 vdo growLogical 子命令增加 VDO 卷的逻辑大小。卷增大后，管理应告知 VDO 卷的新大小之上的任何设备或文件系统。卷可能会增大，如下所示：

```
# vdo growLogical --name=my_vdo --vdoLogicalSize=new_logical_size
```

使用此命令可让存储管理员初始创建 VDO 卷，其逻辑大小足够小，以便安全地耗尽空间。一段时间后，可以评估实际数据缩减率，如果足够，可以增大 VDO 卷的逻辑大小以利用空间节省。

30.4.11. 增加物理卷大小

增加 VDO 卷可用的物理存储量：

1. 增加底层设备的大小。

   确切的流程取决于设备的类型。例如：要调整 MBR 分区的大小，请使用 fdisk 工具，如 第 13.5 节 “使用 fdisk 重新定义分区大小” 所述。

2. 使用 growPhysical 选项将新的物理存储空间添加到 VDO 卷中：

```
# vdo growPhysical --name=my_vdo
```

无法使用这个命令缩小 VDO 卷。

30.4.12. 使用 Ansible 自动化 VDO

您可以使用 Ansible 工具自动部署和管理 VDO。详情请查看：

- Ansible 文档：https://docs.ansible.com/
- VDO Ansible 模块文档：https://docs.ansible.com/ansible/latest/modules/vdo_module.html

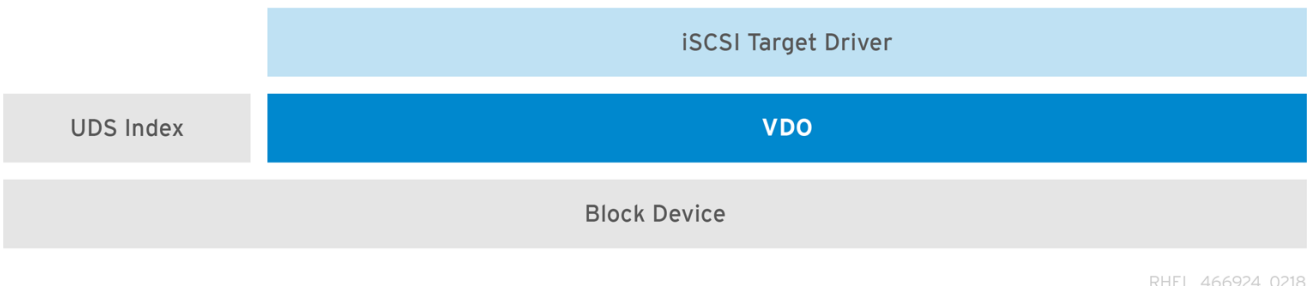
30.5. 部署场景

VDO 可以以不同的方式部署，为块和文件访问以及本地和远程存储提供重复数据删除的存储。因为 VDO 会将重复数据删除存储公开为标准 Linux 块设备，所以它可以用于标准文件系统、iSCSI 和 FC 目标驱动程序或统一存储。

30.5.1. iSCSI 目标

例如，整个 VDO 存储目标都可以导出为 iSCSI 目标到远程 iSCSI 启动器。

图 30.3. 去除重复数据的块存储目标

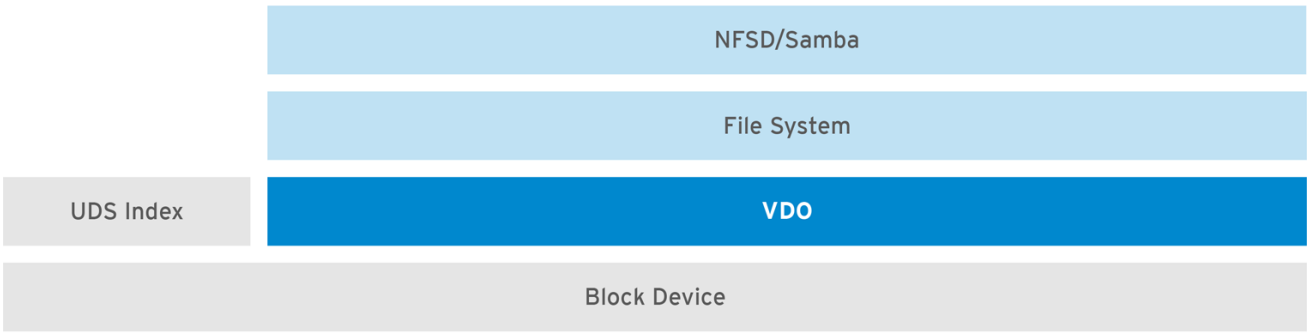


有关 http://linux-iscsi.org/ iSCSI 目标的更多信息，请参阅。

30.5.2. 文件系统

如果需要文件访问权限，可以在 VDO 上创建文件系统，并通过 Linux NFS 服务器或 Samba 向 NFS 或 CIFS 用户公开。

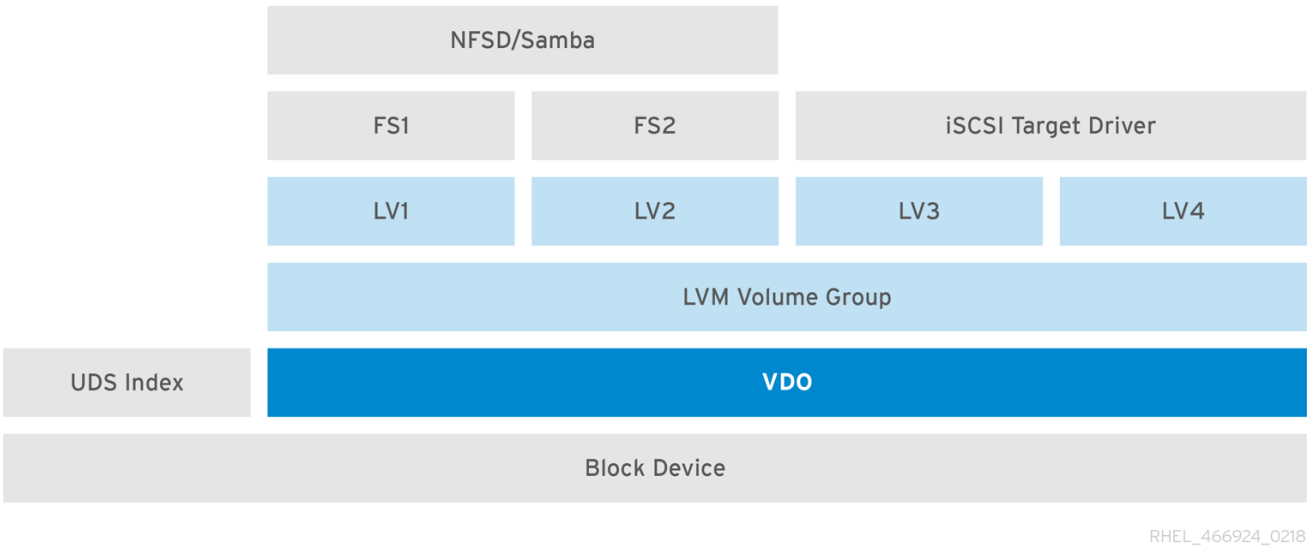
图 30.4. Deduplicated NAS



30.5.3. LVM

功能丰富的系统可能会进一步使用 LVM 来提供多个 LUN，这些 LUN 都由同一去除重复数据的存储池支持。在 图 30.5 “去除重复数据的统一存储” 中，VDO 目标被注册为一个物理卷，以便它可以由 LVM 管理。从删除复制数据的存储池中创建多个逻辑卷（LV1到 LV4）。这样，VDO 可以支持多协议统一块/文件访问底层重复数据删除的存储池。

图 30.5. 去除重复数据的统一存储



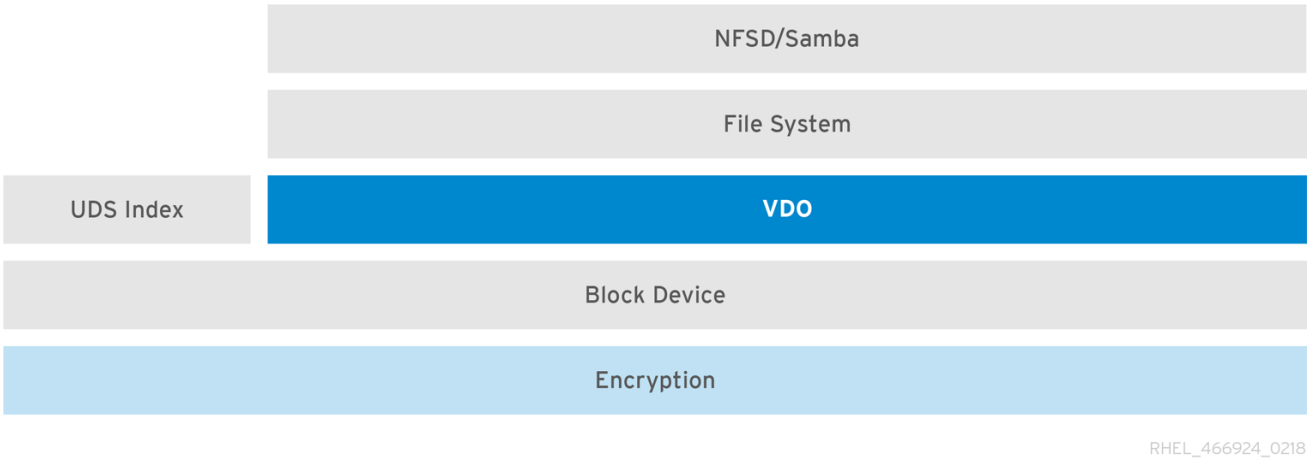
去除重复数据的统一存储设计允许多个文件系统通过 LVM 工具集中使用相同的 deduplication 域。另

外，文件系统可以利用 LVM 快照、写时复制、缩小或增长的功能，所有这些都位于 VDO 之上。

30.5.4. 加密

数据安全性目前至关重要。更多公司对数据加密有内部策略。DM-Crypt 等 Linux 设备映射器机制与 VDO 兼容。加密 VDO 卷有助于确保数据安全性，VDO 以上的任何文件系统仍然会获得用于磁盘优化的去除重复数据功能。请注意，如果数据去除重复数据，则应用上面的 VDO 加密会稍微小；加密会在 VDO 可以去除重复块前显示不同的重复块。

图 30.6. 使用带有加密的 VDO



30.6. 调整 VDO

30.6.1. VDO 调优简介

与调优数据库或其他复杂软件一样，调优 VDO 涉及在多个系统限制之间实现利弊，需要进行一些试验。可用于调优 VDO 的主要控制是分配给不同类型的工作的线程数量、这些线程的 CPU 关联性设置以及缓存设置。

30.6.2. VDO 架构背景信息

VDO 内核驱动程序是多线程的，通过在多个并发 I/O 请求间增加处理成本来提高性能。它将一个线程处理从头到尾的 I/O 请求，而是将不同的工作阶段委派给一个或多个线程或线程组，它们之间传递的消息通过管道进行。这样，一个线程可以序列化对全局数据结构的所有访问，而无需锁定并在每次处理 I/O 操作时解锁它。如果 VDO 驱动程序是精心调优的，则每次线程完成请求处理阶段时，通常会进行另一个排队以进行相同的处理。保持这些线程忙碌可减少上下文切换和调度开销，从而提高性能。单独的线程也用于可阻止的操作系统的一部分，如将 I/O 操作排队到底层存储系统或消息到 UDS。

VDO 使用的各种 worker 线程类型有：

逻辑区线程

逻辑 线程（包括字符串 kvdo:logQ ）在逻辑块号(LBN)之间维护提供给 VDO 设备用户和基础存储系统中的物理块号(PBN)之间的映射。它们还实施锁定，以便尝试写入同一块的两个 I/O 操作不会被同时处理。逻辑区线程在读写操作过程中处于活动状态。

LBN 划分为块(块映射页面 包含 3 MB 以上的 LBN)，这些块被分成划分在线程中 的区域。

处理应该在线程中平均分配，但有些未灵活的访问模式偶尔可能集中在一个线程或其他线程中。例如，在给定块映射页面中频繁访问 LBN 将导致其中一个逻辑线程处理所有这些操作。

可以使用 vdo 命令的
--vdoLogicalThreads=thread count
选项控制逻辑区线程数量

物理区线程

物理, 或 kvdo:physQ, threads 管理数据块分配和维护参考计数。它们在写入操作过程中处于活跃状态。

与 LBNs 一样， PBNs 被分成名为 slabs 的块，这些块被进一步划分为区域， 并分配给分发处理负载的 worker 线程。

可以使用 vdo 命令的
--vdoPhysicalThreads=thread count
选项控制物理区线程数量。

I/O 提交线程

kvdo:bioQ 线程将块 I/O (bio)操作从 VDO 提交到存储系统。它们接受由其他 VDO 线程排队的 I/O 请求，并将它们传递给底层设备驱动程序。这些线程可以与设备关联的数据结构通信并更新与设备关联的数据结构，或者为设备驱动程序的内核线程设置请求进行处理。如果底层设备的请求队列已满，提交 I/O 请求可以阻止，因此此工作由专用线程完成，以避免处理延迟。

如果这些线程经常由 ps 或 top 工具显示在 D 状态，则 VDO 通常会使存储系统忙于 I/O 请求。如果存储系统可以并行服务多个请求，或者请求处理被管道，则这通常可以正常工作。如果线程 CPU 使用率在这些期间非常低，则可能会减少 I/O 提交线程的数量。

CPU 使用量和内存争用取决于 VDO 下的设备驱动程序。如果在添加更多线程时每个 I/O 请求的

CPU 使用率增加，请检查这些设备驱动程序中的 CPU、内存或锁定争用。

可以使用 vdo 命令的 --vdoBioThreads=thread count 选项控制 I/O 提交线程数量。

CPU 处理线程

kvdo:cpuQ 线程可用于执行任何 CPU 密集型工作，如计算哈希值或压缩不阻止或需要独占访问与其他线程类型关联的数据结构。

可以使用 vdo 命令的 --vdoCpuThreads=thread count 选项控制 CPU 处理线程数量。

I/O 确认线程

kvdo:ackQ 线程向位于 atop VDO 的任何位置发出回调（例如，内核页面缓存或应用程序程序线程进行直接 I/O）来报告 I/O 请求的完成。CPU 时间要求和内存争用将依赖于其他内核级别的代码。

可以使用 vdo 命令的 --vdoAckThreads=thread count 选项控制确认线程数量。

不可扩展的 VDO 内核线程：

重复数据删除线程

kvdo:dupeQ 线程使用排队的 I/O 请求和联系 UDS。由于如果服务器无法快速处理请求，或者内核内存受其他系统活动限制，则套接字缓冲区可能会填满，因此如果线程应阻止，其他 VDO 处理可以继续。还有一个超时机制，用于在较长的延迟后跳过 I/O 请求（以秒为单位）。

日志线程

kvdo:journalQ 线程更新恢复日志，并调度用于写入的日志块。VDO 设备只使用一个日志，因此无法在线程间分割工作。

packer thread

启用压缩时，kvdo:packerQ 线程在写入路径中处于活跃状态，收集 kvdo:cpuQ 线程压缩的数据块，以最小化浪费的空间。每个 VDO 设备有一个 packer 数据结构，因此每个 VDO 设备有一个 packer 线程。

30.6.3. 要调整的值

30.6.3.1. CPU/内存

30.6.3.1.1. 逻辑, 物理, cpu, ack 线程数

逻辑、物理、cpu 和 I/O 确认工作可以分布到多个线程中，可以在初始配置或之后重启 VDO 设备时指定的数量。

一个内核或一个线程可以在指定时间内执行有限的工作。只有一个线程计算所有数据块哈希值，例如，对每秒可以处理的数据块数施加硬限制。划分多个线程（和内核）的工作减轻了这种瓶颈。

当线程或核心接近 100% 的使用时，更多工作项目往往会排队进行处理。虽然这可能会导致 CPU 的闲置周期减少，但对单个 I/O 请求设置延迟和延迟通常会增加。根据某些队列模型，利用率级别高于 70% 或 80% 可能会导致超过正常处理时间的延迟。因此，对于具有 50% 或更高利用率的线程或核心，即使这些线程或内核并不总是忙碌，也可能有助于进一步分发工作。

相反，如果线程或 CPU 非常轻便加载（通常是睡眠状态），为它提供工作的可能性更有可能造成一些额外的成本。（尝试唤醒另一个线程的线程必须在调度程序的数据结构上获取全局锁定，并可能会发送处理器中断来转移到其他内核。）当将更多内核配置为运行 VDO 线程时，给定数据片段会尽可能地缓存，因为线程在线程之间移动，或者在内核之间移动时 - 因此，太多的工作分布也会降低性能。

每个 I/O 请求由逻辑、物理和 CPU 线程执行的工作将因工作负载的类型而异，因此系统应该使用服务的不同类型工作负载进行测试。

在涉及成功 deduplication 的同步模式下写入操作需要额外的 I/O 操作（读取之前存储的数据块）、一些 CPU 周期（复制新数据块以确认它们匹配），以及日志更新（将 LBN 映射到之前存储的数据块的 PBN）与新数据的写入。当以 async 模式检测到重复时，以上述读和比较操作的成本避免数据写入操作；每个写入只能发生一个日志更新，无论是否被检测到重复。

如果启用了压缩，对压缩数据进行读写将需要 CPU 线程进行更多处理。

包含所有零字节(零块)的块会特别考虑，因为它们经常发生。特殊条目用于表示块映射中的此类数

据，零块不会写入或从存储设备读取。因此，写入或读取全零块的测试可能会产生误导的结果。也是如此，到更短的测试中，写入零块或未初始化的块（自 VDO 设备创建后永远不会写入）的测试是一样的，因为零或未初始化的块不需要物理线程执行的参考计数。

确认 I/O 操作的唯一任务是不受正在完成的工作类型的影响或正在操作的数据类型，因为每个 I/O 操作都会发出一个回调。

30.6.3.1.2. CPU 关联性和 NUMA

访问 NUMA 节点边界的内存所需的时间比访问本地节点上的内存要长。当 Intel 处理器在节点上的内核间共享最后一个级别的缓存时，节点之间的缓存竞争比节点中缓存争用要大得多。

top 等工具无法区分工作和停滞周期的 CPU 周期。这些工具将缓存争用解释，并将内存访问速度减速为实际工作。因此，在节点间移动线程可能会显示以减少线程的明显 CPU 使用率，同时增加它每秒执行的操作数量。

虽然很多 VDO 内核线程维护只能被一个线程访问的数据结构，但它们会经常交换有关 I/O 请求本身的消息。如果 VDO 线程在多个节点上运行，或者由调度程序将线程从一个节点重新分配给另一个节点，则竞争可能很高。如果可以在与 VDO 线程相同的节点上运行其他与 VDO 相关的工作（如 I/O 提交到 VDO 或中断处理），则竞争可能会进一步减少。如果一个节点没有足够的周期来运行所有与 VDO 相关的工作，则当选择线程移到其他节点时，应该考虑内存争用。

如果实际情况，请使用 taskset 工具在一个节点上收集 VDO 线程。如果也可以在同一节点上运行其他与 VDO 相关的工作，这可能会进一步减少竞争。在这种情况下，如果一个节点缺少 CPU 电源来满足处理需求，那么在选择线程以移动到其他节点时需要考虑内存争用。例如，如果存储设备的驱动程序具有大量用于维护的数据结构，则可能有助于将设备的中断处理和 VDO 的 I/O 提交（调用设备的驱动程序代码）移到另一节点。使 I/O 确认（线程）和更高级别的 I/O 提交线程（用户模式线程执行直接 I/O，或者内核的页面缓存 清除 线程）对也很好。

30.6.3.1.3. 频率节流

如果电源消耗不是问题，请将字符串 性能 写入 /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor 文件（如果它们存在）可能会生成更好的结果。如果这些 sysfs 节点不存在，Linux 或系统的 BIOS 可能会提供其他选项来配置 CPU 频率管理。

性能测量会进一步复杂，CPU 根据工作负载动态变化其频率，因为完成特定工作所需的时间可能会因其他工作而不同，即使没有任务切换或缓存争用，即使没有任务切换或缓存争用，也会不同。

30.6.3.2. Caching

30.6.3.2.1. Block Map Cache

VDO 缓存多个块映射页面以提高效率。缓存大小默认为 128 MB，但可以使用 vdo 命令的 `--blockMapCacheSize=megabytes` 选项增加它。使用较大的缓存可能会给随机访问工作负载带来显著的好处。

30.6.3.2.2. 读取缓存

第二个缓存可用于缓存从存储系统读取的数据块，以验证 VDO 的 deduplication 建议。如果在短时间内看到类似的数据块，则可以减少所需的 I/O 操作数量。

读取缓存也保存包含压缩用户数据的存储块。如果在短时间内写入多个可压缩块，则它们的压缩版本可以位于同一存储系统块中。同样，如果它们在短时间内读取，缓存可能会避免需要从存储系统进行额外的读取。

vdo 命令的 `--readCache={enabled | disabled}` 选项控制是否使用读缓存。如果启用，缓存的最小大小为 8 MB，但可使用 `--readCacheSize=megabytes` 选项增加。管理读取缓存会导致开销小，因此如果存储系统足够快，则它可能无法提高性能。默认情况下禁用读取缓存。

30.6.3.3. 存储系统 I/O

30.6.3.3.1. bio 线程

对于 RAID 配置中的通用硬盘驱动器，一个或多个 bio 线程可能足以提交 I/O 操作。如果存储设备驱动程序需要其 I/O 提交线程才能显著提高工作（更新驱动程序数据结构或与设备通信），则一个或多个线程通常处于空闲状态，因此可能会增加 bio 线程数以编译。但是，根据驱动程序的实现，线程数过高可能会导致缓存或变换锁争用。如果设备访问时间不是在所有 NUMA 节点之间统一，则在节点"关闭"到存储设备控制器上运行 bio 线程会很有帮助。

30.6.3.3.2. IRQ 处理

如果设备驱动程序在其中断处理程序中无法正常工作，且没有使用线程 IRQ 处理程序，则可能会阻止调度程序提供最佳性能。为硬件中断提供服务的 CPU 时间可能类似于普通的 VDO （或其他）内核线程执行。例如，如果硬件 IRQ 处理内核周期需要 30%，则同一内核的忙碌的内核线程只能使用剩余的 70%。但是，如果为该线程排队的工作需要 80% 的核心周期，线程永远不会捕获，调度程序可能只让线程在该内核上运行，而不是将线程切换到更忙的核心。

在大量 VDO 工作负载中使用这样的设备驱动程序可能需要大量循环才能服务硬件中断(顶部显示 标头中的 %hi 指示符)。在这种情况下，可能需要为某些内核分配 IRQ 处理，并调整 VDO 内核线程的 CPU

关联性，使其不在这些内核上运行。

30.6.3.4. 最大 Discard Sectors

可以使用 /sys/kvdo/max_discard_sectors 根据系统使用情况调整 DISCARD (TRIM)操作的最大允许大小。默认值为 8 个扇区（即 4 KB 块）。可以指定较大的大小，但 VDO 仍然会在循环中处理它们，但一次一个块的一个块，确保在启动下一个块前写入一个丢弃的块的元数据更新，并刷新到磁盘。

当使用 VDO 卷作为本地文件系统时，红帽测试发现小的丢弃大小最适合工作，因为 Linux 内核中的通用块设备代码会将大型丢弃请求分成多个较小的请求，并并行提交它们。如果设备上有低 I/O 活动，VDO 可以同时处理许多较小的请求，且比一个大型请求快得多。

如果要将 VDO 设备用作 SCSI 目标，则启动器和目标软件会引入需要考虑的额外因素。如果目标 SCSI 软件是 SCST，它会读取最大丢弃大小并将其转发到启动器。（红帽没有尝试调整 VDO 配置与 LIO SCSI 目标代码。）

因为 Linux SCSI 启动器代码一次只允许一个丢弃操作，因此丢弃超过最大大小的请求会分为多个较小的丢弃并发送，一次发送到目标系统（和 VDO）。因此，除了 VDO 处理串行中的多个小丢弃操作外，两个系统之间的往返通信时间会增加延迟。

设置较大的最大丢弃大小可减少这个通信开销，虽然大型请求将在整个 VDO 传递给 VDO 并一次处理一个 4 KB 块。虽然没有每个块的通信延迟，但较大的块的额外处理时间可能会导致 SCSI 启动器软件超时。

对于 SCSI 目标使用情况，红帽建议将最大丢弃大小配置为中等，同时仍然在启动器的超时设置中保持典型的丢弃时间良好。例如，每几秒钟的额外往返成本（例如，如果 30 秒或 60 秒超时）不应显著影响性能和 SCSI 启动器。

30.6.4. 识别 Bottlenecks

有几个影响 VDO 性能的关键因素，以及很多工具来识别影响最大影响的工具。

如 top 或 ps 等实用程序中所示，线程或 CPU 使用率高于 70%，通常意味着过多的工作被集中在一个线程或一个 CPU 上。然而，在某些情况下，可能意味着 VDO 线程被调度到 CPU 上运行，但没有实际发生任何工作；这种情况可能会在硬件中断处理器处理、内核或 NUMA 节点之间内存争用，或者对 spin 锁定争用。

当使用 top 工具检查系统性能时，红帽建议运行 top -H 来分别显示所有进程线程，然后输入 1 f j

键，后跟 Enter/Return 键；然后，顶部 命令显示单个 CPU 内核的负载，并确定每个进程或线程最后一次运行的 CPU。这些信息可以提供以下 insights：

- 如果内核没有较低的 %id （空闲）和 %wa （等待I/O）值，则它会一直保持在某种程度上工作。

- 如果内核的 %hi 值非常低，则核心正在进行正常处理，这是由内核调度程序进行负载均衡的。向该集合添加更多内核可能会减少负载，只要它不引入 NUMA 争用。

- 如果内核的 %hi 超过百分比，且只为那个内核分配一个线程，%id 和 %wa 为零，则核心将被过度使用，且调度程序不会解决这种情况。在这种情况下，应重新分配内核线程或设备中断处理，使其保存在单独的内核中。

perf 工具可以检查许多 CPU 的性能计数器。红帽建议使用 perf top 子命令作为检查线程或处理器正在执行的工作的起点。例如，bioQ 线程会花费很多周期试图获取 spin 锁定，则 VDO 下设备驱动程序可能会太多竞争，并减少 bioQ 线程的 数量可能缓解这种情况。使用高 CPU （获取 spin 锁定或其他）也可以表示 NUMA 节点之间争用（例如，bio Q 线程和设备中断处理器在不同节点上运行。如果处理器支持它们，则诸如 stalled-cycles-backend、cache-misses 和 node-load-misses 等计数器可能值得关注。

sar 实用程序可以提供关于多个系统统计的定期报告。sar -d 1 命令报告块设备利用率级别（它们至少有一个 I/O 操作百分比）和队列长度（每秒等待的 I/O 请求数）一次。但是，并非所有块设备驱动程序都可以报告此类信息，因此 sar 有用的性可能会依赖于正在使用的设备驱动程序。

30.7. VDO 命令

本节描述了以下 VDO 工具：

vdo

vdo 实用程序管理 VDO 的 kvdo 和 UDS 组件。

它还用于启用或禁用压缩。

vdostats

vdostats 工具以类似 Linux df 实用程序的格式显示每个配置（或指定）设备的统计信息。

30.7.1. vdo

vdo 实用程序管理 VDO 的 kvdo 和 UDS 组件。

概要

```
vdo { activate | changeWritePolicy | create | deactivate | disableCompression | disableDeduplication |
enableCompression | enableDeduplication | growLogical | growPhysical | list | modify | printConfigFile
| remove | start | status | stop }
[ options... ]
```

sub-Commands

表 30.4. VDO Sub-Commands

sub-Command	描述

sub-Command	描述
create	创建 VDO 卷及其关联的索引，并使其可用。如果指定了 **osgiactivate=disabled**，则创建 VDO 卷但不可用。除非给出 zFCP **force**，否则不会覆盖现有的文件系统或格式化的 VDO 卷。此命令必须使用 root 特权运行。适用的选项包括： • **--name=*volume***（必需） • **--device=*device***（必需） • **--activate={enabled | disabled}** • **--indexMem=*gigabytes*** • **--blockMapCacheSize=*megabytes*** • **--blockMapPeriod=*period*** • **--compression={enabled | disabled}** • **--confFile=*file*** • **--deduplication={enabled | disabled}** • **--emulate512={enabled | disabled}** • **--sparseIndex={enabled | disabled}** • **--vdoAckThreads=*thread count*** • **--vdoBioRotationInterval=*I/O count*** • **--vdoBioThreads=*thread count*** • **--vdoCpuThreads=*thread count*** • **--vdoHashZoneThreads=*thread count*** • **--vdoLogicalThreads=*thread count*** • **--vdoLogLevel=*level*** • **--vdoLogicalSize=*megabytes*** • **--vdoPhysicalThreads=*thread count*** • **--readCache={enabled | disabled}** • **--readCacheSize=*megabytes*** • **--vdoSlabSize=*megabytes*** • **--verbose** • **--writePolicy={ auto | sync | async }** • **--logfile=pathname**

sub-Command	描述
remove	删除一个或多个已停止的 VDO 卷和相关索引。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--force** • **--verbose** • **--logfile=pathname**
开始	启动一个或多个已停止的、激活的 VDO 卷和相关服务。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--forceRebuild** • **--verbose** • **--logfile=pathname**
stop	停止一个或多个正在运行的 VDO 卷和相关服务。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--force** • **--verbose** • **--logfile=pathname**
激活	激活一个或多个 VDO 卷。可使用 **开始** 命令启动激活的卷。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--logfile=pathname** • **--verbose**

sub-Command	描述
deactivate	取消激活一个或多个 VDO 卷。取消激活的卷无法通过 **开始** 命令启动。停用当前运行的卷不会停止它。在停止停用的 VDO 卷后，必须先激活它，然后才能再次启动。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
status	以 YAML 格式报告 VDO 系统和卷状态。如果没有运行，这个命令不需要 root 特权，但信息将不完整。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--verbose** • **--logfile=pathname** 有关提供的输出，请参阅 表 30.6 “VDO 状态输出”。
list	显示启动的 VDO 卷列表。**如果指定了 osgiall**，它将显示启动和非启动的卷。此命令必须使用 root 特权运行。适用的选项包括： • **--all** • **--confFile=*file*** • **--logfile=pathname** • **--verbose**

sub-Command	描述
修改	修改一个或多个 VDO 卷的配置参数。更改在下次启动 VDO 设备时生效；已在运行的设备不受影响。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--blockMapCacheSize=*megabytes*** • **--blockMapPeriod=*period*** • **--confFile=*file*** • **--vdoAckThreads=*thread count*** • **--vdoBioThreads=*thread count*** • **--vdoCpuThreads=*thread count*** • **--vdoHashZoneThreads=*thread count*** • **--vdoLogicalThreads=*thread count*** • **--vdoPhysicalThreads=*thread count*** • **--readCache={enabled | disabled}** • **--readCacheSize=*megabytes*** • **--verbose** • **--logfile=pathname**
changeWritePolicy	修改一个或多个正在运行的 VDO 卷的写入策略。此命令必须使用 root 特权运行。 • { **--name=*volume*** | **--all** }（必需） • **--writePolicy={ auto | sync | async }** （必需） • **--confFile=*file*** • **--logfile=pathname** • **--verbose**
enableDeduplication	在一个或多个 VDO 卷中启用 deduplication。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--verbose** • **--logfile=pathname**

sub-Command	描述
disableDeduplication	在一个或多个 VDO 卷中禁用 deduplication。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
启用压缩	在一个或多个 VDO 卷中启用压缩。如果 VDO 卷正在运行，请立即生效。如果 VDO 卷没有运行压缩，则在下次启动 VDO 卷时将启用。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
禁用压缩	禁用一个或多个 VDO 卷的压缩。如果 VDO 卷正在运行，请立即生效。如果 VDO 卷没有运行压缩，则在下次启动 VDO 卷时将禁用。此命令必须使用 root 特权运行。适用的选项包括： • { **--name=*volume*** | **--all** }（必需） • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
growLogical	在 VDO 卷中添加逻辑卷。卷必须存在，必须正在运行。此命令必须使用 root 特权运行。适用的选项包括： • **--name=*volume***（必需） • **--vdoLogicalSize=*megabytes*** （必需） • **--confFile=*file*** • **--verbose** • **--logfile=pathname**

sub-Command	描述
growPhysical	在 VDO 卷中添加物理空间。卷必须存在，必须正在运行。此命令必须使用 root 特权运行。适用的选项包括： • **--name=*volume***（必需） • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
printConfigFile	将配置文件输出到 **stdout**。此命令需要 root 特权。适用的选项包括： • **--confFile=*file*** • **--logfile=pathname** • **--verbose**

选项

表 30.5. VDO 选项

选项	描述
--indexMem=*gigabytes*	以 GB 为单位指定 UDS 服务器内存量；默认大小为 1 GB。可以使用特殊的十进制值 0.25、0.5、0.75，就像任意正整数一样。
--sparseIndex={enabled | disabled}	启用或禁用稀疏索引。默认值为 **。**
--all	表示命令应应用到所有配置的 VDO 卷。不得与 **--name** 一起使用。
--blockMapCacheSize=*megabytes*	指定为缓存块映射页面分配的内存大小；该值必须是 4096 的倍数。使用带有 **B**(ytes), **K**(ilobytes), **M**(egabytes), **G**(igabytes), **T**(erabytes), **P**(etabytes)或 **E**(xabytes)后缀的值是可选的。如果没有提供后缀，则该值将解释为 MB。默认值为 128M；该值必须至少为 128M，且小于 16T。请注意，内存开销为 15%。
--blockMapPeriod=*period*	1 到 16380 之间的值，它决定了在缓存页面刷新到磁盘前可能会累积的块映射更新数量。数值越高，在正常操作期间，崩溃会降低性能降低后的恢复时间。默认值为 16380。在调整这个参数前，请联系您的红帽代表。
--compression={enabled | disabled}	启用或禁用 VDO 设备中的压缩。默认启用默认值。如果需要，可以禁用压缩，以加快对性能的处理速度，不太可能压缩。
--confFile=*file*	指定备用配置文件。默认值为 **/etc/vdoconf.yml**。

选项	描述
--deduplication={enabled | disabled}	在 VDO 设备中启用或禁用 deduplication。**默认是启用的**。在数据没有很好的重复数据删除率但仍然需要压缩的情况下，可能会禁用 deduplication。
--emulate512={enabled | disabled}	启用 512 字节块设备模拟模式。默认值为 **。**
--force	在停止 VDO 卷前卸载挂载的文件系统。
--forceRebuild	在启动只读 VDO 卷前强制离线重建，以便它可以重新上线并可用。**这个选项可能会导致数据丢失或损坏。**
--help	显示 **vdo** 实用程序的文档。
--logfile=pathname	指定此脚本的日志消息定向到的文件。警告和错误消息也始终记录到 syslog。
--name=*volume*	在指定的 VDO 卷中操作。不得与 **--all** 一起使用。
--device=*device*	指定用于 VDO 存储的设备的绝对路径。
--activate={enabled | disabled}	**禁用** 的参数表示仅应创建 VDO 卷。卷不会被启动或启用。**默认是启用的**。
--vdoAckThreads=*thread count*	指定用于确认请求的 VDO I/O 操作的完成线程数量。默认值为 1；该值必须至少为 0，且小于或等于 100。
--vdoBioRotationInterval=*I/O count*	在将工作定向到下一个操作前，指定每个 bio-submission 线程要排队的 I/O 操作数量。默认值为 64；该值必须至少为 1，且小于或等于 1024。
--vdoBioThreads=*thread count*	指定用于向存储设备提交 I/O 操作的线程数量。最小为 1; 最大为 100。默认值为 4；该值必须至少为 1，且小于或等于 100。
--vdoCpuThreads=*thread count*	指定用于 CPU 密集型工作的线程数量，如哈希或压缩。默认值为 2，该值必须至少为 1，且小于或等于 100。
--vdoHashZoneThreads=*thread count*	根据从块数据计算的哈希值，指定 VDO 处理中分离的线程数量。默认值为 **1**; 该值必须至少为 0，且小于或等于 100。**vdoHashZoneThreads**、**vdoLogicalThreads** 和 **vdoPhysicalThreads** 都必须为零或所有非零。
--vdoLogicalThreads=*thread count*	根据从块数据计算的哈希值，指定 VDO 处理中分离的线程数量。该值必须至少为 0，且小于或等于 100。9 或更多个逻辑线程数还需要明确指定足够大的块映射缓存大小。**vdoHashZoneThreads**,**vdoLogicalThreads**, 和 **vdoPhysicalThreads** 必须是零或所有非零。默认值为 1。

选项	描述
--vdoLogLevel=*level*	指定 VDO 驱动程序日志级别：**critical**、**error**、**warning**、**notice**、**info** 或 **debug**。级别是区分大小写的；默认值为 **info**。
--vdoLogicalSize=*megabytes*	以 MB 为单位指定逻辑卷大小。使用带有 **S**(ectors)、**B**(ytes)，**K**(ilobytes)，**M**(egabytes)，**G**(igabytes)，**T**(erabytes)，**P**(etabytes)或 **E**(xabytes)后缀的值是可选的。用于过度置备卷。默认为存储设备的大小。
--vdoPhysicalThreads=*thread count*	指定根据物理块地址在 VDO 处理中从属部分的线程数量。该值必须至少为 0，且小于或等于 16。第一个后的每个额外线程将使用额外的 10 MB RAM。**vdoPhysicalThreads**,**vdoHashZoneThreads**, 和 **vdoLogicalThreads** 必须是零或所有非零。默认值为 1。
--readCache={enabled | disabled}	启用或禁用 VDO 设备中的读取缓存。默认值为 **。如果写入工作负载应该具有高级别 deduplication，或者用于读取高压缩数据的工作负载，则应启用缓存。**
--readCacheSize=*megabytes*	指定额外的 VDO 设备读取缓存大小（以 MB 为单位）。这个空间除一个系统定义的最小之外。使用带有 **B**(ytes)，**K**(ilobytes)，**M**(egabytes)，**G**(igabytes)，**T**(erabytes)，**P**(etabytes)或 **E**(xabytes)后缀的值是可选的。默认值为 0M。每个 bio 线程将每 MB 读缓存使用 1.12 MB 内存。
--vdoSlabSize=*megabytes*	指定 VDO 增长的递增大小。使用较小的大小限制了可容纳的总物理大小。必须是 128M 到 32G 之间的 2 个电源；默认值为 2G。使用带有 **S**(ectors)、**B**(ytes)，**K**(ilobytes)，**M**(egabytes)，**G**(igabytes)，**T**(erabytes)，**P**(etabytes)或 **E**(xabytes)后缀的值是可选的。如果没有使用后缀，则该值将解释为 MB。
--verbose	在执行命令前打印命令。
--writePolicy={ auto | sync | async }	指定写入策略： • **auto** ：根据 VDO 下的存储层选择 **sync** 或 **async**。如果存在写回缓存，则会选择 **async**。否则，将选择 **同步**。 • **同步** ：只有数据被写入后，才会确认写。这是默认策略。**如果底层存储也没有同步，则不支持此策略。** • **async**: Writes 在数据 被缓存 后被确认，以写入稳定存储。未清除的数据无法保证在此模式下保留。

status

子命令返回 YAML 格式的以下信息，并分为几个键，如下所示：

表 30.6. VDO 状态输出

键	描述	
VDO 状态	此密钥中的信息涵盖正在进行状态的主机和日期和时间的名称。在此区域中报告的参数包括：	
	节点	运行 VDO 的系统的主机名。
	Date	运行 vdo status 命令的日期和时间。
内核模块	这个密钥中的信息涵盖了配置的内核。	
	Loaded	是否载入内核模块(True 或 False)。
	版本信息	已配置的 kvdo 版本的信息。
配置	此密钥中的信息涵盖了 VDO 配置文件的位置和状态。	
	File	VDO 配置文件的位置。
	最后修改	VDO 配置文件的最后修改日期。
VDO	为所有 VDO 卷提供配置信息。为每个 VDO 卷报告的参数包括：	
	块大小	VDO 卷的块大小，以字节为单位。
	512 字节模拟	指明卷是否在 512 字节模拟模式下运行。
	启用 deduplication	是否为卷启用 deduplication。
	逻辑大小	VDO 卷的逻辑大小。
	物理大小	VDO 卷的基本物理存储的大小。
	写入策略	写入策略配置的值(sync 或 async)。
	VDO 统计	vdostats 工具的输出。

30.7.2. vdostats

vdostats 工具以类似 Linux df 实用程序的格式显示每个配置（或指定）设备的统计信息。

如果没有使用 root 特权运行，则 vdostats 工具的输出可能不完整。

概要

```
vdostats [ --verbose | --human-readable | --si | --all ] [ --version ] [ device ...]
```

选项

表 30.7. vdostats 选项

选项	描述
--verbose	显示一个或多个 VDO 设备的利用率和块 I/O (bios)统计信息。详情请查看 表 30.9 “vdostats --verbose Output”。
--human-readable	以可读的形式显示块值（基本 2: 1 KB = 2¹⁰ 字节 = 1024 字节）。
--si	**--si** 选项修改 **--human-readable** 选项的输出以使用 SI 单位（基本 10: 1 KB = 10³ 字节 = 1000 字节）。如果没有提供 **--human-readable** 选项，-- **si** 选项不会起作用。
--all	这个选项仅用于向后兼容。现在，它等同于 **--verbose** 选项。
--version	显示 **vdostats** 版本。
device ...	指定要报告的一个或多个特定卷。如果省略此参数，则 **vdostats** 将报告所有设备。

输出

以下示例显示，如果没有提供选项，则输出示例，在 表 30.8 “默认 vdostats 输出” 中进行了描述：

```
Device                1K-blocks    Used       Available    Use%   Space Saving%
/dev/mapper/my_vdo    1932562432   427698104  1504864328   22%    21%
```

表 30.8. 默认 vdostats 输出

项	描述
设备	VDO 卷的路径。
1k-blocks	为 VDO 卷分配的 1K 块总数(=物理卷大小 * 块大小 / 1024)
Used	VDO 卷中使用的 1K 块总数(= 物理块使用 * 块大小 / 1024)
可用	VDO 卷中可用 1K 块的总数(= 物理块 free * 块大小 / 1024)
use%	VDO 卷中使用的物理块的百分比(= 使用块 / 分配的块 * 100)
空间节省%	VDO 卷中保存的物理块的百分比(= [逻辑块已使用 - 物理块] / logical blocks used)

项	描述

--human-readable 选项将块数转换为传统单元(1 KB = 1024 字节)：

```
Device              Size   Used     Available     Use%   Space Saving%
/dev/mapper/my_vdo   1.8T   407.9G    1.4T          22%    21%
```

--human-readable 和 --si 选项将块数转换为 SI 单位(1 KB = 1000 字节)：

```
Device              Size   Used     Available     Use%    Space Saving%
/dev/mapper/my_vdo   2.0T   438G      1.5T          22%     21%
```

--verbose (表 30.9 “vdostats --verbose Output”)选项以 YAML 格式显示一个（或全部）VDO 设备的 VDO 设备统计信息。

在以后的发行版本中，以 粗体显示 的统计信息将继续报告。表 30.9 “vdostats --verbose Output”其余字段主要用于软件支持，可能在以后的版本中有所变化；管理工具不应依赖于它们。管理工具不应依赖于报告任何统计数据的顺序。

表 30.9. vdostats --verbose Output

项	描述
版本	这些统计数据的版本。
发行版本	VDO 的发行版本。
已使用的数据块	VDO 卷目前使用的物理块数量来存储数据。
使用的开销块	VDO 卷目前使用的物理块数量来存储 VDO 元数据。
使用的逻辑块	当前映射的逻辑块数量。
物理块	为 VDO 卷分配的物理块总数。
逻辑块	VDO 卷可以映射的最大逻辑块数。

项	描述
1k-blocks	为 VDO 卷分配的 1K 块总数(=物理卷大小 * 块大小 / 1024)
1k-blocks 使用的	VDO 卷中使用的 1K 块总数(= 物理块使用 * 块大小 / 1024)
1k-blocks 可用	VDO 卷中可用 1K 块的总数(= 物理块 free * 块大小 / 1024)
已使用百分比	VDO 卷中使用的物理块的百分比(= 使用块 / 分配的块 * 100)
保存百分比	VDO 卷中保存的物理块的百分比(= [逻辑块已使用 - 物理块] / logical blocks used)
块映射缓存大小	块映射缓存的大小，以字节为单位。
写入策略	活跃的写入策略(sync 或 async)。这通过 **vdo changeWritePolicy --writePolicy=auto|sync|async** 配置。
块大小	VDO 卷的块大小，以字节为单位。
完成恢复计数	VDO 卷已从未清理关闭中恢复的次数。
只读恢复计数	VDO 卷已从只读模式恢复的次数（通过 **vdo start --forceRebuild**）。
操作模式	指明 VDO 卷是否正常运行，处于恢复模式，或者处于只读模式。
恢复进度(%)	指明在线恢复进度，如果卷没有处于恢复模式，则为 **N/A**。
编写的压缩片段	从 VDO 卷最后一次重启后写入的压缩片段数量。
编写的压缩块	从 VDO 卷最后一次重启后写入的压缩数据的物理块数量。
packer 中的压缩片段	尚未写入的压缩片段数量。
slab 数量	slabs 的总数。
slabs 已打开	从中分配块的 slab 总数。
slabs reopened	从 VDO 启动后，会重新打开 slabs 的次数。
日志磁盘完整计数	请求无法进行恢复日志条目的次数，因为恢复日志已满。
日志提交请求计数	恢复日志请求 slab 日志提交的次数。
日志条目批处理	日志条目写入的数量已开始减去写入的日志条目数。
日志条目已启动	内存中所做的日志条目数。

项	描述
编写日志条目	提交中的日志条目数量减去提交到存储的日志条目数。
写入的日志条目	签发写入的日志条目总数。
已提交的日志条目	写入存储的日志条目数。
日志块批处理	启动的日志块写入数量减去写入的日志块的数量。
日志块已启动	在内存中涉及的日志块数。
日志块编写	编写的日志块数量（具有活动内存中的元数据主机）减去所提交的日志块数。
写入的日志条目	发出写入的日志块总数。
已提交的日志块	写入存储的日志块数量。
slab 日志磁盘完整计数	磁盘上 slab 日志已满的次数。
slab 日志清除计数	条目添加到通过 flush 阈值的 slab 日志中的次数。
slab 日志阻止计数	条目添加到带有阻塞阈值的 slab 日志中的次数。
写入 slab 日志块	发出 slab 日志块写入的数量。
slab journal tail busy count	写入请求阻止等待 slab 日志写入的次数。
写入 slab 摘要块	签发的 slab 摘要块写入的数量。
编写的引用块	发布的引用块写入数量。
块映射脏页面	块映射缓存中脏页面的数量。
块映射清理页面	块映射缓存中清理页面的数量。
块映射空闲页面	块映射缓存中可用页面的数量。
块映射失败页	具有写入错误的块映射缓存页面数量。
块映射传入的页面	被读取到缓存中的块映射缓存页面数量。
块映射传出页面	正在写入的块映射缓存页面数量。
块映射缓存压力	在需要时，空闲页面不可用的次数。

项	描述
块映射读取计数	块映射页面的总数显示为：
块映射写入计数	块映射页面写入的总数。
块映射失败的读取	块映射读取错误的总数。
块映射失败的写入	块映射写入错误的总数。
重新声明的块映射	已回收的块映射页面总数。
块映射读取传出	要写入的页面的块映射读取总数。
缓存中找到的块映射	块映射缓存命中的总数。
需要块映射丢弃	需要丢弃页面的块映射请求总数。
块映射等待页面	必须等待页面的请求总数。
需要块映射获取	需要页面获取的请求总数。
加载的块映射页面	页面获取总数。
保存的块映射页面	保存页面的总数。
块映射冲刷计数	块映射发出的刷新总数。
无效的建议 PBN 数量	索引返回无效建议的次数
没有空格错误计数。	由于 VDO 卷没有空间导致的写入请求数。
只读错误计数	因为 VDO 卷处于只读模式而失败的写入请求数。
实例	VDO 实例。
512 字节模拟	指明 512 字节模拟是否为卷打开或关闭。
正在进行中的当前 VDO IO 请求。	VDO 当前处理的 I/O 请求数量。
正在进行中的最大 VDO IO 请求	VDO 并发处理的最大 I/O 请求数。
当前去除重复查询	当前在 flight 中去除重复数据查询的数量。
最大去除重复查询	动态重复数据删除查询的最大数量。

项	描述
Dedupe 建议有效	数据重复数据建议的次数正确。
dedupe 建议过时	重复数据删除建议的次数不正确。
dedupe 建议超时	重复数据删除查询超时的次数。
flush out	VDO 提交到底层存储的清空请求数。
中的 BIOS...部分的 BIOS...BIOS out...BIOS 元...BIOS 日志...BIOS 页面缓存...BIOS 已完成...bio meta completed...BIOS 日志已完成...已完成 BIOS 页面缓存...BIOS 已确认...BIOS 确认的部分...正在进行中的 BIOS...	这些统计数据使用给定标志计算每个类别中的 bios 数量。类别是： • **BIOS in：** VDO 接收的块 I/O 请求数。 • **区域中 BIOS：** VDO 收到的部分块 I/O 请求数量。仅适用于 512 字节模拟模式。 • **BIOS out ：** 由 VDO 提交到存储设备的非元数据块 I/O 请求的数量。 • **BIOS meta ：** VDO 向存储设备提交的元数据块 I/O 请求数量。 • **BIOS 日志：** VDO 提交到存储设备的恢复日志块 I/O 请求数量。 • **BIOS 页面缓存：** VDO 向存储设备提交的块映射 I/O 请求数。 • **BIOS out completed ：** 由存储设备完成的非元数据块 I/O 请求的数量。 • **BIOS meta completed ：** 存储设备完成的元数据块 I/O 请求数量。 • **BIOS 日志完成：** 由存储设备完成恢复日志块 I/O 请求的数量。 • **BIOS 页面缓存完成：存储设备完成的** 块映射 I/O 请求数量。 • **BIOS 已确认：** VDO 已确认的块 I/O 请求数。 • **BIOS 确认部分： VDO 已确认** 部分块 I/O 请求的数量。仅适用于 512 字节模拟模式。 • **BIOS in progress：** 提交至 VDO 的 bios 数（还没有被确认）。 标记有三种类型： • **Read：** 非写入 bios 的数量（未设置 REQ_WRITE 标志） • **Write ：** 写入 bios （设置了 REQ_WRITE 标志的bios） • **discard ：** 带有 REQ_DISCARD 标志集的 bios 数量
读取缓存访问	VDO 搜索读取缓存的次数。
读取缓存命中	读取缓存命中的数量。

30.8. /SYS中的统计文件

正在运行的 VDO 卷的统计信息可以从 /sys/kvdo/volume_name/statistics 目录中的文件读取，其中 volume_name 是 thhe VDO 卷的名称。这为 vdostats 工具生成的数据提供了一个备用接口，适合由

shell 脚本和管理软件访问。

除了下表中列出的文件外，统计 目录中还有一些文件。在以后的发行版本中无法保证支持这些额外的统计文件。

表 30.10. 统计文件

File	描述
dataBlocksUsed	VDO 卷目前使用的物理块数量来存储数据。
logicalBlocksUsed	当前映射的逻辑块数量。
physicalBlocks	为 VDO 卷分配的物理块总数。
logicalBlocks	VDO 卷可以映射的最大逻辑块数。
模式	指明 VDO 卷是否正常运行，处于恢复模式，或者处于只读模式。

第 31 章 VDO 评估

31.1. 简介

VDO 是为主存储提供内联块级重复数据删除、压缩和精简置备功能的软件。VDO 在 Linux 设备映射器框架中安装，它拥有现有物理块设备的所有权，并将这些设备重新映射到具有数据效率属性的新的更高级别块设备。具体来说，VDO 可将这些设备的有效容量乘以十或更多。这些好处需要额外的系统资源，因此需要测量 VDO 对系统性能的影响。

存储供应商无疑存在内部测试计划和专业知识，他们用来评估新的存储产品。由于 VDO 层有助于识别重复数据删除和压缩，因此可能需要不同的测试。有效的测试计划需要研究 VDO 架构并探索这些项目：

- 特定于 VDO 的可配置属性（性能调优最终用户应用程序）
- 作为原生 4 KB 块设备的影响
- 响应重复数据删除和压缩的访问模式和发布
- 高负载环境中的性能（非常重要）
- 根据应用程序分析成本与容量与性能

由于无法考虑此类因素，则预先创建具有无效测试以及需要客户重复测试和数据收集工作的情况。

31.1.1. 期望和交付

本评估指南旨在增加供应商的内部评估工作，而不是替换。通过大量时间的投资，它可以帮助评估 VDO 整合到现有存储设备中的准确评估。本指南旨在：

- 帮助工程师识别来自测试设备的最佳响应的配置设置
- 提供对基本调优参数的信息，以帮助避免产品错误配置

- 创建一个性能结果组合，作为与"真实"应用程序结果进行比较的参考
- 识别不同的工作负载对性能和数据效率的影响
- 使用 VDO 实现加快产品面市时间

测试结果将帮助红帽工程师在集成到特定存储环境中时协助了解 VDO 的行为。OEM 将会了解如何设计其重复数据删除和压缩功能的设备，以及他们的客户如何调整其应用程序以最佳地使用这些设备。

请注意，本文档中的步骤旨在提供可实际评估 VDO 的条件。更改测试程序或参数可能会导致结果无效。红帽销售工程师可在修改测试计划时提供指导。

31.2. 测试环境准备

在评估 VDO 前，务必要考虑主机系统配置、VDO 配置以及测试过程中使用的工作负载。在数据优化（空间效率）和性能（带宽和延迟）方面，这些选择都会影响基准测试。以下几节中列出了开发测试计划时应考虑的项目。

31.2.1. 系统配置

- 可用 CPU 内核数和类型。这可以通过使用 taskset 实用程序来控制。
- 可用内存和总安装内存.
- 配置存储设备。
- Linux 内核版本.请注意，Red Hat Enterprise Linux 7 只提供一个 Linux 内核版本。
- 安装了软件包。

31.2.2. VDO 配置

- 分区方案
- VDO 卷中使用的文件系统
- 分配给 VDO 卷的物理存储大小
- 创建的逻辑卷的大小
- 稀疏或高密度索引
- 内存大小的 UDS Index
- VDO 的线程配置

31.2.3. 工作负载

- 生成测试数据的工具类型
- 并发客户端数
- 写入数据中重复的 4 KB 块的数量
- 读和写的特征
- 工作集大小

可能需要在某些测试之间重新创建 VDO 卷，以确保每个测试都在同一磁盘环境中执行。在测试部分阅读有关此内容的更多信息。

31.2.4. 支持的系统配置

红帽已在 Intel 64 构架中使用 Red Hat Enterprise Linux 7 测试了 VDO。

有关 VDO 的系统要求，请参阅 第 30.2 节 “系统要求”。

在评估 VDO 时，建议使用以下工具：

- 灵活的 I/O Tester 版本 2.08 或更高版本；可从 fio 软件包获得
- sysstat 版本 8.1.2-2 或更高版本；可从 sysstat 软件包获得

31.2.5. 预测试系统准备

这部分论述了如何配置系统设置以便在评估过程中获得最佳性能。在任何特定测试中建立的隐式范围之外测试可能会导致因为正常结果导致测试时间丢失。例如，本指南描述了一个测试，它对 100 GB 地址范围执行随机读取。要测试一组 500 GB，为 VDO 块映射缓存分配的 DRAM 数量应该相应地增加。

- 系统配置
  - 确定您的 CPU 在最高级别的性能设置中运行。
  - 如果使用 BIOS 配置或 Linux cpupower 工具，禁用频率扩展。
  - 如果可能达到最大吞吐量，请启用 Turbo 模式。turbo 模式在测试结果中引入了一些差异，但性能将达到或超过没有 Turbo 的测试。
- Linux 配置
  - 对于基于磁盘的解决方案，Linux 提供多个 I/O 调度程序算法，以在多个读写请求排队时进行处理。默认情况下，Red Hat Enterprise Linux 使用 CFQ（完全公平排队）调度程序，它以改进旋转磁盘（硬磁盘）访问的方式排列请求。我们建议将死线调度程序用于旋转磁盘，发现它在红帽实验室测试中提供更好的吞吐量和延迟。按如下方式更改设备设置：

```
# echo "deadline" > /sys/block/device/queue/scheduler
```

- 对于基于闪存的解决方案，noop 调度程序演示了红帽实验室测试中卓越的随机访问吞吐量和延迟。按如下方式更改设备设置：

```
# echo "noop" > /sys/block/device/queue/scheduler
```

- 存储设备配置

文件系统(ext4、XFS 等)可能会对性能有唯一影响；它们通常会降低性能，从而更难以隔离 VDO 对结果的影响。如果合理，我们建议测量原始块设备的性能。如果无法做到这一点，请使用目标实施中使用的文件系统格式化设备。

31.2.6. VDO 内部结构

我们相信，对 VDO 机制的总体了解对于完整且成功评估至关重要。当测试计划中希望开发测试计划或设计新的 stimuli 以模拟特定应用程序或用例时，这种理解尤为重要。如需更多信息，请参阅 第 30 章 VDO 集成。

红帽测试计划被编写为使用默认 VDO 配置进行操作。在开发新测试时，必须调整下一节中列出的一些 VDO 参数。

31.2.7. VDO 优化

高负载

或许，生成最佳性能的最重要策略是确定最佳 I/O 队列深度，这是代表存储系统上负载的特征。大多数现代存储系统以高 I/O 深度进行最佳性能。VDO 的性能最好与多个并发请求一起演示。

同步与.异步写策略

VDO 可使用两个写入策略之一进行操作，即同步或异步。默认情况下，VDO 会自动为您的底层存储设备选择适当的写入策略。

测试性能时，您需要知道选择哪个写入策略 VDO。以下命令显示 VDO 卷的写入策略：

```
# vdo status --name=my_vdo
```

有关写入策略的详情，请参考 “VDO 写策略概述”一节 和 第 30.4.2 节 “选择 VDO 写入模式”。

元数据缓存

VDO 维护从逻辑块地址到物理块地址的映射表，VDO 在访问任何特定块时必须查找相关的映射。默认情况下，VDO 在 DRAM 中分配 128 MB 的元数据缓存，以支持一次高效地访问 100 GB 的逻辑空间。测试计划生成适合此配置选项的工作负载。

大于配置的缓存大小的工作集合将需要额外的 I/O 来服务请求，在这种情况下会出现性能下降。如果有额外的内存可用，则块映射缓存应该更大。如果工作集大于内存中可以保存的块映射缓存，则可能需要进行额外的 I/O 悬停的头来查找相关的块映射页面。

VDO 多线程配置

必须调整 VDO 的线程配置才能获得最佳性能。有关在创建 VDO 卷时如何修改这些设置的信息，请参阅 VDO 集成指南。请联系您的红帽销售工程师，讨论如何设计测试以查找最佳设置。

数据内容

由于 VDO 执行重复数据删除和压缩，因此必须选择测试数据集来有效地处理这些功能。

31.2.8. 测试读性能的特别注意事项

在测试读取性能时，必须考虑以下因素：

1. 如果 从未写入 4 KB 块，VDO 不会对存储执行 I/O，并将立即使用零块响应。
2. 如果 写入了 4 KB 块但包含所有零，VDO 不会对存储执行 I/O，并将立即使用零块响应。

当没有可读取数据时，读取性能会非常快。这使得读取测试需要预先填充实际数据。

31.2.9. cross Talk

要防止一个测试影响另一个测试的结果，建议为每个测试的每个迭代创建一个新的 VDO 卷。

31.3. 数据效率测试过程

VDO 成功验证取决于遵循一个精心设计的测试过程。本节提供了一系列后续步骤，以及预期结果，作为参与评估时需要考虑的测试示例。

测试环境

下一节中的测试案例对测试环境进行以下假设：

- 一个或多个 Linux 物理块设备可用。
- 目标块设备（例如 /dev/sdb）大于 512 GB。
- 已安装灵活的 I/O Tester (fio)版本 2.1.1 或更高版本。
- 已安装 VDO。

以下信息应该在每个测试开始时记录，以确保测试环境被完全理解：

- 使用的 Linux 构建，包括内核构建号。
- 从 rpm -qa 命令获取安装的软件包的完整列表。
- 完整的系统规格：
  - CPU 类型和数量（可在 /proc/cpuinfo 中提供）。
  - 安装的内存以及基础操作系统运行后可用（可在 /proc/meminfo 中使用）。
  - 使用的驱动器控制器的类型。

- 使用的磁盘的类型和数量。

- 正在运行的进程的完整列表（从 ps aux 或类似的列表）。

- 物理卷的名称以及为 VDO 创建的卷组名称(pvs 和 vgs 列表)。

- 格式化 VDO 卷时使用的文件系统（若有）。

- 挂载的目录的权限。

- /etc/vdoconf.yaml 的内容。

- VDO 文件的位置。

您可以通过运行 sosreport 来捕获大量所需的信息。

工作负载

有效地测试 VDO 需要使用模拟实际工作负载的数据集。数据集应在可以重复数据删除和/或压缩的数据之间提供平衡，这些数据无法在不同条件下演示性能。

有几个工具可以同步生成具有可重复特征的数据。特别是 VDbench 和 fio 的两个实用程序，建议在测试过程中使用。

本指南使用 fio。了解参数对于成功评估至关重要：

表 31.1. fio 选项

参数	描述	值
--size	数据 fio 将为每个作业发送到目标（请参阅以下 **numjobs**）。	100 GB

参数	描述	值
--bs	fio 生成的每个读/写请求的块大小。红帽建议 4 KB 块大小来匹配 VDO 的 4 KB 默认	4k
--numjobs	fio 将为运行基准而创建的作业数量。 每个作业发送 **--size** 参数指定的数据量。 第一个作业将数据发送到 **--offset** 参数指定的偏移处的设备。除非提供了 **--offset_increment** 参数，否则后续作业写入磁盘的同一区域（覆盖）将使每个作业从上一个作业开始的位置偏移。要在闪存上达到峰值性能，至少有两个作业。一个作业通常足以饱和旋转型磁盘（HDD）吞吐量。	1 (HDD) 2 (SSD)
--thread	指示 fio 作业在线程中运行，而不是被分叉，这可以通过限制上下文切换来提供更好的性能。	<N/A>
--ioengine	Linux 中有几个可用的 I/O 引擎可以使用 fio 进行测试。红帽测试使用异步非缓冲引擎(**libaio**)。如果您对其他引擎感兴趣，请与红帽销售工程师进行讨论。 Linux **libaio** 引擎用于评估一个或多个进程同时进行随机请求的工作负载。**libaio** 在检索任何数据前，允许从单个线程中异步进行多个请求，这限制了在通过同步引擎提供给多个线程时所需的上下文切换数量。	libaio
--direct	设置后，直接允许将请求提交到设备，绕过 Linux 内核的页面缓存。 libaio Engine: **libaio** 必须与启用 **direct** (=1)一起使用，或者内核可能对所有 I/O 请求使用 **sync** API。	1 (libaio)
--iodepth	任意时间中的 I/O 缓冲区的数量。 高 **iodepth** 通常会提高性能，特别是用于随机读取或写入。高深度可确保控制器始终具有批处理请求。但是，设置 **iodepth** 太大（超过 1K，通常）可能会导致不必要的延迟。虽然红帽建议在 128 到 512 之间有一个 **iodepth**，但最终的值是一个利弊的，它取决于应用程序如何容忍延迟。	128 （最小）
--iodepth_batch_submit	当 **iodepth** 缓冲池开始为空时创建的 I/O 数量。此参数在测试过程中将任务从 I/O 限制为从 I/O 切换到缓冲区。	16
--iodepth_batch_complete	提交批处理前要完成的 I/O 数量 (**iodepth_batch_complete**)。此参数在测试过程中将任务从 I/O 限制为从 I/O 切换到缓冲区。	16
--gtod_reduce	禁用日常调用来计算延迟。如果启用，此设置将降低吞吐量，因此应启用(=1)，除非需要延迟测量。	1

31.3.1. 配置 VDO 测试卷

1.在 512 GB 物理卷中创建大小为 1 TB 的 VDO 卷

1. 创建 VDO 卷。

   - 要在同步存储之上测试 VDO async 模式，请使用 --writePolicy=async 选项创建一个异步卷：

     ```
     # vdo create --name=vdo0 --device=/dev/sdb \
             --vdoLogicalSize=1T --writePolicy=async --verbose
     ```

   - 要在 同步 存储之上测试 VDO 同步模式，请使用 --writePolicy=sync 选项创建一个同步卷：

     ```
     # vdo create --name=vdo0 --device=/dev/sdb \
             --vdoLogicalSize=1T --writePolicy=sync --verbose
     ```

2. 使用 XFS 或 ext4 文件系统格式化新设备。

   - 对于 XFS：

     ```
     # mkfs.xfs -K /dev/mapper/vdo0
     ```

   - 对于 ext4：

     ```
     # mkfs.ext4 -E nodiscard /dev/mapper/vdo0
     ```

3. 挂载格式化的设备：

   ```
   # mkdir /mnt/VDOVolume
   # mount /dev/mapper/vdo0 /mnt/VDOVolume && \
     chmod a+rwx /mnt/VDOVolume
   ```

31.3.2. 测试 VDO 效率

2.测试读取和写入到 VDO 卷

1. 将 32 GB 的随机数据写入 VDO 卷：

```
$ dd if=/dev/urandom of=/mnt/VDOVolume/testfile bs=4096 count=8388608
```

2. **从 VDO 卷中读取数据，并将其写入 VDO 卷中的另一个位置：**

```
$ dd if=/mnt/VDOVolume/testfile of=/home/user/testfile bs=4096
```

3. **使用 diff 比较两个文件，该文件应该报告这些文件是相同的文件：**

```
$ diff -s /mnt/VDOVolume/testfile /home/user/testfile
```

4. **将文件复制到 VDO 卷中的第二个位置：**

```
$ dd if=/home/user/testfile of=/mnt/VDOVolume/testfile2 bs=4096
```

5. **将第三个文件与第二个文件进行比较。这应该报告这些文件是相同的：**

```
$ diff -s /mnt/VDOVolume/testfile2 /home/user/testfile
```

3.删除 VDO 卷

1. **卸载在 VDO 卷中创建的文件系统：**

```
# umount /mnt/VDOVolume
```

2. **运行命令从系统中删除 VDO 卷 vdo0 ：**

```
# vdo remove --name=vdo0
```

3. **验证卷是否已移除。VDO 分区的 vdo 列表中 不应有列表：**

```
# vdo list --all | grep vdo
```

4.测量重复数据

1. **按照 第 31.3.1 节 “配置 VDO 测试卷” 创建并挂载 VDO 卷。**

2. 在名为 vdo1 到 vdo10 的 VDO 卷上创建 10 个目录，以存放测试数据集的 10 个副本：

```
$ mkdir /mnt/VDOVolume/vdo{01..10}
```

3. 根据文件系统检查所使用的磁盘空间量：

```
$ df -h /mnt/VDOVolume

Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vdo0      1.5T  198M  1.4T   1% /mnt/VDOVolume
```

考虑在表中模拟结果：

统计	裸机文件系统	seed 后	10 个副本后
文件系统使用的大小	198 MB		
已使用的 VDO 数据			
使用 VDO 逻辑			

4. 运行以下命令并记录值。"数据块使用"是 VDO 中运行的物理设备中用户数据所使用的块数。"使用的逻辑块"是优化前使用的块数。它将用作测量的起点

```
# vdostats --verbose | grep "blocks used"

data blocks used               : 1090
overhead blocks used           : 538846
logical blocks used            : 6059434
```

5. 在 VDO 卷顶层创建数据源文件

```
$ dd if=/dev/urandom of=/mnt/VDOVolume/sourcefile bs=4096 count=1048576

4294967296 bytes (4.3 GB) copied, 540.538 s, 7.9 MB/s
```

6. 重新检查已使用的物理磁盘空间量。这应该显示与刚才写入的文件对应的块数量的增加：

```
$ df -h /mnt/VDOVolume
```

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vdo0      1.5T  4.2G  1.4T   1% /mnt/VDOVolume
```

```
# vdostats --verbose | grep "blocks used"

data blocks used               : 1050093 (increased by 4GB)
overhead blocks used           : 538846 (Did not change)
logical blocks used            : 7108036 (increased by 4GB)
```

7. **将文件复制到 10 个子目录中的每个子目录中：**

```
$ for i in {01..10}; do
  cp /mnt/VDOVolume/sourcefile /mnt/VDOVolume/vdo$i
  done
```

8. **再次检查已使用的物理磁盘空间量（使用的数据块）。这个数字应该与上述步骤 6 的结果类似，因为文件系统日志记录和元数据，只需要稍微增加一些：**

```
$ df -h /mnt/VDOVolume

Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vdo0      1.5T   45G  1.3T   4% /mnt/VDOVolume
```

```
# vdostats --verbose | grep "blocks used"

data blocks used               : 1050836 (increased by 3M)
overhead blocks used           : 538846
logical blocks used            : 17594127 (increased by 41G)
```

9. **从写测试数据之前找到的值中减去文件系统所使用的空间的新值。从文件系统的角度来看，这是此测试消耗的空间量。**

10. **观察您记录统计中的空间节能：**

注意：在下表中，值已转换为 MB/GB。vdostats "blocks" 为 4,096 B。

统计	裸机文件系统	seed 后	10 个副本后
文件系统使用的大小	198 MB	4.2 GB	45 GB
已使用的 VDO 数据	4 MB	4.1 GB	4.1 GB

统计	裸机文件系统	seed 后	10 个副本后
使用 VDO 逻辑	23.6 GB*	27.8 GB	68.7 GB

** 1.6 TB 格式的驱动器的文件系统开销*

5.测量压缩

1. *创建一个至少 **10 GB** 物理和逻辑大小的 **VDO** 卷。添加选项来禁用 **deduplication** 并启用压缩：*

```
# vdo create --name=vdo0 --device=/dev/sdb \
        --vdoLogicalSize=10G --verbose \
        --deduplication=disabled --compression=enabled
```

2. *在传输前检查 **VDO** 统计；记录使用的数据块和逻辑块（两者均应为零）：*

```
# vdostats --verbose | grep "blocks used"
```

3. *使用 **XFS** 或 **ext4** 文件系统格式化新设备。*

   - *对于 **XFS**：*

   ```
   # mkfs.xfs -K /dev/mapper/vdo0
   ```

   - *对于 **ext4**：*

   ```
   # mkfs.ext4 -E nodiscard /dev/mapper/vdo0
   ```

4. *挂载格式化的设备：*

```
# mkdir /mnt/VDOVolume
# mount /dev/mapper/vdo0 /mnt/VDOVolume && \
  chmod a+rwx /mnt/VDOVolume
```

5. 同步 VDO 卷以完成所有未完成压缩：

```
# sync && dmsetup message vdo0 0 sync-dedupe
```

6. 再次检查 VDO 统计信息。logical blocks used - 使用的数据块数是文件系统压缩的 4 KB 块数量。VDO 优化文件系统开销以及实际的用户数据：

```
# vdostats --verbose | grep "blocks used"
```

7. 将 /lib 的内容复制到 VDO 卷中。记录总大小：

```
# cp -vR /lib /mnt/VDOVolume

...
sent 152508960 bytes  received 60448 bytes  61027763.20 bytes/sec
total size is 152293104  speedup is 1.00
```

8. 同步 Linux 缓存和 VDO 卷：

```
# sync && dmsetup message vdo0 0 sync-dedupe
```

9. 再次检查 VDO 统计。观察使用的逻辑和数据块：

```
# vdostats --verbose | grep "blocks used"
```

- logical blocks used - 使用的数据块代表 /lib 文件的副本所使用的空间量（以 4 KB 块为单位）。
- 总大小（来自 “4.测量重复数据”一节中的表）-（逻辑块 used-data 块使用 * 4096） = 压缩保存的字节数。

10. 删除 VDO 卷：

```
# umount /mnt/VDOVolume && vdo remove --name=vdo0
```

6.测试 VDO 压缩效率

1. 按照 第 31.3.1 节 “配置 VDO 测试卷” 创建并挂载 VDO 卷。

2. 在没有删除卷的情况下，在 “4.测量重复数据”一节 和 “5.测量压缩”一节 中重复实验。观察 vdostats 中空间节省的变化。

3. 使用您自己的数据集进行测试。

7.了解 TRIM 和 DISCARD

精简配置允许逻辑或虚拟存储空间大于底层存储。文件系统等应用程序从更大的存储的虚拟层上运行以及数据效率的技术（如数据重复数据删除）中受益，从而减少了存储所有数据所需的物理数据块数量。要从这些存储节省中受益，物理存储层需要知道何时删除应用程序数据。

传统的文件系统不必在删除数据时通知底层存储。使用精简配置的存储的文件系统发送 TRIM 或 DISCARD 命令，以便在不再需要逻辑块时通知存储系统。这些命令可以在使用 discard 挂载选项删除块时发送，或通过运行工具（如 fstrim ）以受控的方式发送这些命令，告知文件系统检测哪些逻辑块没有被使用，并以 TRIM 或 DISCARD 命令的形式向存储系统发送信息。

重要

有关精简配置如何工作的更多信息，请参阅 Red Hat Enterprise Linux 7 Logical Volume Manager Administration Guide 中的 Thinly-Provisioned Logical Volumes (Thin Volumes)。

查看它如何工作：

1. 按照 第 31.3.1 节 “配置 VDO 测试卷” 创建并挂载新的 VDO 逻辑卷。

2. 修剪文件系统以删除任何不需要的块（可能需要很长时间）：

```
# fstrim /mnt/VDOVolume
```

3. 输入以下内容记录下表中的初始状态：

```
$ df -m /mnt/VDOVolume
```

要查看文件系统中使用了多少容量，并运行 **vdostats** 来查看正在使用的物理和虚拟数据块的数量。

4. 在 VDO 上运行的文件系统中创建一个带有非重复数据的 1 GB 文件：

```
$ dd if=/dev/urandom of=/mnt/VDOVolume/file bs=1M count=1K
```

然后收集相同的数据。该文件系统应该已经使用额外的 **1 GB**，使用的数据块和逻辑块的相似增加。

5. 运行 **fstrim /mnt/VDOVolume** 并确认在创建新文件后这不会影响。

6. 删除 **1 GB** 文件：

```
$ rm /mnt/VDOVolume/file
```

检查并记录参数。文件系统知道文件已被删除，但没有更改物理或逻辑块的数量，因为文件删除尚未与底层存储通信。

7. 运行 **fstrim /mnt/VDOVolume** 并记录相同的参数。**fstrim** 在文件系统中查找空闲块，并为未使用的地址向 VDO 卷发送 **TRIM** 命令，这会释放相关的逻辑块，**VDO** 会处理 **TRIM** 来释放底层物理块。

步骤	已使用的文件空间 (MB)	已使用的数据块	使用的逻辑块
初始空间			
添加 1 GB 文件			
运行 **fstrim**			
删除 1 GB 文件			
运行 **fstrim**			

根据此练习，需要 TRIM 进程，以便底层存储可以准确了解容量利用率。fstrim 是一个命令行工具，可一次性分析多个块以提高效率。另一种方法是在挂载时使用文件系统丢弃选项。discard 选项会在每个文件系统块被删除后更新底层存储，这可能会降低吞吐量，但可为很好的利用率感知提供。另请务必了解 TRIM 或 DISCARD 未使用块对 VDO 并不独一无二；任何精简置备的存储系统都有相同的挑战

31.4. 性能测试过程

本节的目标是构建安装了 VDO 的设备的性能配置集。每个测试都应该使用且未安装 VDO 的情况下运行，以便根据基本系统性能评估 VDO 的性能。

31.4.1. 阶段 1：I/O Depth 的影响，修正了 4 KB 块

此测试的目标是确定为设备产生最佳吞吐量和最低延迟的 I/O 深度。VDO 使用 4 KB 扇区大小，而不是传统 512 B 在传统存储设备中使用。较大的扇区大小允许其支持更高容量存储、提高性能，并与大多数操作系统使用的缓存缓冲区大小匹配。

1. 执行 4 KB I/O 和 I/O 深度为 1, 8, 16, 32, 64, 128, 256, 512, 1024 的四级测试：

   - 顺序 100% 读取，在固定 4 KB * 中
   - 顺序 100% 写入，在固定 4 KB
   - 随机 100% 读取，在固定 4 KB * 中
   - 随机 100% 写入，在固定 4 KB **

   * 通过首先执行写 fio 作业，在读取测试过程中预先填充任何可能读取的区域

** 在 4 KB 随机写入 I/O 运行后重新创建 VDO 卷

shell 测试输入模拟器（写入）示例：

```
# for depth in 1 2 4 8 16 32 64 128 256 512 1024 2048; do
  fio --rw=write --bs=4096 --name=vdo --filename=/dev/mapper/vdo0 \
      --ioengine=libaio --numjobs=1 --thread --norandommap --runtime=300\
      --direct=1 --iodepth=$depth --scramble_buffers=1  --offset=0 \
      --size=100g
  done
```

2. 在各个数据点记录吞吐量和延迟，然后记录图形。

3. 重复测试以完成四级测试：--rw=randwrite、--rw=read 和 --rw=randread。

结果为图表，如下所示。感兴趣的点是跨范围的行为和 inflection 的点，其中增加了 I/O 深度，以降低吞吐量收益。顺序访问和随机访问将以不同值进行峰值，但适用于所有类型的存储配置。在 图 31.1 “I/O Depth Analysis” 中，注意每个性能曲线中的 "knee"。标记 1 标识 X 点的峰值顺序吞吐量，标记 2 标识 Z 点的峰值随机 4 KB 吞吐量。

- 这个特定设备不会从顺序 4 KB I/O depth > X 中受益。请注意，带宽带宽收益减少，平均请求延迟将增加每个额外 I/O 请求的 1:1。

- 这个特定设备不会从随机 4 KB I/O depth > Z 中获益。请注意，带宽收益会减少，平均请求延迟会增加每个额外 I/O 请求的 1:1。

图 31.1. I/O Depth Analysis

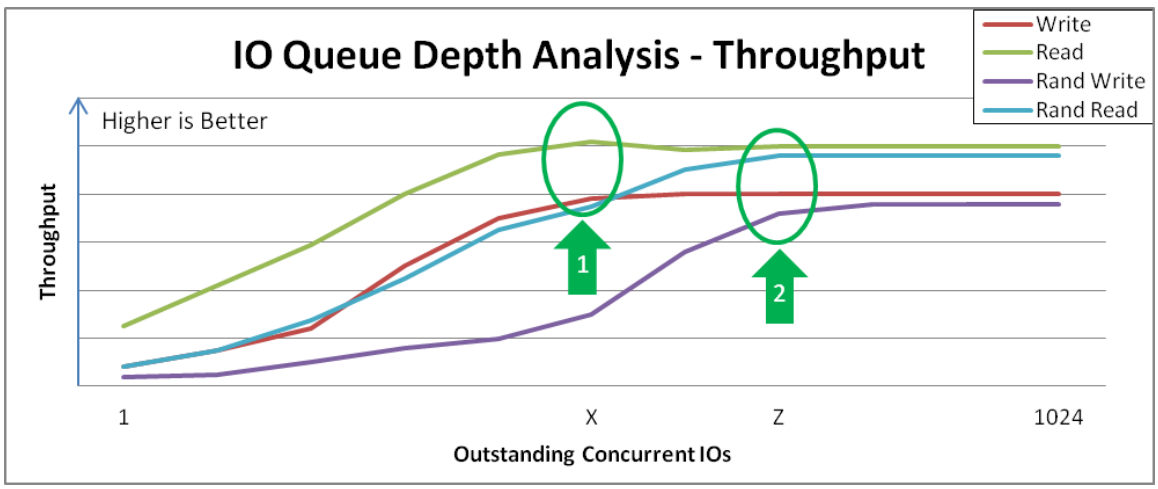
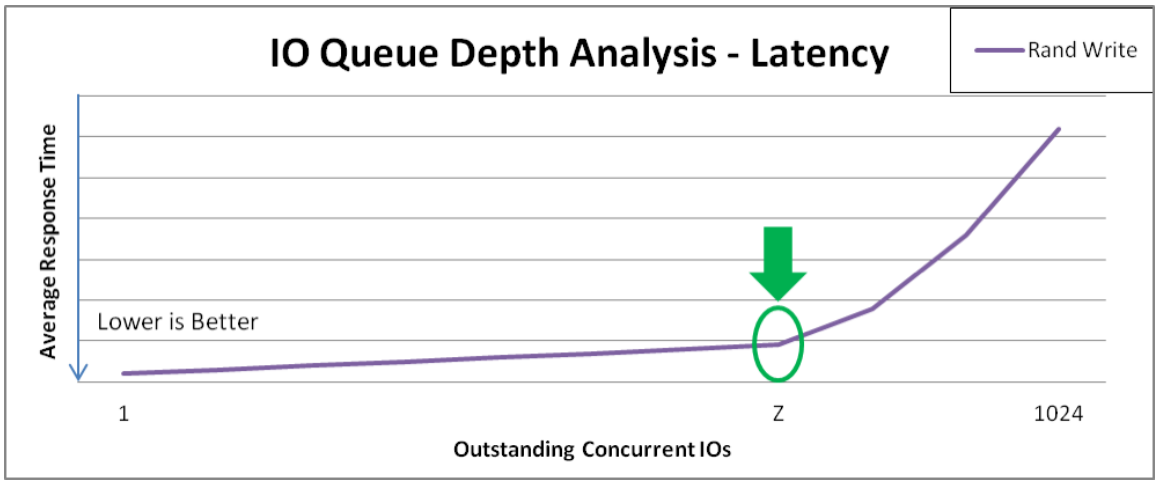


图 31.2 “为 Random Writes 增加 I/O 的延迟响应” 显示 图 31.1 “I/O Depth Analysis” 中 curve 的 "knee" 后随机写入延迟的示例。基准测试实践应在这些点上测试，以达到最低响应时间损失的最大吞吐量。当我们在这个示例设备的测试计划中转发时，我们将使用 I/O depth = Z 来收集其他数据

图 31.2. 为 Random Writes 增加 I/O 的延迟响应



31.4.2. 阶段 2： I/O 请求大小的影响

此测试的目标是了解在上一步中确定的最佳 I/O 深度下测试下系统性能的块大小。

1. *在固定 I/O 深度中执行四级测试，在范围 8 KB 到 1 MB 时，块大小有不同(2 的指数为 2)。请记住，要预先填充要读取的区域，并在测试之间重新创建卷。*

2. *将 I/O Depth 设置为 第 31.4.1 节 “阶段 1：I/O Depth 的影响，修正了 4 KB 块” 中决定的值。*

   测试输入模拟器（写入）示例：

   ```
   # z=[see previous step]
   # for iosize in 4 8 16 32 64 128 256 512 1024; do
     fio --rw=write --bs=$iosize\k --name=vdo --filename=/dev/mapper/vdo0
         --ioengine=libaio --numjobs=1 --thread --norandommap --runtime=300
         --direct=1 --iodepth=$z --scramble_buffers=1 --offset=0 --size=100g
     done
   ```

3. *在各个数据点记录吞吐量和延迟，然后记录图形。*

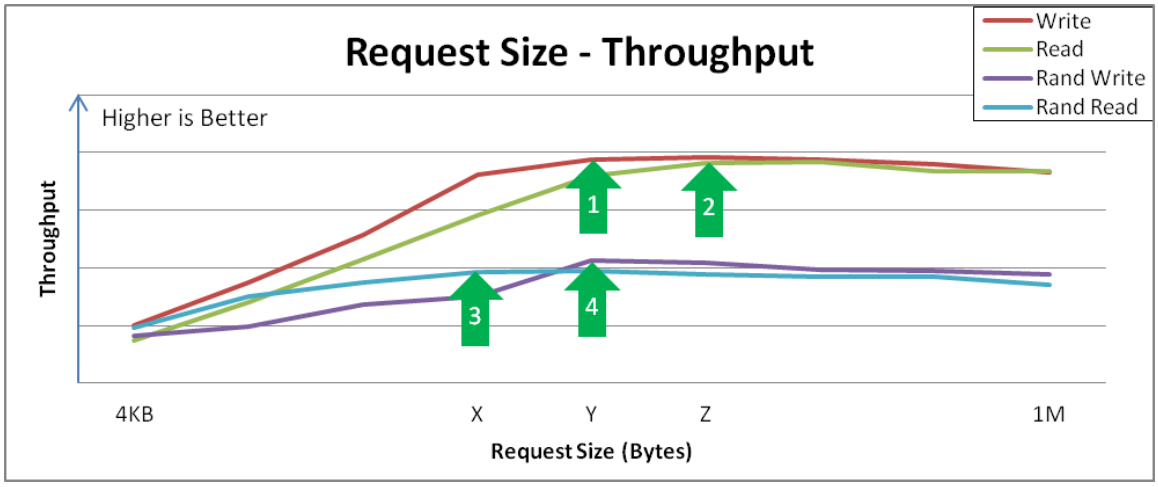
4. 重复测试以完成四级测试： --rw=randwrite、--rw=read 和 --rw=randread。

您可以发现结果有几点。在本例中：

- 顺序写入在请求大小 Y 时达到峰值吞吐量。此 curve 演示了可配置或自然地由特定请求大小划分的应用程序可能会感知性能。更大的请求大小通常会提供更多吞吐量，因为 4 KB I/O 可能会从合并中受益。
- 顺序读取在 Z 点达到类似的峰值吞吐量。请记住，在 I/O 完成后，I/O 完成后会增加一个类似的峰值吞吐量，而不会增加额外的吞吐量。将设备调优为不接受大于这个大小的 I/O 是明智的。
- 随机读取在 X 点达到峰值吞吐量。有些设备可能会在大型请求大小随机访问时达到近似的吞吐量率，而其他设备会在与纯顺序访问不同时受到更多的损失。
- 随机写入在 Y 点达到峰值吞吐量。随机写入涉及对去除重复设备的大多数交互，VDO 则实现高性能，特别是在请求大小和/或 I/O 深度较大时。

此测试 图 31.3 “请求大小与.吞吐量分析和密钥检查点” 的结果可帮助了解存储设备的特性以及特定应用程序的用户体验。向红帽销售工程师咨询，以确定是否有进一步调整，以在不同请求大小上提高性能。

图 31.3. 请求大小与.吞吐量分析和密钥检查点



31.4.3. 阶段 3：减少密集读和写 I/O 的影响

此测试的目的是了解您的带有 VDO 的设备在混合 I/O 加载时的行为方式（读/写），分析读/写混合在最佳随机队列深度和请求大小从 4 KB 到 1 MB。您应该根据您的具体情况使用任何合适的内容。

1. *在固定 I/O 深度中执行四级测试，在 8 KB 到 256 KB 的块大小(2 的指数)上进行四个测试，并以 10% 的增量设置读取百分比，从 0% 开始。请记住，要预先填充要读取的区域，并在测试之间重新创建卷。*

2. *将 I/O Depth 设置为 第 31.4.1 节 “阶段 1：I/O Depth 的影响，修正了 4 KB 块” 中决定的值。*

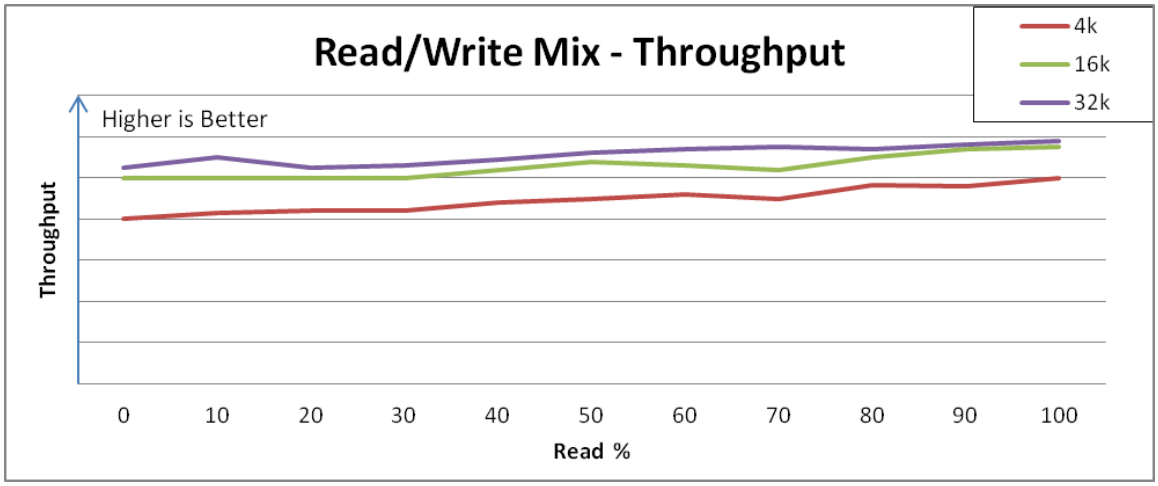
   测试输入模拟（读/写混合）示例：

```
# z=[see previous step]
# for readmix in 0 10 20 30 40 50 60 70 80 90 100; do
    for iosize in 4 8 16 32 64 128 256 512 1024; do
      fio --rw=rw --rwmixread=$readmix --bs=$iosize\k --name=vdo \
          --filename=/dev/mapper/vdo0 --ioengine=libaio --numjobs=1 --thread \
          --norandommap --runtime=300 --direct=0 --iodepth=$z --scramble_buffers=1 \
          --offset=0 --size=100g
    done
  done
```

3. *在各个数据点记录吞吐量和延迟，然后记录图形。*

图 31.4 “跨 Varying Read/Write Mixes 的性能下降” 显示 VDO 如何响应 I/O 负载的示例：

图 31.4. 跨 Varying Read/Write Mixes 的性能下降



性能(aggregate)和延迟(aggregate)在混合的读取和写入范围之间相对一致，从较低最大写入吞吐量到更高的最大读吞吐量。

这个行为可能因不同的存储而异，但重要的观察是，在不同负载和/或下性能是，您可以了解演示特定读/写混合的应用程序的性能期望。如果您发现任何意外的结果，红帽销售工程师将可以帮助您了解它是否是 VDO 还是需要修改的存储设备。

注意： 没有呈现类似的响应一致性的系统通常会表示一个子优化配置。如果出现这种情况，请联络您的红帽销售工程师。

31.4.4. 阶段 4：应用程序环境

这些最终测试的目的是了解在实际应用程序环境中部署时，使用 VDO 的系统的行为方式。如果可能，请使用实际应用程序并使用目前所学的知识；请考虑限制设备上允许的队列深度，如果可能，请调优应用以对这些块大小最有益于 VDO 性能的请求。

请求大小、I/O 负载、读/写模式等一般很难预测，因为它们会因应用程序用例（如文件与虚拟桌面与数据库）而异，并且应用程序通常因特定操作或多租户访问而不同。

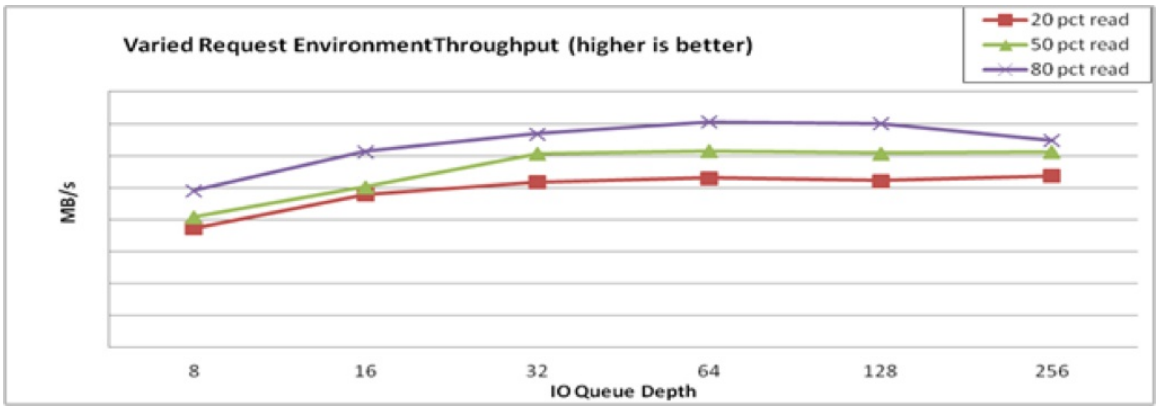
最终测试显示了混合环境中的通用 VDO 性能。如果对预期环境有更具体的信息，还要测试这些设置。

测试输入模拟（读/写混合）示例：

```
# for readmix in 20 50 80; do
    for iosize in 4 8 16 32 64 128 256 512 1024; do
      fio --rw=rw --rwmixread=$readmix --bsrange=4k-256k --name=vdo \
          --filename=/dev/mapper/vdo0 --ioengine=libaio --numjobs=1 --thread \
          --norandommap --runtime=300 --direct=0 --iodepth=$iosize \
          --scramble_buffers=1 --offset=0 --size=100g
    done
  done
```

在各个数据点记录吞吐量和延迟，然后记录图表(图 31.5 “混合环境性能”)。

图 31.5. 混合环境性能



31.5. 问题报告

如果在使用 VDO 时遇到问题，务必要收集尽可能多的信息，以帮助红帽销售工程师尝试重现问题。

问题报告应包含以下内容：

- 测试环境的详细描述；具体信息请查看 “测试环境”一节
- VDO 配置
- 生成此问题的用例
- 在错误时执行的操作
- 控制台或终端上任何错误消息的文本
- 内核日志文件
- 内核崩溃转储（如果可用）
- sosreport 的结果，这将捕获描述整个 Linux 环境的数据

31.6. 总结

通过此或任何其他精心设计的评估计划是将 VDO 集成到任何存储系统的一个重要步骤。评估过程对于了解性能并捕获潜在的兼容性问题非常重要。此评估的结果集合不仅演示重复数据删除和压缩，还提供实施 VDO 的系统的性能配置集。结果有助于确定实际应用程序中所实现的结果是否如预期、稳定或是否短预期。最后，我们还可使用这些结果来帮助预测使用 VDO 良好的应用程序类型。

附录 A. 红帽客户门户网站 LABS 与存储管理相关

红帽客户门户网站 Labs 是旨在帮助您提高性能、解决问题、识别安全问题并优化配置的工具。本附录概述了与存储管理相关的红帽客户门户网站 Labs。所有红帽客户门户网站 Labs 都可以通过 访问 https://access.redhat.com/labs/。

SCSI DECODER

SCSI decoder 被设计为解码 /log 8:0:1:: 文件或日志文件片断中的 SCSI 错误消息，因为这些错误消息可能很难了解用户。

使用 SCSI 解码器 单独诊断每个 SCSI 错误消息，并获取解决方案来有效地解决问题。

文件系统布局计算器

在提供描述当前或计划的存储选项后，文件系统布局计算器 决定创建 ext3、ext4 和 xfs 文件系统的最佳参数。将光标移到问号("?")中以获得特定选项的简短说明，或向下滚动以读取所有选项的摘要。

使用 文件系统布局计算器 生成一个命令，该命令在指定 RAID 存储上使用提供的参数创建文件系统。复制生成的命令，并以 root 用户身份执行它，以创建所需的文件系统。

LVM RAID CALCULATOR

LVM RAID Calculator 在指定了存储选项后，决定在给定 RAID 存储上创建逻辑卷(LVM)的最佳参数。将光标移到问号("?")中以获得特定选项的简短说明，或向下滚动以读取所有选项的摘要。

LVM RAID Calculator 生成在给定 RAID 存储上创建 LVM 的命令序列。以 root 用户身份复制并执行生成的命令，以创建所需的 LVM。

ISCSI HELPER

iSCSI Helper 通过互联网协议(IP)网络提供块存储，并允许在服务器虚拟化中使用存储池。

使用 iSCSI Helper 生成一个脚本，该脚本根据您提供的设置，为系统准备其 iSCSI 目标(server)或 iSCSI 启动器(client)。

SAMBA CONFIGURATION HELPER

Samba Configuration Helper 创建一个配置，它通过 Samba 提供基本文件和打印机共享：

- 点 Server 指定基本服务器设置。
- 单击 Shares 以添加您要共享的目录
- 单击 Server 以单独添加附加打印机。

多路径帮助器

多路径帮助器 为 Red Hat Enterprise Linux 5、6 和 7 上的多路径设备创建最佳配置。按照以下步骤，您可以创建高级多路径配置，如自定义别名或设备黑名单。

Multipath Helper 还提供 multipath.conf 文件以进行审阅。当您达到所需的配置时，请下载安装脚本以在服务器上运行。

NFS HELPER

NFS Helper 简化了配置新的 NFS 服务器或客户端。按照以下步骤指定 export 和 挂载选项。然后，生成可下载的 NFS 配置脚本。

多路径配置可视化工具

多路径配置可视化工具 分析 sosreport 中的文件，并提供视觉化多路径配置的图表。使用 多路径配置可视化工具 来显示：

- 主机组件，包括服务器端的主机总线适配器(HBA)、本地设备和 iSCSI 设备
- 存储端的存储组件
- 服务器和存储之间的光纤或以太网组件
- 所有上述组件的路径

您可以上传以 .xz、.gz 或 .bz2 格式压缩的 sosreport，或者在您选择作为客户端侧分析的源的目录中提取 sosreport。

RHEL BACKUP 和 RESTORE ASSISTANT

RHEL Backup 和 Restore Assistant 提供了有关备份和恢复工具的信息，以及 Linux 使用的常见场景。

描述的工具：

- *转储和恢复 ：用于备份 ext2、ext3 和 ext4 文件系统。*
- *tar 和 cpio ：用于归档或恢复文件和文件夹，特别是在备份磁带驱动器时。*
- *rsync ：执行备份操作并在位置之间同步文件和目录。*
- *dd: 通过独立于涉及的文件系统或操作系统，将文件从源复制到目标块。*

描述的场景：

- *灾难恢复*
- *硬件迁移*
- *分区表备份*
- *重要文件夹备份*
- *增量备份*

- *不同的备份*

附录 B. 修订历史记录

修订 4-10 异步更新	Mon Aug 10 2020	Marek Suchanek
修订 4-09 异步更新	Mon Jan 7 2019	Marek Suchanek
修订 4-08 为 7.6 GA 发布准备文档	Mon Oct 23 2018	Marek Suchanek
修订 4-07 异步更新	Thu Sep 13 2018	Marek Suchanek
修订 4-00 7.5 GA 出版物的文档版本。	Fri Apr 6 2018	Marek Suchanek
修订 3-95 异步更新	Thu Apr 5 2018	Marek Suchanek
修订 3-93 新章节：VDO 集成	Mon Mar 5 2018	Marek Suchanek
修订 3-92 异步更新	Fri Feb 9 2018	Marek Suchanek
修订 3-90 7.5 Alpha 出版物的版本。	Wed Dec 6 2017	Marek Suchanek
修订 3-86 异步更新。	Mon Nov 6 2017	Marek Suchanek
修订 3-80 7.4 GA 出版物的文件版本。	Thu Jul 27 2017	Milan Navratil
修订 3-77 异步更新。	Wed May 24 2017	Milan Navratil
修订 3-68 7.3 GA 出版物版本。	Fri Oct 21 2016	Milan Navratil
修订 3-67 异步更新。	Fri Jun 17 2016	Milan Navratil
修订 3-64 7.2 GA 版本。	Wed Nov 11 2015	Jana Heves
修订 3-33 7.1 GA 版本	Wed Feb 18 2015	Jacquelynn East
修订 3-26 添加了 Ceph 概述	Wed Jan 21 2015	Jacquelynn East
修订 3-22	Thu Dec 4 2014	Jacquelynn East

7.1 beta

修订 3-4 Thu Jul 17 2014 Jacquelynn East
在 targetcli 上添加了新的章节

修订 3-1 Tue Jun 3 2014 Jacquelynn East
7.0 GA 版本

索引

符号

存储访问参数

I/O 校准和大小，存储访问参数

安装存储配置

IBM System z 上的 DASD 和 zFCP 设备，IBM System Z 上的 DASD 和 zFCP 设备

iSCSI 检测和配置，iSCSI 检测和配置

使用 LUKS/dm-crypt 加密块设备，使用 LUKS 加密块设备

单独分区（用于 /home、/opt、/usr/local），/home、/opt、/usr/local 的单独分区

启用了 DIF/DIX 的块设备，启用了 DIF/DIX 的块设备

新功能，安装过程中的存储注意事项

更新，安装过程中的存储注意事项

过时的 BIOS RAID 元数据，过时的 BIOS RAID 元数据

通道命令字(CCW)，IBM System Z 上的 DASD 和 zFCP 设备

安装程序支持

RAID，Anaconda 安装程序中的 RAID 支持

安装过程中的存储注意事项

IBM System z 上的 DASD 和 zFCP 设备，IBM System Z 上的 DASD 和 zFCP 设备

iSCSI 检测和配置，iSCSI 检测和配置

updates，安装过程中的存储注意事项

使用 LUKS/dm-crypt 加密块设备，使用 LUKS 加密块设备

单独分区（用于 /home、/opt、/usr/local），/home、/opt、/usr/local 的单独分区

启用了 DIF/DIX 的块设备，启用了 DIF/DIX 的块设备

新功能，安装过程中的存储注意事项

过时的 BIOS RAID 元数据，过时的 BIOS RAID 元数据

通道命令字(CCW)，IBM System Z 上的 DASD 和 zFCP 设备

导出的文件系统

无盘系统，为无盘客户端配置导出的文件系统

将以太网接口配置为使用 FCoE

FcoE，通过以太网接口配置光纤通道

层次结构、文件系统，文件系统层次结构标准(FHS)概述.

工具（用于分区和其他文件系统功能）

I/O 校准和大小，分区和文件系统工具

xfs_bmap

XFS，其他 XFS 文件系统工具

xfs_copy

XFS，其他 XFS 文件系统工具

xfs_db

XFS，其他 XFS 文件系统工具

xfs_freeze

XFS，暂停 XFS 文件系统

xfs_fsr

XFS，其他 XFS 文件系统工具

xfs_growfs

XFS，增加 XFS 文件系统的大小

xfs_info

XFS，其他 XFS 文件系统工具

xfs_mdrestore

XFS，其他 XFS 文件系统工具

xfs_metadump

XFS，其他 XFS 文件系统工具

xfs_quota

XFS，XFS 配额管理

xfs_repair

XFS，修复 XFS 文件系统