# Solaris ZFS 管理指南

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

文件号码 819–7065–12
2008 年 4 月

080410@19860

# 目录

# 前言

《Solaris ZFS 管理指南》提供有关设置和管理 Solaris™ ZFS 文件系统的信息。

本指南中包含基于 SPARC® 和基于 x86 的系统的信息。

---

**注 –** 此 Solaris 发行版支持使用以下 SPARC 和 x86 系列处理器体系结构的系统：UltraSPARC®、SPARC64、AMD64、Pentium 和 Xeon EM64T。支持的系统可以在 http://www.sun.com/bigadmin/hcl 上的 Solaris OS: Hardware Compatibility Lists 中找到。本文档列举了在不同类型的平台上进行实现时的所有差别。

在本文档中，这些与 x86 相关的术语表示以下含义：

- "x86" 泛指 64 位和 32 位的 x86 兼容产品系列。
- "x64" 指出了有关 AMD64 或 EM64T 系统的特定 64 位信息。
- “32 位 x86” 指出了有关基于 x86 的系统的特定 32 位信息。

若想了解本发行版支持哪些系统，请参见 Solaris 10 硬件兼容性列表。

---

## 目标读者

本指南适用于对设置和管理 Solaris ZFS 文件系统感兴趣的任何用户。最好具有使用 Solaris 操作系统 (Operating System, OS) 或其他 UNIX® 版本的经验。

## 本书的结构

下表介绍了本书中的各章。

| 章 | 说明 |
| --- | --- |
| 第 1 章 | 概述 ZFS 及其功能和优点。本章还介绍了一些基本概念和术语。 |
| 第 2 章 | 提供通过简单池和文件系统设置简单 ZFS 配置的逐步说明。本章还介绍了创建 ZFS 文件系统所需的硬件和软件。 |

| 章 | 说明 |
| --- | --- |
| 第 3 章 | 确定使 ZFS 显著区别于传统文件系统的重要功能。了解这些关键差异有助于在使用传统工具与 ZFS 交互时避免混淆。 |
| 第 4 章 | 提供有关如何创建和管理存储池的详细说明。 |
| 第 5 章 | 提供有关管理 ZFS 文件系统的详细信息，其中包括分层文件系统布局、属性继承以及自动挂载点管理和共享交互等概念。 |
| 第 6 章 | 介绍如何创建和管理 ZFS 快照和克隆。 |
| 第 7 章 | 介绍如何使用访问控制列表 (access control list, ACL) 通过提供比标准 UNIX 权限更详尽的权限来保护 ZFS 文件。 |
| 第 8 章 | 提供有关使用 ZFS 卷、在安装了区域的 Solaris 系统中使用 ZFS 以及备用根池的信息。 |
| 第 9 章 | 介绍如何确定 ZFS 故障模式以及如何从中进行恢复。本章还介绍了防止故障的步骤。 |

# 相关书籍

以下书籍提供了有关常规 Solaris 系统管理主题的相关信息：

- Solaris《系统管理指南：基本管理》
- Solaris《系统管理指南：高级管理》
- Solaris《系统管理指南：设备和文件系统》
- Solaris《系统管理指南：安全性服务》
- 《Solaris Volume Manager 管理指南》

# 文档、支持和培训

Sun Web 站点提供有关以下附加资源的信息：

- 文档 (http://www.sun.com/documentation/)
- 支持 (http://www.sun.com/support/)
- 培训 (http://www.sun.com/training/)

# 印刷约定

下表介绍了本书中的印刷约定。

**表 P–1** 印刷约定

| 字体或符号 | 含义 | 示例 |
| --- | --- | --- |
| `AaBbCc123` | 命令、文件和目录的名称；计算机屏幕输出 | 编辑 `.login` 文件。<br>使用 `ls -a` 列出所有文件。<br>`machine_name% you have mail.` |
| **`AaBbCc123`** | 用户键入的内容，与计算机屏幕输出的显示不同 | `machine_name%` **`su`**<br>`Password:` |
| *aabbcc123* | 要使用实名或值替换的命令行占位符 | 删除文件的命令为 `rm` *filename*。 |
| *AaBbCc123* | 保留未译的新词或术语以及要强调的词 | 这些称为 *Class* 选项。<br>**注意**：有些强调的项目在联机时以粗体显示。 |
| **新词术语强调** | 新词或术语以及要强调的词 | **高速缓存**是存储在本地的副本。<br>请**勿**保存文件。 |
| 《书名》 | 书名 | 阅读《用户指南》的第 6 章。 |

# 命令中的 shell 提示符示例

下表列出了 C shell、Bourne shell 和 Korn shell 的缺省 UNIX 系统提示符和超级用户提示符。

**表 P–2** shell 提示符

| shell | 提示符 |
| --- | --- |
| C shell 提示符 | `machine_name%` |
| C shell 超级用户提示符 | `machine_name#` |
| Bourne shell 和 Korn shell 提示符 | `$` |
| Bourne shell 和 Korn shell 超级用户提示符 | `#` |

◆ ◆ ◆ 第 1 章

# Solaris ZFS 文件系统（介绍）

本章概述了 Solaris™ ZFS 文件系统及其功能和优点。本章还介绍了在本书所有其余部分中使用的一些基本术语。

本章包含以下各节：



## ZFS 中的新增功能

本节概述了 ZFS 文件系统的新增功能。

## 改进的 zpool status 输出

**Solaris 10 8/07 发行版**：使用 zpool status -v 命令可显示包含持久性错误的文件的列表。以前，必须使用 find -inum 命令从显示的 inode 列表中标识文件名。

有关显示包含持久性错误的文件列表的更多信息，请参见第 155 页中的 “修复损坏的文件或目录”。

## ZFS 和 Solaris iSCSI 改进

**Solaris 10 8/07 发行版**：在此 Solaris 发行版中，可以通过对 ZFS 卷设置 shareiscsi 属性将 ZFS 卷创建为 Solaris iSCSI 目标设备。此方法是快速设置 Solaris iSCSI 目标的便捷途径。例如：

```
# zfs create -V 2g tank/volumes/v2
# zfs set shareiscsi=on tank/volumes/v2
# iscsitadm list target
Target: tank/volumes/v2
    iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
    Connections: 0
```

创建 iSCSI 目标后，应设置 iSCSI 启动器。有关设置 Solaris iSCSI 启动器的信息，请参见《系统管理指南：设备和文件系统》中的第 15 章 “配置 Solaris iSCSI 启动器（任务）”。

有关将 ZFS 卷作为 iSCSI 目标来管理的更多信息，请参见第 132 页中的 “使用 ZFS 卷作为 Solaris iSCSI 目标”。

## ZFS 命令历史记录 (zpool history)

**Solaris 10 8/07 发行版**：在此 Solaris 发行版中，ZFS 会自动记录成功修改池状态信息的 zfs 和 zpool 命令。例如：

```
# zpool history
History for 'newpool':
2007-04-25.11:37:31 zpool create newpool mirror c0t8d0 c0t10d0
2007-04-25.11:37:46 zpool replace newpool c0t10d0 c0t9d0
2007-04-25.11:38:04 zpool attach newpool c0t9d0 c0t11d0
2007-04-25.11:38:09 zfs create newpool/user1
2007-04-25.11:38:15 zfs destroy newpool/user1
```

```
History for 'tank':
2007-04-25.11:46:28 zpool create tank mirror c1t0d0 c2t0d0 mirror c3t0d0 c4t0d0
```

借助此功能，用户或 Sun 技术支持人员可以**精确**确定已执行的 ZFS 命令集，来排除错误情况。

可以使用 `zpool history` 命令标识特定存储池。例如：

```
# zpool history newpool
History for 'newpool':
History for 'newpool':
2007-04-25.11:37:31 zpool create newpool mirror c0t8d0 c0t10d0
2007-04-25.11:37:46 zpool replace newpool c0t10d0 c0t9d0
2007-04-25.11:38:04 zpool attach newpool c0t9d0 c0t11d0
2007-04-25.11:38:09 zfs create newpool/user1
2007-04-25.11:38:15 zfs destroy newpool/user1
```

历史记录日志有如下特点：

- 不能禁用日志。
- 日志持久保存在磁盘上，这意味着系统重新引导后，将保存日志。
- 日志作为环形缓冲区来实现。最小大小为 128 KB。最大大小为 32 MB。
- 对于较小的池，日志最大大小限定在池大小的 1%，其中池大小在池创建时确定。
- 无需任何管理，这意味着不需要调整日志大小或更改日志位置。

目前，`zpool history` 命令不记录 *user-ID*、*hostname* 或 *zone-name*。

有关 ZFS 问题故障排除的更多信息，请参见第 142 页中的 “确定 ZFS 中的问题” 。

# ZFS 属性改进

## ZFS `xattr` 属性

**Solaris 10 8/07 发行版**：可以使用 `xattr` 属性为特定的 ZFS 文件系统禁用或启用扩展属性。缺省值为 on。有关 ZFS 属性的说明，请参见第 74 页中的 “ZFS 属性介绍” 。

## ZFS `canmount` 属性

**Solaris 10 8/07 发行版**：借助新的 `canmount` 属性，能够指定是否可以使用 `zfs mount` 命令挂载数据集。有关更多信息，请参见第 80 页中的 “`canmount` 属性” 。

## ZFS 用户属性

**Solaris 10 8/07 发行版**：除了可以导出内部统计信息或控制 ZFS 文件系统行为的标准本机属性外，ZFS 还支持用户属性。用户属性对 ZFS 行为没有影响，但可通过用户环境中有意义的信息来注释数据集。

有关更多信息，请参见第 81 页中的 “ZFS 用户属性” 。

### 在创建 ZFS 文件系统时设置属性

**Solaris 10 8/07 发行版**：在此 Solaris 发行版中，不但可以在创建文件系统后设置属性，还可以在创建文件系统时设置属性。

以下示例演示了等效的语法：

```
# zfs create tank/home
# zfs set mountpoint=/export/zfs tank/home
# zfs set sharenfs=on tank/home
# zfs set compression=on tank/home

# zfs create -o mountpoint=/export/zfs -o sharenfs=on -o compression=on tank/home
```

## 显示所有 ZFS 文件系统信息

**Solaris 10 8/07 发行版**：在此 Solaris 发行版中，可以使用各种形式的 `zfs get` 命令来显示有关所有数据集的信息（如果未指定数据集）。在早期发行版中，使用 `zfs get` 命令无法获取所有数据集信息。

例如：

```
# zfs get -s local all
tank/home               atime          off                    local
tank/home/bonwick       atime          off                    local
tank/home/marks         quota          50G                    local
```

## 新 `zfs receive -F` 选项

**Solaris 10 8/07 发行版**：在此 Solaris 发行版中，可以在 `zfs receive` 命令中使用新的 `-F` 选项，强制文件系统回滚到执行接收之前的最新快照。如果在发生回滚和启动接收之间修改了文件系统，可能需要使用此选项。

有关更多信息，请参见第 105 页中的 “恢复 ZFS 快照” 。

## 递归 ZFS 快照

**Solaris 10 11/06 发行版**：使用 `zfs snapshot` 命令创建文件系统快照时，可以使用 `-r` 选项为所有后代文件系统递归创建快照。此外，使用 `-r` 选项还可以在销毁快照时递归销毁所有后代快照。

递归 ZFS 快照可作为一个原子操作快速创建。要么一起创建快照（一次创建所有快照），要么不创建任何快照。原子快照操作的优点是始终在一个一致的时间捕获快照数据，即使跨后代文件系统也是如此。

有关更多信息，请参见第 98 页中的 “创建和销毁 ZFS 快照”。

## 双奇偶校验 RAID-Z (raidz2)

**Solaris 10 11/06 发行版**：现在，冗余 RAID-Z 配置可以具有单奇偶校验或双奇偶校验，这意味着可以分别承受一个或两个设备故障，而不会丢失任何数据。可以为双奇偶校验 RAID-Z 配置指定 `raidz2` 关键字。还可以为单奇偶校验 RAID-Z 配置指定 `raidz` 或 `raidz1` 关键字。

有关更多信息，请参见第 41 页中的 “创建 RAID-Z 存储池”或 `zpool`(1M)。

## ZFS 存储池设备的热备件

**Solaris 10 11/06 发行版**：借助 ZFS 热备件功能，可以在一个或多个存储池中确定可用来替换发生故障或出现错误的设备的磁盘。指定一个设备作为**热备件**，意味着如果池中的某一活动设备发生故障，热备件将自动替换该故障设备。或者，也可以用热备件手动替换存储池中的设备。

有关更多信息，请参见第 52 页中的 “在存储池中指定热备件”和 `zpool`(1M)。

## 使用 ZFS 克隆替换 ZFS 文件系统 (zfs promote)

**Solaris 10 11/06 发行版**：借助 `zfs promote` 命令，可以使用现有 ZFS 文件系统的克隆来替换该文件系统。当您要在备用版本的文件系统上运行测试而后使其成为活动文件系统时，此功能将很有帮助。

有关更多信息，请参见第 102 页中的 “使用 ZFS 克隆替换 ZFS 文件系统”和 `zfs`(1M)。

## 升级 ZFS 存储池 (zpool upgrade)

**Solaris 10 6/06 发行版**：通过使用 `zpool upgrade` 命令，可以将存储池升级到更新的版本，以利用最新功能。此外，`zpool status` 命令已经修改，可在池运行较早的版本时发出通知。

有关更多信息，请参见第 68 页中的 “升级 ZFS 存储池”和 `zpool`(1M)。

如果要在包含来自以前 Solaris 发行版的池的系统上使用 ZFS 管理控制台，请确保在使用 ZFS 管理控制台之前先升级池。要查看池是否需要升级，请使用 `zpool status` 命令。有关 ZFS 管理控制台的信息，请参见第 20 页中的 “基于 Web 的 ZFS 管理”。

# 使用 ZFS 克隆非全局区域以及其他增强功能

**Solaris 10 6/06 发行版**：当源 zonepath 和目标 zonepath 都驻留在 ZFS 上并且位于同一个池中时，zoneadm clone 现在可以自动使用 ZFS 克隆功能来克隆区域。此增强功能意味着 zoneadm clone 将实施源 zonepath 的 ZFS 快照并设置目标 zonepath。快照命名为 SUNWzoneX，其中 X 是用来区分多个快照的唯一 ID。目标区域的 zonepath 用来指定 ZFS 克隆。将执行软件清点，以使系统可对将来使用的快照进行验证。请注意，如果需要，仍可以指定复制 ZFS zonepath 而非 ZFS 克隆。

要多次克隆源区域，应向 zoneadm 中添加一个新参数，以指定应使用现有快照。系统将验证现有快照在目标中是否可用。此外，现在区域安装过程能够检测何时可为区域创建 ZFS 文件系统，卸载过程能够检测何时可以销毁区域中的 ZFS 文件系统。然后，zoneadm 命令将自动执行这些步骤。

在安装了 Solaris 容器的系统上使用 ZFS 时，请切记以下几点：

- 请勿使用 ZFS 快照功能来克隆区域
- 可以将 ZFS 文件系统委托给非全局区域或添加到非全局区域。有关更多信息，请参见第 134 页中的 “向非全局区域中添加 ZFS 文件系统”或第 134 页中的 “将数据集委托给非全局区域”。
- 在 Solaris 10 发行版中，请勿将 ZFS 文件系统用作全局区域根路径或非全局区域根路径。在 Solaris Express 发行版中，可以将 ZFS 用作区域根路径，但请记住，不支持对这些区域进行修补或升级。

有关更多信息，请参见《系统管理指南：Solaris Containers－资源管理和 Solaris Zones》。

# ZFS 备份和恢复命令已重命名

**Solaris 10 6/06 发行版**：在此 Solaris 发行版中，zfs backup 和 zfs restore 命令已分别重命名为 zfs send 和 zfs receive，以便更准确地描述其功能。这些命令的功能是保存和恢复 ZFS 数据流表示。

有关这些命令的更多信息，请参见第 103 页中的 “保存和恢复 ZFS 数据”。

# 恢复已销毁的存储池

**Solaris 10 6/06 发行版**：此发行版中包括 zpool import -D 命令，通过该命令可以恢复以前使用 zpool destroy 命令销毁的池。

有关更多信息，请参见第 66 页中的 “恢复已销毁的 ZFS 存储池”。

## 集成 ZFS 与 Fault Manager

**Solaris 10 6/06 发行版**：此发行版集成了 ZFS 诊断引擎，该诊断引擎可诊断和报告池故障和设备故障。另外，还可与池或设备的故障关联的校验和 I/O 设备和池错误。

该诊断引擎不包括校验和以及 I/O 错误的预测性分析，也不包括基于故障分析的主动操作。

如果出现 ZFS 故障，则可能显示以下类似来自 `fmd` 的消息：

```
SUNW-MSG-ID: ZFS-8000-D3, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Fri Mar 10 11:09:06 MST 2006
PLATFORM: SUNW,Ultra-60, CSN: -, HOSTNAME: neo
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: b55ee13b-cd74-4dff-8aff-ad575c372ef8
DESC: A ZFS device failed.  Refer to http://sun.com/msg/ZFS-8000-D3 for more information.
AUTO-RESPONSE: No automated response will occur.
IMPACT: Fault tolerance of the pool may be compromised.
REC-ACTION: Run 'zpool status -x' and replace the bad device.
```

通过查看建议的操作（位于 `zpool status` 命令中的具体指令之后），可快速确定和解决故障问题。

有关从所报告的 ZFS 问题中恢复的示例，请参见第 147 页中的 “修复缺少的设备”。

## 新增 `zpool clear` 命令

**Solaris 10 6/06 发行版**：此发行版包括 `zpool clear` 命令，该命令用于清除与设备或池关联的错误计数。以前，错误计数是在使用 `zpool online` 命令使池中的设备联机时清除的。有关更多信息，请参见 `zpool`(1M) 和第 50 页中的 “清除存储池设备”。

## 紧凑 NFSv4 ACL 格式

**Solaris 10 6/06 发行版**：在此发行版中，有三种 NFSv4 ACL 格式可用：详细格式、位置格式和紧凑格式。新增的紧凑和位置 ACL 格式可用于设置和显示 ACL。可以使用 `chmod` 命令设置所有 3 种 ACL 格式。可以使用 `ls -V` 命令显示紧凑 ACL 格式和位置 ACL 格式，使用 `ls -v` 命令显示详细 ACL 格式。

有关更多信息，请参见第 127 页中的 “以缩写格式设置和显示 ZFS 文件的 ACL”、`chmod`(1) 和 `ls`(1)。

# 文件系统监视工具 (fsstat)

**Solaris 10 6/06 发行版：** fsstat 是一个新的文件系统监视工具，可用于报告文件系统操作。可按挂载点或文件系统类型来报告活动。以下示例显示常规的 ZFS 文件系统活动。

```
$ fsstat zfs
 new  name   name  attr  attr lookup rddir  read read  write write
 file remov  chng   get   set    ops   ops   ops bytes   ops bytes
7.82M 5.92M 2.76M 1.02G 3.32M  5.60G 87.0M  363M 1.86T 20.9M  251G zfs
```

有关更多信息，请参见 fsstat(1M)。

# 基于 Web 的 ZFS 管理

**Solaris 10 6/06 发行版：** 可以使用一种基于 Web 的 ZFS 管理工具来执行许多管理操作。通过此工具，可以执行以下任务：

- 创建新存储池。
- 为现有池添加功能。
- 将存储池移动（导出）到另一个系统。
- 导入以前导出的存储池，使其可在另一个系统中使用。
- 查看有关存储池的信息。
- 创建文件系统。
- 创建卷。
- 捕获文件系统或卷的快照。
- 将文件系统回滚到以前的快照。

通过安全 Web 浏览器访问以下 URL，可以访问 ZFS 管理控制台：

```
https://system-name:6789/zfs
```

如果键入了适当的 URL 但无法访问 ZFS 管理控制台，则表明可能未启动服务器。要启动服务器，请运行以下命令：

```
# /usr/sbin/smcwebserver start
```

如果希望服务器在系统引导时自动启动，请运行以下命令：

```
# /usr/sbin/smcwebserver enable
```

---

**注 –** 不能使用 Solaris Management Console (smc) 管理 ZFS 存储池或文件系统。

---

# 什么是 ZFS？

ZFS 文件系统是一种革新性的新文件系统，可从根本上改变文件系统的管理方式，并具有目前面市的其他任何文件系统所没有的功能和优点。根据设计，ZFS 具有稳定、可伸缩性和便于管理等优点。

## ZFS 池存储

ZFS 使用**存储池**的概念来管理物理存储。以前，文件系统是在单个物理设备的基础上构造的。为了利用多个设备和提供数据冗余性，引入了**卷管理器**的概念来提供单个设备的映像，以便无需修改文件系统即可利用多个设备。此设计增加了更多复杂性，并最终阻碍了特定文件系统的继续发展，因为这类文件系统无法控制数据在虚拟卷上的物理放置。

ZFS 可完全避免使用卷管理。ZFS 将设备聚集到存储池中，而不是强制要求创建虚拟卷。存储池说明了存储的物理特征（设备布局、数据冗余等），并充当可以从其创建文件系统的任意数据存储库。文件系统不再仅限于单个设备，从而可与池中的所有文件系统共享空间。您不再需要预先确定文件系统的大小，因为文件系统会在分配给存储池的空间内自动增长。添加新存储器后，无需执行其他操作，池中的所有文件系统即可立即使用所增加的空间。在许多方面，存储池都类似于虚拟内存系统。内存 DIMM 添加到系统后，操作系统并不强制您调用某些命令来配置该内存并将其指定给单个进程。系统中的所有进程都会自动使用所增加的内存。

## 事务性语义

ZFS 是事务性文件系统，这意味着文件系统状态在磁盘上始终是一致的。传统文件系统可就地覆写数据，这意味着如果计算机断电（例如，在分配数据块到将其链接到目录中的时间段内断电），则会使文件系统处于不一致状态。以前，此问题是通过使用 `fsck` 命令解决的。此命令负责检查和验证文件系统状态，尝试修复该过程中的任何不一致性问题。此问题使管理员非常苦恼，并且从来无法保证解决所有可能的问题。最近，文件系统引入了**日志记录**的概念。日志记录过程在单独的日志中记录操作，然后在出现系统崩溃时可以安全地重放该日志。由于数据需要写入两次，因此此过程会引入不必要的开销，并通常导致一组新问题，如无法正确地重放日志时。

对于事务性文件系统，数据是使用**写复制**语义管理的。数据永远不会被覆写，并且任何操作序列会全部被提交或全部被忽略。此机制意味着文件系统绝对不会因意外断电或系统崩溃而被损坏。因此，无需存在 `fsck` 等效项。尽管最近写入的数据片段可能丢失，但是文件系统本身将始终是一致的。此外，只有在写入同步数据（使用 `O_DSYNC` 标志写入）后才返回，因此同步数据决不会丢失。

# 校验和与自我修复数据

对于 ZFS，所有数据和元数据都通过使用用户可选择的算法来执行校验和操作。提供校验和操作的传统文件系统出于卷管理层和传统文件系统设计的必要，会逐块执行此操作。传统设计意味着某些故障模式（如将完整块写入不正确的位置）可能会生成校验和正确的数据，而该数据实际上并不正确。ZFS 校验和的存储方式可确保检测到这些故障模式并可以正常地从其中进行恢复。所有校验和操作与数据恢复都是在文件系统层执行的，并且对应用程序是透明的。

此外，ZFS 还会提供自我修复数据。ZFS 支持具有不同数据冗余级别的存储池，包括镜像和 RAID-5 变化形式。检测到坏的数据块时，ZFS 从另一个冗余的副本中提取正确数据，并将错误数据替换为正确副本以对其进行修复。

# 独一无二的可伸缩性

ZFS 经过了全新设计，是目前为止可伸缩性最高的文件系统。该文件系统本身是 128 位的，所允许的存储空间是 256 quadrillion zettabyte ($256x10^{15}$ ZB) 的存储。所有元数据都是动态分配的，因此在首次创建时无需预先分配 inode，否则就会限制文件系统的可伸缩性。所有算法在编写时都考虑到了可伸缩性。目录最多可以包含 $2^{48}$（256 万亿）项，并且对于文件系统数或文件系统中可以包含的文件数不存在限制。

# ZFS 快照

**快照**是文件系统或卷的只读副本。可以快速而轻松地创建快照。最初，快照不会占用池中的任何附加空间。

活动数据集中的数据更改时，快照通过继续引用旧数据来占用空间。因此，快照可防止将数据释放回池中。

# 简化的管理

最重要的是，ZFS 提供了一种极度简化的管理模型。通过使用分层文件系统布局、属性继承以及自动管理挂载点和 NFS 共享语义，ZFS 可轻松创建和管理文件系统，而无需使用多个命令或编辑配置文件。可以轻松设置配额或预留空间，启用或禁用压缩，或者通过单个命令管理许多文件系统的挂载点。可以检查或修复设备，而不必了解一组单独的卷管理器命令。可以捕获文件系统的无数即时快照。可以备份和恢复单个文件系统。

ZFS 通过分层结构管理文件系统，该分层结构允许对属性（如配额、预留空间、压缩和挂载点）进行这一简化管理。在此模型中，文件系统会成为中央控制点。文件系统本身的开销非常小（相当于新目录），因此鼓励您为每个用户、项目、工作区等创建一个文件系统。通过此设计，可定义细分的管理点。

# ZFS 术语

本节介绍了在本书中使用的基本术语：

校验和　文件系统块中数据的 256 位散列。校验和功能的范围可以从简单快速的 fletcher2（缺省值）到强加密散列（如 SHA256）。

克隆　其初始内容与快照内容相同的文件系统。

有关克隆的信息，请参见第 101 页中的 “ZFS 克隆概述”。

数据集　以下 ZFS 实体的通用名称：克隆、文件系统、快照或卷。

每个数据集由 ZFS 名称空间中的唯一名称标识。数据集使用以下格式进行标识：

*pool*/*path*[ @*snapshot*]

*pool*　标识包含数据集的存储池的名称

*path*　数据集对象的斜杠分隔路径名

*snapshot*　用于标识数据集快照的可选组件

有关数据集的更多信息，请参见第 5 章。

文件系统　包含标准 POSIX 文件系统的数据集。

有关文件系统的更多信息，请参见第 5 章。

镜像　在两个或更多磁盘上存储相同数据副本的虚拟设备。如果镜像中的任一磁盘出现故障，则该镜像中的其他任何磁盘都可以提供相同的数据。

池　设备的逻辑组，用于说明可用存储的布局和物理特征。数据集的空间是从池中分配的。

有关存储池的更多信息，请参见第 4 章。

RAID-Z　在多个磁盘上存储数据和奇偶校验的虚拟设备，与 RAID-5 类似。有关 RAID-Z 的更多信息，请参见第 38 页中的 “RAID-Z 存储池配置”。

重新同步　将数据从一个设备传输到另一个设备的过程称为**重新同步**。例如，如果替换了镜像组件或使其脱机，则最新镜像组件中的数据会复制到刚恢复的镜像组件。此过程在传统的卷管理产品中称为**镜像重新同步**。

有关 ZFS 重新同步的更多信息，请参见第 152 页中的 “查看重新同步状态”。

快照　文件系统或卷在指定时间点的只读映像。

有关快照的更多信息，请参见第 97 页中的 “ZFS 快照概述”。

虚拟设备　池中的逻辑设备，可以是物理设备、文件或设备集合。

有关虚拟设备的更多信息，请参见第 37 页中的 “标识存储池中的虚拟设备” 。

卷　用于模仿物理设备的数据集。例如，可以创建 ZFS 卷作为交换设备。

有关 ZFS 卷的更多信息，请参见第 131 页中的 “ZFS 卷” 。

# ZFS 组件命名要求

每个 ZFS 组件必须根据以下规则进行命名：

- 不允许空组件。
- 每个组件只能包含字母数字字符以及以下四个特殊字符：
  - 下划线 (_)
  - 连字符 (-)
  - 冒号 (:)
  - 句点 (.)
- 池名称必须以字母开头，但以下限制除外：
  - 不允许使用起始序列 `c[0-9]`。
  - 名称 `log` 为保留名称。
  - 不允许使用以 `mirror`、`raidz` 或 `spare` 开头的名称，因为这些名称是保留名称。

  此外，池名称不得包含百分比符号 (`%`)。
- 数据集名称必须以字母数字字符开头。数据集名称不得包含百分比符号 (`%`)。

◆ ◆ ◆ 第 2 章

# ZFS 入门

本章提供了有关设置简单 ZFS 配置的逐步说明。学完本章之后，您应基本了解 ZFS 命令的工作原理，并可以创建简单的池和文件系统。本章是综合概述，有关更多详细信息，请参阅后续章节。

本章包含以下各节：



## ZFS 硬件和软件要求及建议

尝试使用 ZFS 软件之前，请确保查看了以下硬件和软件要求及建议：

- 运行 Solaris 10 6/06 发行版或更高版本的 SPARC™ 或 x86 系统。
- 最小磁盘空间为 128 MB。用于存储池所需的最小磁盘空间量约为 64 MB。
- 目前，建议用于安装 Solaris 系统的最小内存量为 512 MB。但为了获得更好的 ZFS 性能，建议至少使用 1 GB 或更多内存。
- 如果创建镜像磁盘配置，建议使用多个控制器。

## 创建基本 ZFS 文件系统

ZFS 管理在设计过程中考虑了简单性。ZFS 设计的目标之一是减少创建可用文件系统所需的命令数。创建新池的同时会创建一个新 ZFS 文件系统，并自动将其挂载。

以下示例说明如何通过一个命令同时创建名为 `tank` 的非冗余存储池和名为 `tank` 的 ZFS 文件系统。假定整个磁盘 `/dev/dsk/c1t0d0` 可供使用。

```
# zpool create tank c1t0d0
```

**注 –** 此命令将创建一个非冗余池。即使单个存储对象存在于硬件 RAID 阵列或软件卷管理器中，也建议不要将非冗余池配置用于生产环境。ZFS 只能检测这些配置中的错误。ZFS 可用冗余数据更正池配置中的错误。有关冗余 ZFS 池配置的更多信息，请参见第 38 页中的 “ZFS 存储池的复制功能”。

新 ZFS 文件系统 `tank` 可根据需要使用 `c1t0d0` 中任意大小的磁盘空间，并会自动挂载在 `/tank` 中。

```
# mkfile 100m /tank/foo
# df -h /tank
Filesystem             size   used  avail capacity  Mounted on
tank                   80G   100M    80G     1%    /tank
```

在池内，可能需要创建其他文件系统。文件系统可提供管理点，用于管理同一池中不同的数据集。

以下示例说明如何在存储池 `tank` 中创建名为 `fs` 的文件系统。假定整个磁盘 `/dev/dsk/c1t0d0` 可供使用。

```
# zpool create tank mirror c1t0d0 c2t0d0
# zfs create tank/fs
```

新 ZFS 文件系统 `tank/fs` 可根据需要使用 `c1t0d0` 中任意大小的磁盘空间，并会自动挂载在 `/tank/fs` 中。

```
# mkfile 100m /tank/fs/foo
# df -h /tank/fs
Filesystem             size   used  avail capacity  Mounted on
tank/fs                 80G   100M    80G     1%    /tank/fs
```

在大多数情况下，您可能要创建并组织与您公司的需要相符的文件系统分层结构。有关创建 ZFS 文件系统的分层结构的更多信息，请参见第 28 页中的 “创建 ZFS 文件系统分层结构”。

# 创建 ZFS 存储池

上一示例说明了 ZFS 的简单性。本章的其余部分将说明一个更复杂的示例，与您的环境中所遇到的情况相似。第一个任务是确定存储要求并创建存储池。该池描述了存储的物理特征，并且必须在创建任何文件系统之前创建。

## ▼ 如何确定 ZFS 存储池的存储要求

1 **确定可用设备。**

创建存储池之前，必须先确定用于存储数据的设备。这些设备必须是大小至少为 128 MB 的磁盘，并且不能由操作系统的其他部分使用。设备可以是预先格式化的磁盘上的单个片，也可以是 ZFS 格式化为单个大片的整个磁盘。

对于第 27 页中的 “如何创建 ZFS 存储池”中使用的存储示例，假定磁盘 `/dev/dsk/c1t0d0` 和 `/dev/dsk/c1t1d0` 全部都可供使用。

有关磁盘及其使用和标记方法的更多信息，请参见第 35 页中的 “使用 ZFS 存储池中的磁盘”。

2 **选择数据复制。**

ZFS 支持多种类型的数据复制，这确定了池可以经受的硬件故障的类型。ZFS 支持非冗余（条带化）配置以及镜像和 RAID-Z（RAID-5 的变化形式）。

第 27 页中的 “如何创建 ZFS 存储池”中使用的存储示例使用了两个可用磁盘的基本镜像。

有关 ZFS 复制功能的更多信息，请参见第 38 页中的 “ZFS 存储池的复制功能”。

## ▼ 如何创建 ZFS 存储池

1 **成为超级用户或承担具有适当 ZFS 权限配置文件的等效角色。**

有关 ZFS 权限配置文件的更多信息，请参见第 138 页中的 “ZFS 权限配置文件”。

2 **选择池名称。**

池名称用于在使用 `zpool` 或 `zfs` 命令时标识存储池。大多数系统都只需一个池，因此只要满足第 24 页中的 “ZFS 组件命名要求”中所述的命名要求，即可选择您喜欢的任何名称。

3 **创建池。**

例如，创建名为 `tank` 的镜像池。

```
# zpool create tank mirror c1t0d0 c1t1d0
```

如果一个或多个设备包含其他文件系统或正在使用中，则该命令不能创建池。

有关创建存储池的更多信息，请参见第 40 页中的 “创建 ZFS 存储池”。

有关如何确定设备使用情况的更多信息，请参见第 42 页中的 “检测使用中的设备”。

4 **查看结果。**

使用 `zpool list` 命令可以确定是否已成功创建池。

```
# zpool list
NAME                    SIZE    USED   AVAIL    CAP  HEALTH     ALTROOT
tank                     80G    137K     80G     0%  ONLINE     -
```

有关查看池状态的更多信息，请参见第 56 页中的 “查询 ZFS 存储池的状态” 。

# 创建 ZFS 文件系统分层结构

创建用于存储数据的存储池之后，即可创建文件系统分层结构。分层结构是用于组织信息的简单但功能强大的机制。使用过文件系统的任何用户对分层结构也都很熟悉。

使用 ZFS 可将文件系统组织为任意分层结构，其中每个文件系统仅有一个父级。分层结构的根始终是池名称。ZFS 通过支持属性继承来利用此分层结构，以便可在整个文件系统树中快速轻松地设置公用属性。

## ▼ 如何确定 ZFS 文件系统分层结构

1 **选择文件系统粒度。**

ZFS 文件系统是管理的中心点。它们是轻量型的，很容易创建。适用的模型是每个用户或项目对应一个文件系统，因为此模型允许按用户或按项目控制属性、快照和备份。

第 29 页中的 “如何创建 ZFS 文件系统”中创建了两个 ZFS 文件系统 `bonwick` 和 `billm`。

有关管理文件系统的更多信息，请参见第 5 章。

2 **对相似的文件系统进行分组。**

使用 ZFS 可将文件系统组织为分层结构，以便可对相似的文件系统进行分组。此模型提供了一个用于控制属性和管理文件系统的管理中心点。应使用一个公用名称来创建相似的文件系统。

对于第 29 页中的 “如何创建 ZFS 文件系统”中的示例，两个文件系统都放置在名为 `home` 的文件系统下。

3 **选择文件系统属性。**

大多数文件系统特征都是通过使用简单属性来控制的。这些属性可以控制多种行为，包括文件系统的挂载位置、共享方式、是否使用压缩以及是否有任何生效的配额。

对于第 29 页中的 “如何创建 ZFS 文件系统”中的示例，所有起始目录都挂载在 `/export/zfs/`*user* 中，都通过使用 NFS 来共享并且都已启用压缩。此外，还对 `bonwick` 强制实施了 10 GB 的配额。

有关属性的更多信息，请参见第 74 页中的 “ZFS 属性介绍” 。

## ▼ 如何创建 ZFS 文件系统

1 **成为超级用户或承担具有适当 ZFS 权限配置文件的等效角色。**

有关 ZFS 权限配置文件的更多信息，请参见第 138 页中的 “ZFS 权限配置文件” 。

2 **创建所需的分层结构。**

在本示例中，创建了一个可充当各文件系统的容器的文件系统。

```
# zfs create tank/home
```

然后，在池 tank 中的 home 文件系统下对各文件系统进行分组。

3 **设置继承的属性。**

建立文件系统分层结构之后，设置应在所有用户之间共享的任何属性：

```
# zfs set mountpoint=/export/zfs tank/home
# zfs set sharenfs=on tank/home
# zfs set compression=on tank/home
# zfs get compression tank/home
NAME             PROPERTY       VALUE                      SOURCE
tank/home        compression    on                         local
```

现在提供了一项新功能，通过该功能可在创建文件系统时设置文件系统属性。例如：

```
# zfs create -o mountpoint=/export/zfs -o sharenfs=on -o compression=on tank/home
```

有关属性和属性继承的更多信息，请参见第 74 页中的 “ZFS 属性介绍” 。

4 **创建各文件系统。**

请注意，文件系统可能已创建，并可能已在 home 级别更改了属性。所有属性均可在使用文件系统的过程中动态进行更改。

```
# zfs create tank/home/bonwick
# zfs create tank/home/billm
```

这些文件系统从其父级继承属性设置，因此会自动挂载在 /export/zfs/*user* 中并且通过 NFS 共享。您无需编辑 /etc/vfstab 或 /etc/dfs/dfstab 文件。

有关创建文件系统的更多信息，请参见第 72 页中的 “创建 ZFS 文件系统” 。

有关挂载和共享文件系统的更多信息，请参见第 89 页中的 “挂载和共享 ZFS 文件系统” 。

5 **设置文件系统特定的属性。**

在本示例中，为用户 bonwick 指定了 10 GB 的配额。此属性可对该用户可以使用的空间量施加限制，而无需考虑池中的可用空间大小。

```
# zfs set quota=10G tank/home/bonwick
```

6 **查看结果。**

使用 zfs list 命令查看可用的文件系统信息：

```
# zfs list
NAME                   USED  AVAIL  REFER  MOUNTPOINT
tank                  92.0K  67.0G   9.5K  /tank
tank/home             24.0K  67.0G     8K  /export/zfs
tank/home/billm          8K  67.0G     8K  /export/zfs/billm
tank/home/bonwick        8K  10.0G     8K  /export/zfs/bonwick
```

请注意，用户 bonwick 仅有 10 GB 的可用空间，而用户 billm 则可使用整个池 (67 GB)。

有关查看文件系统状态的更多信息，请参见第 83 页中的 “查询 ZFS 文件系统信息”。

有关空间的使用和计算方法的更多信息，请参见第 32 页中的 “ZFS 空间记帐”。

◆ ◆ ◆ 第 3 章

# 3 ZFS 与传统文件系统之间的差别

本章讨论 ZFS 与传统文件系统之间的一些重要差别。了解这些关键差别有助于在使用传统工具与 ZFS 进行交互时避免混淆。

本章包含以下各节：



## ZFS 文件系统粒度

以前，文件系统被局限于一个设备，因此文件系统自身会受到该设备大小的限制。由于存在大小限制，因此创建和重新创建传统文件系统很耗时，有时候还很难。传统的卷管理产品可帮助管理此过程。

由于 ZFS 文件系统不局限于特定设备，因此可以轻松、快捷地创建，其创建方法与目录的创建方法相似。在为存储池分配的空间内，ZFS 文件系统可以自动增长。

要管理许多用户子目录，可以为每个用户创建一个文件系统，而不是只创建一个文件系统（如 `/export/home`）。此外，ZFS 还提供了一个文件系统分层结构，这样只需应用分层结构内文件系统可继承的属性，便可轻松设置和管理许多文件系统。

有关创建文件系统分层结构的示例，请参见第 28 页中的 “创建 ZFS 文件系统分层结构”。

# ZFS 空间记帐

ZFS 建立在池存储概念的基础上。与典型文件系统映射到物理存储器不同，池中的所有 ZFS 文件系统都共享该池中的可用存储器。因此，即使文件系统处于非活动状态，实用程序（例如 `df`）报告的可用空间也会发生变化，因为池中的其他文件系统会使用或释放空间。注意，使用配额可以限制最大文件系统大小。有关配额的信息，请参见第 95 页中的 “设置 ZFS 文件系统的配额”。使用预留功能可以保证文件系统拥有相应空间。有关预留的信息，请参见第 96 页中的 “设置 ZFS 文件系统的预留空间”。此模型与从同一文件系统（例如 `/home`）挂载多个目录的 NFS 模型非常相似。

ZFS 中的所有元数据都是动态分配的。其他大部分文件系统都会预分配其大量元数据。因此，创建文件系统时需要针对此元数据的即时空间成本。此行为还意味着文件系统支持的文件总数是预先确定的。由于 ZFS 根据需要分配其元数据，因此不需要初始空间成本，并且文件数只受可用空间的限制。对于 ZFS 文件系统，对 `df -g` 命令输出的解释必须和其他文件系统不同。报告的 `total files` 只是根据池中可用的存储量得出的估计值。

ZFS 是事务性文件系统。大部分文件系统修改都捆绑到事务组中，并异步提交至磁盘。这些修改在被提交到磁盘之前称为**暂挂更改**。已用空间量、可用空间量以及文件或文件系统引用的空间量并不考虑暂挂更改。通常，暂挂更改仅占用几秒钟的时间。即使使用 `fsync(3c)` 或 `O_SYNC` 提交对磁盘的更改，也不一定可以保证有关空间使用情况的信息会立即更新。

## 空间不足行为

文件系统的快照开销很小，并且很容易在 ZFS 中创建。在大多数 ZFS 环境中，快照很可能是通用的。有关 ZFS 快照的信息，请参见第 6 章。

尝试释放空间时，快照的存在会引起某种意外行为。通常，获取适当的权限后，可从整个文件系统中删除一个文件，此操作会使文件系统有更多的可用空间。但是，如果要删除的文件存在于文件系统的快照中，则删除该文件不会获得任何空间。快照将继续引用该文件使用的块。

由于需要创建新版本的目录来反映名称空间的新状态，因此删除文件会占用更多的磁盘空间。此行为意味着，尝试删除文件时可能获得意外的 `ENOSPC` 或 `EDQUOT`。

# 挂载 ZFS 文件系统

ZFS 旨在降低复杂性和减轻管理负担。例如，如果使用现有文件系统，则必须在每次添加新文件系统时编辑 `/etc/vfstab` 文件。ZFS 可根据数据集的属性自动挂载和取消挂载文件系统，从而消除了上述要求。无需管理 `/etc/vfstab` 文件中的 ZFS 项。

有关挂载和共享 ZFS 文件系统的更多信息，请参见第 89 页中的 “挂载和共享 ZFS 文件系统”。

# 传统卷管理

如第 21 页中的 “ZFS 池存储” 中所述，ZFS 不需要单独的卷管理器。ZFS 对原始设备执行操作，因此可能会创建由逻辑卷（软件或硬件）构成的存储池。由于 ZFS 在使用原始物理设备时可获得最佳工作状态，因此建议不使用此配置。使用逻辑卷可能会牺牲性能和/或可靠性，因此应尽量避免。

# 新 Solaris ACL 模型

Solaris OS 的旧版本支持主要基于 POSIX 式 ACL 规范的 ACL 实现。基于 POSIX 草案的 ACL 用来保护 UFS 文件。基于 NFSv4 规范的新 ACL 模型用来保护 ZFS 文件。

与旧模型相比，新 Solaris ACL 模型的主要变化如下：

- 基于 NFSv4 规范并与 NT 样式的 ACL 相似。
- 提供了更详尽的访问权限集。
- 分别使用 `chmod` 和 `ls` 命令（而非 `setfacl` 和 `getfacl` 命令）进行设置和显示。
- 提供了更丰富的继承语义，用于指定如何将访问权限从目录应用到子目录等。

有关将 ACL 用于 ZFS 文件的更多信息，请参见第 7 章。

◆ ◆ ◆ 第 4 章

# 管理 ZFS 存储池

本章介绍如何创建和管理 ZFS 存储池。

本章包含以下各节：



## ZFS 存储池的组件

以下各节提供有关以下存储池组件的详细信息：



### 使用 ZFS 存储池中的磁盘

存储池的最基本元素是一个物理存储器。物理存储器可以是大小至少为 128 MB 的任何块设备。通常，此设备是 `/dev/dsk` 目录中对系统可见的一个硬盘驱动器。

存储设备可以是整个磁盘 (`c1t0d0`) 或单个片 (`c0t0d0s7`)。建议的操作模式是使用整个磁盘，在这种情况下，无需对磁盘专门进行格式化。ZFS 可格式化使用 EFI 标签的磁盘以包含单个大片。以此方式使用磁盘时，`format` 命令显示的分区表与以下信息类似：

```
Current partition table (original):
Total disk sectors available: 71670953 + 16384 (reserved sectors)
```

```
Part      Tag    Flag     First Sector         Size         Last Sector
  0        usr    wm                34       34.18GB          71670953
  1 unassigned    wm                 0           0                0
  2 unassigned    wm                 0           0                0
  3 unassigned    wm                 0           0                0
  4 unassigned    wm                 0           0                0
  5 unassigned    wm                 0           0                0
  6 unassigned    wm                 0           0                0
  7 unassigned    wm                 0           0                0
  8   reserved    wm          71670954        8.00MB          71687337
```

要使用整个磁盘，必须使用标准 Solaris 约定命名磁盘，如 `/dev/dsk/cXtXdXsX`。一些第三方驱动程序使用不同的命名约定，或者将磁盘放置在除 `/dev/dsk` 目录以外的位置中。要使用这些磁盘，必须手动标记磁盘并为 ZFS 提供片。

创建包含整个磁盘的存储池时，ZFS 会应用 EFI 标签。创建包含磁盘片的存储池时，可以使用传统的 Solaris VTOC 标签来标记磁盘。

应仅在以下情况下使用片：

- 设备名称是非标准名称。
- ZFS 和其他文件系统（如 UFS）之间共享单个磁盘。
- 磁盘用作交换设备或转储设备。

可以使用全路径（如 `/dev/dsk/c1t0d0`）或构成 `/dev/dsk` 目录中设备名称的缩略名称（如 `c1t0d0`）来指定磁盘。例如，以下是有效的磁盘名称：

- `c1t0d0`
- `/dev/dsk/c1t0d0`
- `c0t0d6s2`
- `/dev/foo/disk`

如果为存储池指定整个磁盘，ZFS 将使用**整个**磁盘。这意味着将删除已定义的任何现有 `fdisk` 分区。如果要在具有现有 `fdisk` 分区的磁盘上创建 ZFS 存储池，则可以通过指定片 (`c1t0d0s7`) 而不是整个磁盘 (`c1t0d0`) 来创建存储池。

创建 ZFS 存储池的最简单方法是使用整个物理磁盘。在从磁盘片、硬件 RAID 阵列中的 LUN 或基于软件的卷管理器所提供的卷中生成池时，无论从管理、可靠性还是性能的角度而言，ZFS 配置都变得越来越复杂。以下注意事项可能有助于确定如何用其他硬件或软件存储解决方案来配置 ZFS：

- 如果在硬件 RAID 阵列中的 LUN 上构建 ZFS 配置，则需要了解 ZFS 冗余功能与该阵列所提供的冗余功能之间的关系。有些配置可能会提供足够的冗余和性能，而其他配置可能不会提供足够的冗余和性能。
- 可以使用基于软件的卷管理器（如 Solaris™ 卷管理器 (Solaris Volume Manager, SVM) 或 Veritas 卷管理器 (Veritas Volume Manager, VxVM)）所提供的卷来为 ZFS 构建逻辑设备。但是，建议不要使用这些配置。尽管 ZFS 可在这类设备上正常运行，但结果可能是实际性能低于最佳性能。

有关存储池建议的其他信息，请参见 ZFS 最佳做法站点：

http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide

磁盘由其路径及其设备 ID（如果可用）标识。使用此方法，可以在系统中重新配置设备，而不必更新任何 ZFS 状态。如果磁盘在控制器 1 和控制器 2 之间切换，则 ZFS 可使用设备 ID 检测到该磁盘已移动，并且现在应使用控制器 2 对其进行访问。设备 ID 对于驱动器固件是唯一的。尽管不大可能，但确实有一些固件更新更改了设备 ID。如果发生这种情况，ZFS 仍可以按路径访问设备，并自动更新存储的设备 ID。如果无意中同时更改了设备的路径和 ID，则将池导出再重新导入后才能使用该池。

## 使用 ZFS 存储池中的文件

ZFS 还允许将 UFS 文件用作存储池中的虚拟设备。此功能主要用于测试和启用简单的实验，而不是用于生产。原因是**文件的任何使用都依赖于基础文件系统以实现一致性**。如果创建了由 UFS 文件系统中的文件支持的 ZFS 池，即会隐式依赖于 UFS 来保证正确性和同步语义。

但是，如果首次试用 ZFS，或者在没有足够的物理设备时尝试更复杂的布局，则文件会非常有用。所有文件必须以完整路径的形式指定，并且大小至少为 64 MB。如果移动或重命名某个文件，则必须将池导出再重新导入才能使用该池，这是因为没有设备 ID（可以按其查找文件）与文件相关联。

## 标识存储池中的虚拟设备

每个存储池都由一个或多个虚拟设备组成。**虚拟设备**是存储池的内部表示形式，用于说明物理存储的布局及其故障特征。因此，虚拟设备表示用于创建存储池的磁盘设备或文件。

两种顶层虚拟设备可提供数据冗余：镜像虚拟设备和 RAID-Z 虚拟设备。这些虚拟设备由磁盘、磁盘片或文件构成。

在镜像虚拟设备和 RAID-Z 虚拟设备之外的池中使用的磁盘、磁盘片或文件本身用作顶层虚拟设备。

存储池通常由多个顶层虚拟设备构成。ZFS 将在池内的所有顶层虚拟设备中以动态方式对数据进行条带化。

# ZFS 存储池的复制功能

ZFS 在镜像配置和 RAID-Z 配置中提供数据冗余和自我修复属性。



## 镜像存储池配置

镜像存储池配置至少需要两个磁盘，而且磁盘最好位于不同的控制器上。可以在一个镜像配置中使用许多磁盘。此外，还可以在每个池中创建多个镜像。从概念上讲，简单的镜像配置与以下内容类似：

```
mirror c1t0d0 c2t0d0
```

从概念上讲，更复杂的镜像配置与以下内容类似：

```
mirror c1t0d0 c2t0d0 c3t0d0 mirror c4t0d0 c5t0d0 c6t0d0
```

有关创建镜像存储池的信息，请参见第 40 页中的 “创建镜像存储池”。

## RAID-Z 存储池配置

除镜像存储池配置外，ZFS 还提供具有单奇偶校验容错性或双奇偶校验容错性的 RAID-Z 配置。单奇偶校验 RAID-Z 与 RAID-5 类似。双奇偶校验 RAID-Z 与 RAID-6 类似。

所有与 RAID-5 类似的传统算法（例如 RAID-4、RAID-6、RDP 和 EVEN-ODD）都存在称为 “RAID-5 写入漏洞” 的问题。如果仅写入了 RAID-5 条带的一部分，并且在所有块成功写入磁盘之前断电，则奇偶校验将永远与数据不同步，因此是无用的，除非后续的完全条带化写操作将其覆写。在 RAID-Z 中，ZFS 使用可变宽度的 RAID 条带，以便所有写操作都是完全条带化写操作。这是唯一可行的设计，因为 ZFS 通过以下方式将文件系统和设备管理集成在一起：文件系统的元数据包含有关基础数据冗余模型的足够信息以处理可变宽度的 RAID 条带。RAID-Z 是世界上针对 RAID-5 写入漏洞的第一个仅使用软件的解决方案。

一个 RAID-Z 配置包含 N 个大小为 X 的磁盘，其中有 P 个奇偶校验磁盘，该配置可以存放大约 (N-P)*X 字节的数据，并且只有在 P 个设备出现故障时才会危及数据完整性。单奇偶校验 RAID-Z 配置至少需要两个磁盘，双奇偶校验 RAID-Z 配置至少需要三个磁盘。例如，如果一个单奇偶校验 RAID-Z 配置中有三个磁盘，则奇偶校验数据占用的空间与其中一个磁盘的空间相等。除此之外，创建 RAID-Z 配置无需任何其他特殊硬件。

从概念上讲，包含三个磁盘的 RAID-Z 配置与以下内容类似：

```
raidz c1t0d0 c2t0d0 c3t0d0
```

从概念上讲，更复杂的 RAID-Z 配置与以下内容类似：

```
raidz c1t0d0 c2t0d0 c3t0d0 c4t0d0 c5t0d0 c6t0d0 c7t0d0 raidz c8t0d0 c9t0d0 c10t0d0 c11t0d0
c12t0d0 c13t0d0 c14t0d0
```

如果要创建包含许多磁盘的 RAID-Z 配置（如本示例所示），则最好将包含 14 个磁盘的 RAID-Z 配置拆分为两个包含 7 个磁盘的分组。若 RAID-Z 配置包含的分组中的磁盘数目为一位数 (1-9)，则该配置的性能应该更好。

有关创建 RAID-Z 存储池的信息，请参见第 41 页中的 “创建 RAID-Z 存储池”。

有关基于性能和空间考虑在镜像配置或 RAID-Z 配置之间进行选择的更多信息，请参见以下 blog：

http://blogs.sun.com/roller/page/roch?entry=when_to_and_not_to

有关 RAID-Z 存储池建议的其他信息，请参见 ZFS 最佳做法站点：

http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide

# 冗余配置中的自我修复数据

ZFS 在镜像配置或 RAID-Z 配置中提供了自我修复数据。

检测到坏的数据块时，ZFS 不仅会从另一个冗余副本中提取正确的数据，还会通过将错误数据替换为正确的副本对其进行修复。

# 存储池中的动态条带化

对于添加到池中的每个虚拟设备，ZFS 会跨越所有可用设备以动态方式对数据进行条带化。由于是在写入时确定放置数据的位置，因此在分配时不会创建固定宽度的条带。

向池中添加虚拟设备时，ZFS 会将数据逐渐分配给新设备，以便维护性能和空间分配策略。每个虚拟设备也可以是包含其他磁盘设备或文件的镜像或 RAID-Z 设备。使用此配置，可以灵活地控制池的故障特征。例如，可以通过 4 个磁盘创建以下配置：

- 使用动态条带化的四个磁盘
- 一个四向 RAID-Z 配置
- 使用动态条带化的两个双向镜像

尽管 ZFS 支持在同一池中组合不同类型的虚拟设备，但是建议不要采用这种做法。例如，可以创建一个包含一个双向镜像和一个三向 RAID-Z 配置的池。但是，容错能力几乎与最差的虚拟设备（在本示例中为 RAID-Z）相同。建议做法是使用相同类型的顶层虚拟设备，并且每个设备的冗余级别相同。

# 创建和销毁 ZFS 存储池

以下各节介绍创建和销毁 ZFS 存储池的不同情况。



根据设计，可快速轻松地创建和销毁池。但是，执行这些操作请务必谨慎。虽然进行了检查，以防止在新的池中使用现已使用的设备，但是 ZFS 无法始终知道设备何时已在使用中。销毁池更为容易。请谨慎使用 `zpool destroy`。这是一个会产生重大后果的简单命令。

## 创建 ZFS 存储池

要创建存储池，请使用 `zpool create` 命令。此命令采用池名称和任意数目的虚拟设备作为参数。池名称必须符合第 24 页中的 “ZFS 组件命名要求”中概述的命名约定。

### 创建基本存储池

以下命令创建了一个名为 `tank` 的新池，该池由磁盘 `c1t0d0` 和 `c1t1d0` 组成：

```
# zpool create tank c1t0d0 c1t1d0
```

这些整个磁盘可在 `/dev/dsk` 目录中找到，并由 ZFS 适当标记以包含单个大片。数据通过这两个磁盘以动态方式进行条带化。

### 创建镜像存储池

要创建镜像池，请使用 `mirror` 关键字，后跟将组成镜像的任意数目的存储设备。可以通过在命令行中重复使用 `mirror` 关键字指定多个镜像。以下命令创建了一个包含两个双向镜像的池：

```
# zpool create tank mirror c1d0 c2d0 mirror c3d0 c4d0
```

第二个 `mirror` 关键字表示将指定新的顶层虚拟设备。数据通过这两个镜像以动态方式进行条带化，并会相应地在每个磁盘之间创建冗余数据。

目前，ZFS 镜像配置中支持以下操作：

- 向现有镜像配置中添加用于其他顶层 `vdev` 的另一组磁盘。有关更多信息，请参见第 45 页中的 “向存储池中添加设备”。
- 向现有镜像配置中附加其他磁盘。或者，向非复制配置中附加其他磁盘，以创建镜像配置。有关更多信息，请参见第 47 页中的 “附加和分离存储池中的设备”。
- 只要可供替换的磁盘大于或等于要被替换的设备，便可替换现有镜像配置中的一个或多个磁盘。有关更多信息，请参见第 51 页中的 “替换存储池中的设备”。

- 只要剩余设备可为配置提供足够冗余，便可分离镜像配置中的一个或多个磁盘。有关更多信息，请参见第 47 页中的 “附加和分离存储池中的设备”。

目前，镜像配置中不支持以下操作：

- 不能从镜像存储池中彻底删除设备。对于此功能，已经申请了 RFE（请求提高）。
- 不能出于备份目的而分割或中断镜像。对于此功能，已经申请了 RFE（请求提高）。

## 创建 RAID-Z 存储池

创建单奇偶校验 RAID-Z 池与创建镜像池基本相同，不同之处是使用 `raidz` 或 `raidz1` 关键字而不是 `mirror`。以下示例说明如何创建一个包含由 5 个磁盘组成的单个 RAID-Z 设备的池：

```
# zpool create tank raidz c1t0d0 c2t0d0 c3t0d0 c4t0d0 /dev/dsk/c5t0d0
```

本示例表明可以使用全路径指定相应的磁盘。`/dev/dsk/c5t0d0` 设备与 `c5t0d0` 设备相同。

可以使用磁盘片创建类似的配置。例如：

```
# zpool create tank raidz c1t0d0s0 c2t0d0s0 c3t0d0s0 c4t0d0s0 c5t0d0s0
```

但是，必须预先格式化磁盘，使其包含适当大小的片 0。

可在创建池时使用 `raidz2` 关键字来创建双奇偶校验 RAID-Z 配置。例如：

```
# zpool create tank raidz2 c1t0d0 c2t0d0 c3t0d0
# zpool status -v tank
  pool: tank
 state: ONLINE
 scrub: none requested
config:

        NAME          STATE     READ WRITE CKSUM
        tank          ONLINE       0     0     0
          raidz2      ONLINE       0     0     0
            c1t0d0    ONLINE       0     0     0
            c2t0d0    ONLINE       0     0     0
            c3t0d0    ONLINE       0     0     0

errors: No known data errors
```

目前，ZFS RAID-Z 配置中支持以下操作：

- 向现有 RAID-Z 配置中添加用于其他顶层 `vdev` 的另一组磁盘。有关更多信息，请参见第 45 页中的 “向存储池中添加设备”。

- 只要可供替换的磁盘大于或等于要被替换的设备，便可替换现有 RAID-Z 配置中的一个或多个磁盘。有关更多信息，请参见第 51 页中的 “替换存储池中的设备”。

目前，RAID-Z 配置中不支持以下操作：

- 向现有 RAID-Z 配置中附加其他磁盘。
- 从 RAID-Z 配置中分离磁盘。
- 不能从 RAID-Z 配置中彻底删除设备。对于此功能，已经申请了 RFE（请求提高）。

有关 RAID-Z 配置的更多信息，请参见第 38 页中的 “RAID-Z 存储池配置”。

# 处理 ZFS 存储池创建错误

出现池创建错误可以有许多原因。其中一些原因是显而易见的（如指定的设备不存在），而其他原因则不太明显。

## 检测使用中的设备

格式化设备之前，ZFS 会首先确定 ZFS 或操作系统的某个其他部分是否正在使用磁盘。如果磁盘正在使用，则可能会显示类似以下的错误：

```
# zpool create tank c1t0d0 c1t1d0
invalid vdev specification
use '-f' to override the following errors:
/dev/dsk/c1t0d0s0 is currently mounted on /. Please see umount(1M).
/dev/dsk/c1t0d0s1 is currently mounted on swap. Please see swap(1M).
/dev/dsk/c1t1d0s0 is part of active ZFS pool zeepool. Please see zpool(1M).
```

使用 `-f` 选项可以覆盖其中的一些错误，但是无法覆盖大多数错误。使用 `-f` 选项无法覆盖使用以下各项产生的错误，必须手动对这些错误进行更正：

**挂载的文件系统** 磁盘或其中一片包含当前挂载的文件系统。要更正此错误，请使用 `umount` 命令。

**/etc/vfstab 中的文件系统** 磁盘包含 `/etc/vfstab` 文件中列出的文件系统，但当前未挂载该文件系统。要更正此错误，请删除或注释掉 `/etc/vfstab` 文件中的相应行。

**专用转储设备** 正在将磁盘用作系统的专用转储设备。要更正此错误，请使用 `dumpadm` 命令。

**ZFS 池的一部分** 磁盘或文件是活动 ZFS 存储池的一部分。要更正此错误，请使用 `zpool` 命令销毁池。

以下使用情况检查用作帮助性警告，并可以使用 `-f` 选项进行覆盖以创建池：

**包含文件系统** 磁盘包含已知的文件系统，尽管该系统未挂载并且看起来未被使用。

| | |
|---|---|
| **卷的一部分** | 磁盘是 SVM 卷的一部分。 |
| **实时升级** | 正在将磁盘用作 Solaris Live Upgrade 的替换引导环境。 |
| **导出的 ZFS 池的一部分** | 磁盘是已导出的或者从系统中手动删除的存储池的一部分。如果是后一种情况，则会将池的状态报告为**可能处于活动状态**，因为磁盘可能是也可能不是由其他系统使用的网络连接驱动器。覆盖可能处于活动状态的池时请务必谨慎。 |

以下示例说明如何使用 `-f` 选项：

```
# zpool create tank c1t0d0
invalid vdev specification
use '-f' to override the following errors:
/dev/dsk/c1t0d0s0 contains a ufs filesystem.
# zpool create -f tank c1t0d0
```

理想的情况是，更正错误而不是使用 `-f` 选项。

## 不匹配的复制级别

建议不要创建包含不同复制级别的虚拟设备的池。`zpool` 命令可尝试防止意外创建冗余级别不匹配的池。如果尝试创建具有这样配置的池，则会显示类似以下的错误：

```
# zpool create tank c1t0d0 mirror c2t0d0 c3t0d0
invalid vdev specification
use '-f' to override the following errors:
mismatched replication level: both disk and mirror vdevs are present
# zpool create tank mirror c1t0d0 c2t0d0 mirror c3t0d0 c4t0d0 c5t0d0
invalid vdev specification
use '-f' to override the following errors:
mismatched replication level: 2-way mirror and 3-way mirror vdevs are present
```

可以使用 `-f` 选项覆盖这些错误，但建议不要采用这种做法。此命令还会发出警告，指明正使用大小不同的设备创建镜像池或 RAID-Z 池。尽管允许此配置，但是冗余级别不匹配会导致较大设备上产生未使用的空间，并要求使用 `-f` 选项覆盖警告。

## 在预运行模式下创建存储池

由于创建池可能以不同方式意外失败，并且格式化磁盘这一操作可能产生危害，因此 `zpool create` 命令具有一个附加选项 `-n`，此选项可用于模拟创建池，而无需实际将数据写入磁盘。此选项执行设备使用中检查和复制级别验证，并报告该过程中出现的任何错误。如果未找到错误，则会显示类似以下的输出：

```
# zpool create -n tank mirror c1t0d0 c1t1d0
would create 'tank' with the following layout:
```

```
        tank
          mirror
            c1t0d0
            c1t1d0
```

如果不实际创建池，则无法检测到某些错误。最常见的示例是在同一配置中两次指定同一设备。由于不写入数据本身便无法可靠地检测到此错误，因此 `create -n` 命令可能会报告运行成功，但在实际运行时又无法创建池。

### 存储池的缺省挂载点

创建池时，根数据集的缺省挂载点是 */pool-name*。此目录必须不存在或者为空。如果目录不存在，则会自动创建该目录。如果该目录为空，则根数据集会挂载在现有目录的顶层。要使用不同的缺省挂载点创建池，请在 `zpool create` 命令中使用 `-m` 选项：

```
# zpool create home c1t0d0
default mountpoint '/home' exists and is not empty
use '-m' option to specify a different default
# zpool create -m /export/zfs home c1t0d0

# zpool create home c1t0d0
default mountpoint '/home' exists and is not empty
use '-m' option to provide a different default
# zpool create -m /export/zfs home c1t0d0
```

此命令会创建一个新池 `home` 和挂载点为 `/export/zfs` 的 `home` 数据集。

有关挂载点的更多信息，请参见第 89 页中的 “管理 ZFS 挂载点”。

## 销毁 ZFS 存储池

池是通过使用 `zpool destroy` 命令进行销毁的。即使池中包含已挂载的数据集，此命令仍会销毁池。

```
# zpool destroy tank
```

**注意 –** 销毁池时请务必小心。请确保确实要销毁池，并始终保留数据副本。如果意外销毁了不该销毁的池，则可以尝试恢复该池。有关更多信息，请参见第 66 页中的 “恢复已销毁的 ZFS 存储池”。

### 销毁包含故障设备的池

销毁池这一操作要求将数据写入磁盘，以指示池不再有效。此状态信息可防止执行导入操作时这些设备作为潜在的池显示出来。在一个或多个设备不可用的情况下，仍可以销毁池。但是，必需的状态信息将不会写入这些损坏的设备。

创建新池时，这些设备在适当修复后其状态将报告为**可能处于活动状态**，并在搜索要导入的池时会显示为有效设备。如果池中包含足够多的故障设备以致于池本身出现故障（意味着顶层虚拟设备出现故障），则此命令将列显一条警告，并且在不使用 `-f` 选项的情况下无法完成。此选项是必需的，因为无法打开池，以致无法知道数据是否存储在池中。例如：

```
# zpool destroy tank
cannot destroy 'tank': pool is faulted
use '-f' to force destruction anyway
# zpool destroy -f tank
```

有关池和设备的运行状况的更多信息，请参见第 59 页中的 “确定 ZFS 存储池的运行状况”。

有关导入池的更多信息，请参见第 65 页中的 “导入 ZFS 存储池”。

# 管理 ZFS 存储池中的设备

第 35 页中的 “ZFS 存储池的组件”中介绍了有关设备的大多数基本信息。创建池后，即可执行几项任务来管理池中的物理设备。

- 第 45 页中的 “向存储池中添加设备”
- 第 47 页中的 “附加和分离存储池中的设备”
- 第 49 页中的 “使存储池中的设备联机和脱机”
- 第 50 页中的 “清除存储池设备”
- 第 51 页中的 “替换存储池中的设备”
- 第 52 页中的 “在存储池中指定热备件”

## 向存储池中添加设备

通过添加新的顶层虚拟设备，可以向池中动态添加空间。此空间立即可供池中的所有数据集使用。要向池中添加新虚拟设备，请使用 `zpool add` 命令。例如：

```
# zpool add zeepool mirror c2t1d0 c2t2d0
```

虚拟设备的格式与 zpool create 命令中使用的虚拟设备格式相同，并且应用相同的规则。将对设备进行检查以确定是否正在使用这些设备，此命令在不使用 -f 选项的情况下无法更改冗余级别。此命令还支持 -n 选项，以便可以执行预运行。例如：

```
# zpool add -n zeepool mirror c3t1d0 c3t2d0
would update 'zeepool' to the following configuration:
      zeepool
        mirror
            c1t0d0
            c1t1d0
        mirror
            c2t1d0
            c2t2d0
        mirror
            c3t1d0
            c3t2d0
```

此命令语法会将镜像设备 c3t1d0 和 c3t2d0 添加到 zeepool 的现有配置中。

有关如何执行虚拟设备验证的更多信息，请参见第 42 页中的 “检测使用中的设备”。

**示例 4–1** 向 RAID-Z 配置中添加磁盘

可按类似方式向 RAID-Z 配置中添加其他磁盘。以下示例说明如何将包含一个 RAID-Z 设备（由 3 个磁盘构成）的存储池转换为包含两个 RAID-Z 设备（由 3 个磁盘构成）的存储池。

```
# zpool status
  pool: rpool
 state: ONLINE
 scrub: none requested
config:
        NAME          STATE     READ WRITE CKSUM
        rpool         ONLINE       0     0     0
          raidz1      ONLINE       0     0     0
            c1t2d0    ONLINE       0     0     0
            c1t3d0    ONLINE       0     0     0
            c1t4d0    ONLINE       0     0     0

errors: No known data errors
# zpool add rpool raidz c2t2d0 c2t3d0 c2t4d0
# zpool status
  pool: rpool
 state: ONLINE
 scrub: none requested
config:
        NAME          STATE     READ WRITE CKSUM
```

**示例 4–1** 向 RAID-Z 配置中添加磁盘 （续）

```
        rpool        ONLINE       0     0     0
          raidz1     ONLINE       0     0     0
            c1t2d0   ONLINE       0     0     0
            c1t3d0   ONLINE       0     0     0
            c1t4d0   ONLINE       0     0     0
          raidz1     ONLINE       0     0     0
            c2t2d0   ONLINE       0     0     0
            c2t3d0   ONLINE       0     0     0
            c2t4d0   ONLINE       0     0     0

errors: No known data errors
```

# 附加和分离存储池中的设备

除了 `zpool add` 命令外，还可以使用 `zpool attach` 命令将新设备添加到现有镜像设备或非镜像设备中。

**示例 4–2** 将双向镜像存储池转换为三向镜像存储池

在本示例中，`zeepool` 是现有的双向镜像，通过将新设备 `c2t1d0` 附加到现有设备 `c1t1d0` 可将其转换为三向镜像。

```
# zpool status
  pool: zeepool
 state: ONLINE
 scrub: none requested
config:
        NAME        STATE     READ WRITE CKSUM
        zeepool     ONLINE       0     0     0
          mirror    ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
errors: No known data errors
# zpool attach zeepool c1t1d0 c2t1d0
# zpool status
  pool: zeepool
 state: ONLINE
 scrub: resilver completed with 0 errors on Fri Jan 12 14:47:36 2007
config:

        NAME        STATE     READ WRITE CKSUM
        zeepool     ONLINE       0     0     0
          mirror    ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
```

示例 4–2 将双向镜像存储池转换为三向镜像存储池 （续）

```
            c1t1d0  ONLINE       0     0     0
            c2t1d0  ONLINE       0     0     0
```

如果现有设备是双向镜像的一部分，则附加新设备将创建三向镜像，依此类推。在任一情况下，新设备都会立即开始重新同步。

示例 4–3 将非冗余 ZFS 存储池转换为镜像 ZFS 存储池

此外，还可以通过使用 zpool attach 命令将非冗余存储池转换为冗余存储池。例如：

```
# zpool create tank c0t1d0
# zpool status
  pool: tank
 state: ONLINE
 scrub: none requested
config:
        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          c0t1d0    ONLINE       0     0     0

errors: No known data errors
# zpool attach tank c0t1d0 c1t1d0
# zpool status
  pool: tank
 state: ONLINE
 scrub: resilver completed with 0 errors on Fri Jan 12 14:55:48 2007
config:
        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror    ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
```

可以使用 zpool detach 命令从镜像存储池中分离设备。例如：

```
# zpool detach zeepool c2t1d0
```

但是，如果不存在数据的其他有效副本，则拒绝此操作。例如：

```
# zpool detach newpool c1t2d0
cannot detach c1t2d0: only applicable to mirror and replacing vdevs
```

# 使存储池中的设备联机和脱机

使用 ZFS 可使单个设备脱机或联机。硬件不可靠或无法正常工作时（假定该情况只是暂时的），ZFS 会继续对设备读写数据。如果该情况不是暂时的，则可能会指示 ZFS 通过使设备脱机来忽略该设备。ZFS 不会向已脱机的设备发送任何请求。

---

**注 –** 设备无需脱机即可进行替换。

---

需要临时断开存储器时，可以使用 `offline` 命令。例如，如果需要以物理方式将阵列与一组光纤通道交换机断开连接并将该阵列连接到另一组交换机，则可使 LUN 从 ZFS 存储池所使用的阵列中脱机。在该阵列重新连接并可正常使用新的一组交换机后，便可使上述 LUN 联机。在 LUN 脱机期间添加到存储池中的数据将在 LUN 恢复联机后重新同步到 LUN 中。

假定所涉及的系统在连接到新交换机后可以立即看到存储器（可能与以前采用不同的控制器），并且您的池设置为 RAID-Z 配置或镜像配置，则上述情况是可能的。

## 使设备脱机

可以使用 `zpool offline` 命令使设备脱机。如果设备是磁盘，则可以使用路径或短名称指定设备。例如：

```
# zpool offline tank c1t0d0
bringing device c1t0d0 offline
```

使设备脱机时，请牢记以下要点：

- 不能将池脱机到它出现故障的点。例如，不能使 RAID-Z 配置中的两个设备脱机，也不能使顶层虚拟设备脱机。

  ```
  # zpool offline tank c1t0d0
  cannot offline c1t0d0: no valid replicas
  ```

- 缺省情况下，脱机状态是持久性的。重新引导系统时，设备会一直处于脱机状态。

  要暂时使设备脱机，请使用 `zpool offline -t` 选项。例如：

  ```
  # zpool offline -t tank c1t0d0
   bringing device 'c1t0d0' offline
  ```

  重新引导系统时，此设备会自动恢复到 `ONLINE` 状态。

- 当设备脱机时，它不会从存储池中分离出来。如果尝试使用其他池中的脱机设备，那么即使在销毁原始池之后，也会看到类似如下内容的消息：

  *device* `is part of exported or potentially active ZFS` *pool*`. Please see zpool(1M)`

如果要在销毁原始存储池之后使用其他存储池中的脱机设备，请先使该设备恢复联机，然后销毁原始存储池。

如果要保留原始存储池，则使用其他存储池中的设备的另一种方法是用另一个类似的设备替换原始存储池中的现有设备。有关替换设备的信息，请参见第 51 页中的 “替换存储池中的设备”。

查询池的状态时，已脱机的设备以 OFFLINE 状态显示。有关查询池的状态的信息，请参见第 56 页中的 “查询 ZFS 存储池的状态”。

有关设备运行状况的更多信息，请参见第 59 页中的 “确定 ZFS 存储池的运行状况”。

### 使设备联机

使设备脱机后，即可使用 zpool online 命令对其进行恢复：

```
# zpool online tank c1t0d0
bringing device c1t0d0 online
```

使设备联机时，已写入池中的任何数据都将与最新可用的设备重新同步。请注意，不能通过使设备联机来替换磁盘。如果使设备脱机，替换驱动器，然后再尝试使该设备联机，则设备将一直处于故障状态。

如果尝试使故障设备联机，则使用 fmd 将显示以下类似消息：

```
# zpool online tank c1t0d0
Bringing device c1t0d0 online
#
SUNW-MSG-ID: ZFS-8000-D3, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Thu Aug 31 11:13:59 MDT 2006
PLATFORM: SUNW,Ultra-60, CSN: -, HOSTNAME: neo
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: e11d8245-d76a-e152-80c6-e63763ed7e4f
DESC: A ZFS device failed.  Refer to http://sun.com/msg/ZFS-8000-D3 for more information.
AUTO-RESPONSE: No automated response will occur.
IMPACT: Fault tolerance of the pool may be compromised.
REC-ACTION: Run ’zpool status -x’ and replace the bad device.
```

有关替换故障设备的更多信息，请参见第 147 页中的 “修复缺少的设备”。

## 清除存储池设备

如果设备因出现故障（导致在 zpool status 输出中列出错误）而脱机，则可以使用 zpool clear 命令清除错误计数。

如果不指定任何参数，则此命令将清除池中的所有设备错误。例如：

```
# zpool clear tank
```

如果指定了一个或多个设备，则此命令仅清除与指定设备关联的错误。例如：

```
# zpool clear tank c1t0d0
```

有关清除 zpool 错误的更多信息，请参见第 150 页中的 “清除瞬态错误” 。

# 替换存储池中的设备

可以使用 zpool replace 命令替换存储池中的设备。

如果使用冗余池中同一位置的另一设备以物理方式替换某一设备，则只需标识被替换的设备。ZFS 会识别出这是位于同一位置的不同磁盘。例如，要通过删除磁盘并在同一位置替换该磁盘来替换出现故障的磁盘 (c1t1d0)，请使用类似以下内容的语法：

```
# zpool replace tank c1t1d0
```

如果要替换的设备位于只包含一个设备的非冗余存储池中，则需要同时指定两个设备。例如：

```
# zpool replace tank c1t1d0 c1t2d0
```

在替换 ZFS 存储池中的设备时，请切记以下注意事项：

- 可供替换的设备的大小必须大于或等于镜像配置或 RAID-Z 配置中所有设备的最小大小。
- 如果可供替换的设备较大，则完成替换后池容量将增加。目前，必须导出并导入池，才能查看扩展的容量。例如：

```
# zpool list tank
NAME   SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
tank  16.8G    94K  16.7G     0%  ONLINE  -
# zpool replace tank c0t0d0 c0t4d0
# zpool list tank
NAME   SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
tank  16.8G   112K  16.7G     0%  ONLINE  -
# zpool export tank
# zpool import tank
# zpool list tank
NAME   SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
tank  33.9G   114K  33.9G     0%  ONLINE  -
```

有关导出和导入池的更多信息，请参见第 61 页中的 “迁移 ZFS 存储池”。

- 目前，在增大属于存储池的现有 LUN 的大小时，也必须执行导出和导入步骤，才能查看扩展的容量。
- 替换较大池中的多个磁盘需要较长时间，这是因为需要将数据重新同步到新磁盘。此外，还可以考虑在两次磁盘替换操作之间运行 `zpool scrub` 命令，以确保可供替换的设备可以正常运行，并且正确写入数据。

有关如何替换设备的更多信息，请参见第 147 页中的 “修复缺少的设备”和第 149 页中的 “修复损坏的设备”。

# 在存储池中指定热备件

借助热备件功能，可以在一个或多个存储池中确定能用来替换发生故障或失败的磁盘。指定一个设备作为**热备件**意味着该设备不是池中的活动设备，但如果池中的某一活动设备发生故障，热备件将自动替换该故障设备。

可通过以下方式将设备指定为热备件：

- 使用 `zpool create` 命令创建池时
- 使用 `zpool add` 命令创建池之后
- 热备件设备可在多个池之间共享

在创建池时将设备指定为热备件。例如：

```
# zpool create zeepool mirror c1t1d0 c2t1d0 spare c1t2d0 c2t2d0
# zpool status zeepool
pool: zeepool
 state: ONLINE
 scrub: none requested
config:

        NAME         STATE     READ WRITE CKSUM
        zeepool      ONLINE       0     0     0
          mirror     ONLINE       0     0     0
            c1t1d0   ONLINE       0     0     0
            c2t1d0   ONLINE       0     0     0
        spares
          c1t2d0     AVAIL
          c2t2d0     AVAIL
```

通过在创建池之后将设备添加到池中来指定热备件。例如：

```
# zpool add -f zeepool spare c1t3d0 c2t3d0
# zpool status zeepool
pool: zeepool
```

```
 state: ONLINE
 scrub: none requested
config:

        NAME         STATE     READ WRITE CKSUM
        zeepool      ONLINE       0     0     0
          mirror     ONLINE       0     0     0
            c1t1d0   ONLINE       0     0     0
            c2t1d0   ONLINE       0     0     0
        spares
          c1t3d0     AVAIL
          c2t3d0     AVAIL
```

指定为热备件的设备可由多个池共享。例如：

```
# zpool create zeepool mirror c1t1d0 c2t1d0 spare c1t2d0 c2t2d0
# zpool create tank raidz c3t1d0 c4t1d0 spare c1t2d0 c2t2d0
```

可使用 `zpool remove` 命令从存储池中删除热备件。例如：

```
# zpool remove zeepool c1t2d0
# zpool status zeepool
pool: zeepool
 state: ONLINE
 scrub: none requested
config:

        NAME         STATE     READ WRITE CKSUM
        zeepool      ONLINE       0     0     0
          mirror     ONLINE       0     0     0
            c1t1d0   ONLINE       0     0     0
            c2t1d0   ONLINE       0     0     0
        spares
          c1t3d0     AVAIL
```

如果存储池当前正在使用热备件，则不能将其删除。

使用 ZFS 热备件时，请切记以下几点：

- 目前，`zpool remove` 命令只能用来删除热备件。
- 添加磁盘作为备件时，该磁盘的大小应等于或大于池中最大磁盘的大小。允许向池中添加更小的磁盘作为备件。但是，当自动激活或使用 `zpool replace` 命令激活较小的备用磁盘时，操作将失败，并显示类似以下内容的错误：

  ```
  cannot replace disk3 with disk4: device is too small
  ```

## 在存储池中激活和取消激活热备件

可通过以下方式激活热备件：

- 手动替换－通过 zpool replace 命令用热备件替换存储池中的故障设备。
- 自动替换－接收到故障信息后，FMA 代理将检查池中是否有任何可用的热备件。如果有，将使用可用备件替换故障设备。

  如果当前正在使用的热备件发生故障，代理将分离该备件，从而取消替换。然后，代理将尝试用另一个热备件（如果有）替换该设备。目前，由于 ZFS 诊断引擎仅在设备从系统中消失时才会发出故障信息，因此此功能受到限制。

  目前没有自动化响应可使原始设备恢复联机。必须显式执行以下示例中介绍的操作之一。未来的增强功能将允许 ZFS 订阅热插拔事件并在系统上发生替换时自动替换受影响的设备。

通过 zpool replace 命令使用热备件手动替换设备。例如：

```
# zpool replace zeepool c2t1d0 c2t3d0
# zpool status zeepool
  pool: zeepool
 state: ONLINE
 scrub: resilver completed with 0 errors on Fri Jun  2 13:44:40 2006
config:

        NAME            STATE     READ WRITE CKSUM
        zeepool         ONLINE       0     0     0
          mirror        ONLINE       0     0     0
            c1t2d0      ONLINE       0     0     0
            spare       ONLINE       0     0     0
              c2t1d0    ONLINE       0     0     0
              c2t3d0    ONLINE       0     0     0
        spares
          c1t3d0        AVAIL
          c2t3d0        INUSE     currently in use

errors: No known data errors
```

如果热备件可用，将自动替换故障设备。例如：

```
# zpool status -x
  pool: zeepool
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-D3
 scrub: resilver completed with 0 errors on Fri Jun  2 13:56:49 2006
```

```
config:

        NAME                 STATE     READ WRITE CKSUM
        zeepool              DEGRADED     0     0     0
          mirror             DEGRADED     0     0     0
            c1t2d0           ONLINE       0     0     0
            spare            DEGRADED     0     0     0
              c2t1d0         UNAVAIL      0     0     0  cannot open
              c2t3d0         ONLINE       0     0     0
        spares
          c1t3d0             AVAIL
          c2t3d0             INUSE     currently in use

errors: No known data errors
```

目前，有三种方式可用来取消激活热备件：

- 通过从存储池中删除热备件来取消热备件
- 用热备件替换原始设备
- 永久性换入热备件

替换故障设备后，用 `zpool detach` 命令使热备件返回备件集。例如：

```
# zpool detach zeepool c2t3d0
# zpool status zeepool
  pool: zeepool
 state: ONLINE
 scrub: resilver completed with 0 errors on Fri Jun  2 13:58:35 2006
config:

        NAME                 STATE     READ WRITE CKSUM
        zeepool              ONLINE       0     0     0
          mirror             ONLINE       0     0     0
            c1t2d0           ONLINE       0     0     0
            c2t1d0           ONLINE       0     0     0
        spares
          c1t3d0             AVAIL
          c2t3d0             AVAIL

errors: No known data errors
```

# 查询 ZFS 存储池的状态

zpool list 命令提供了许多方法来请求有关池状态的信息。可用信息通常分为以下三个类别：基本使用情况信息、I/O 统计信息和运行状况。本节介绍了所有这三种类型的存储池信息。



## 显示基本的 ZFS 存储池信息

可以使用 zpool list 命令显示有关池的基本信息。

### 列出有关所有存储池的信息

如果不使用参数，则此命令会显示系统中所有池的所有字段。例如：

```
# zpool list
NAME                    SIZE    USED   AVAIL    CAP  HEALTH     ALTROOT
tank                   80.0G   22.3G   47.7G    28%  ONLINE     -
dozer                   1.2T    384G    816G    32%  ONLINE     -
```

此输出显示了以下信息：

NAME
: 池的名称。

SIZE
: 池的总大小，等于所有顶层虚拟设备大小的总和。

USED
: 由所有数据集和内部元数据分配的空间量。请注意，此数量与在文件系统级别报告的空间量不同。

  有关确定可用文件系统空间的更多信息，请参见第 32 页中的 “ZFS 空间记帐” 。

AVAILABLE
: 池中未分配的空间量。

CAPACITY (CAP)
: 已用空间量，以总空间的百分比表示。

HEALTH
: 池的当前运行状况。

  有关池运行状况的更多信息，请参见第 59 页中的 “确定 ZFS 存储池的运行状况” 。

ALTROOT
: 池的备用根（如果有）。

  有关备用根池的更多信息，请参见第 137 页中的 “使用 ZFS 备用根池” 。

另外，也可以通过指定池的名称来收集特定池的统计信息。例如：

```
# zpool list tank
NAME                    SIZE    USED   AVAIL    CAP  HEALTH     ALTROOT
tank                   80.0G   22.3G   47.7G    28%  ONLINE     -
```

### 列出特定的存储池统计信息

可以使用 `-o` 选项请求特定的统计信息。使用此选项可以生成自定义报告或快速列出相关信息。例如，要仅列出每个池的名称和大小，可使用以下语法：

```
# zpool list -o name,size
NAME                     SIZE
tank                    80.0G
dozer                    1.2T
```

列名称与第 56 页中的 “列出有关所有存储池的信息”中列出的属性相对应。

### 使用脚本处理 ZFS 存储池输出

`zpool list` 命令的缺省输出旨在提高可读性，因此不能轻易用作 shell 脚本的一部分。为了便于在程序中使用该命令，可以使用 `-H` 选项以便不显示列标题，并使用制表符而不是空格分隔字段。例如，请求系统中所有池的名称的简单列表：

```
# zpool list -Ho name
tank
dozer
```

以下是另一个示例：

```
# zpool list -H -o name,size
tank   80.0G
dozer  1.2T
```

## 查看 ZFS 存储池 I/O 统计信息

要请求池或特定虚拟设备的 I/O 统计信息，请使用 `zpool iostat` 命令。与 `iostat` 命令类似，此命令也可以显示目前为止所有 I/O 活动的静态快照，以及每个指定时间间隔的更新统计信息。报告的统计信息如下：

`USED CAPACITY`
: 当前存储在池或设备中的数据量。由于具体的内部实现的原因，此数字与可供实际文件系统使用的空间量有少量差异。

  有关池空间与数据集空间之间的差异的更多信息，请参见第 32 页中的 “ZFS 空间记帐”。

| | |
|---|---|
| AVAILABLE CAPACITY | 池或设备中的可用空间量。与 used 统计信息一样，这与可供数据集使用的空间量也有少量差异。 |
| READ OPERATIONS | 发送到池或设备的读取 I/O 操作数，包括元数据请求。 |
| WRITE OPERATIONS | 发送到池或设备的写入 I/O 操作数。 |
| READ BANDWIDTH | 所有读取操作（包括元数据）的带宽，以每秒单位数表示。 |
| WRITE BANDWIDTH | 所有写入操作的带宽，以每秒单位数表示。 |

## 列出池范围的统计信息

如果不使用任何选项，则 zpool iostat 命令会显示自引导以来系统中所有池的累积统计信息。例如：

```
# zpool iostat
               capacity     operations    bandwidth
pool         used  avail   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
tank         100G  20.0G   1.2M   102K   1.2M  3.45K
dozer       12.3G  67.7G   132K  15.2K  32.1K  1.20K
```

由于这些统计信息是自引导以来累积的，因此，如果池相对空闲，则带宽可能显示为较低。通过指定时间间隔，可以请求查看更准确的当前带宽使用情况。例如：

```
# zpool iostat tank 2
               capacity     operations    bandwidth
pool         used  avail   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
tank         100G  20.0G   1.2M   102K   1.2M  3.45K
tank         100G  20.0G    134      0  1.34K      0
tank         100G  20.0G     94    342  1.06K   4.1M
```

在本示例中，此命令仅显示了池 tank 的使用情况统计信息，每隔两秒显示一次，直到键入 Ctrl-C 组合键为止。或者，也可以再指定 count 参数，该参数用于使命令在指定的重复次数之后终止。例如， zpool iostat 2 3 每隔两秒列显一次摘要信息，重复三次，共六秒。如果存在单个池，则会在连续的行上显示统计信息。如果存在多个池，则用附加虚线分隔每次重复，以提供直观的分隔效果。

## 列出虚拟设备的统计信息

除了池范围的 I/O 统计信息外，zpool iostat 命令还可以显示特定虚拟设备的统计信息。此命令可用于识别异常缓慢的设备，或者只是观察 ZFS 生成的 I/O 的分布情况。要请求完整的虚拟设备布局以及所有 I/O 统计信息，请使用 zpool iostat -v 命令。例如：

```
# zpool iostat -v
               capacity     operations    bandwidth
tank         used  avail   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
mirror      20.4G  59.6G      0     22      0  6.00K
  c1t0d0        -      -      1    295  11.2K   148K
  c1t1d0        -      -      1    299  11.2K   148K
----------  -----  -----  -----  -----  -----  -----
total       24.5K   149M      0     22      0  6.00K
```

按虚拟设备查看 I/O 统计信息时，请注意两个重要事项。

- 首先，只有顶层虚拟设备才有空间使用量。在镜像和 RAID-Z 虚拟设备中分配空间的方法是特定于实现的，不能简单地表示为一个数字。
- 其次，这些数字可能不会完全按期望的那样累加。具体来说，通过 RAID-Z 设备和通过镜像设备进行的操作不是完全均等的。这种差异在创建池之后即特别明显，因为在创建池的过程中直接对磁盘执行了大量 I/O，但在镜像级别并没有考虑这些 I/O。随着时间的推移，这些数字应会逐渐相等，尽管损坏的、无响应的或脱机的设备也可能影响此对称性。

检查虚拟设备统计信息时，可以使用相同的一组选项（时间间隔和计次）。

# 确定 ZFS 存储池的运行状况

ZFS 提供了一种检查池和设备运行状况的集成方法。池的运行状况是根据其所有设备的状态确定的。使用 `zpool status` 命令可以显示此状态信息。此外，池和设备的可能故障由 `fmd` 报告，并会显示在系统控制台上和 `/var/adm/messages` 文件中。本节介绍如何确定池和设备的运行状况。本章不介绍如何修复运行不良的池或从其恢复。有关疑难解答和数据恢复的更多信息，请参见第 9 章。

每个设备都可以处于以下状态之一：

| | |
|---|---|
| `ONLINE` | 设备处于正常工作状态。尽管仍然可能会出现一些瞬态错误，但是设备在其他方面处于正常工作状态。 |
| `DEGRADED` | 虚拟设备出现故障，但仍能够工作。此状态在镜像或 RAID-Z 设备缺少一个或多个组成设备时最为常见。池的容错能力可能会受到损害，因为另一个设备中的后续故障可能无法恢复。 |
| `FAULTED` | 虚拟设备完全无法访问。此状态通常表示设备出现全面故障，以致于 ZFS 无法向该设备发送数据或从该设备接收数据。如果顶层虚拟设备处于此状态，则完全无法访问池。 |
| `OFFLINE` | 管理员已将虚拟设备显式脱机。 |
| `UNAVAILABLE` | 无法打开设备或虚拟设备。在某些情况下，包含 `UNAVAILABLE` 设备的池会以 `DEGRADED` 模式显示。如果顶层虚拟设备不可用，则无法访问池中 |

的任何设备。

池的运行状况是根据其所有顶层虚拟设备的运行状况确定的。如果所有虚拟设备状态都为 ONLINE，则池的状态也为 ONLINE。如果任何一个虚拟设备状态为 DEGRADED 或 UNAVAILABLE，则池的状态也为 DEGRADED。如果顶层虚拟设备的状态为 FAULTED 或 OFFLINE，则池的状态也为 FAULTED。处于故障状态的池完全无法访问。附加或修复必需的设备后，才能恢复数据。处于降级状态的池会继续运行，但是，如果池处于联机状态，则可能无法实现相同级别的数据冗余或数据吞吐量。

## 基本的存储池运行状况

请求池运行状况的简单概述的最简单方法是使用 zpool status 命令：

```
# zpool status -x
all pools are healthy
```

通过为命令指定池名称，可以检查特定池。如下节所述，应检查不处于 ONLINE 状态的所有池是否存在潜在的问题。

## 详细运行状况

使用 -v 选项可以请求更详细的运行状况摘要。例如：

```
# zpool status -v tank
  pool: tank
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist
        for the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-2Q
 scrub: none requested
config:

        NAME                STATE     READ WRITE CKSUM
        tank                DEGRADED     0     0     0
          mirror            DEGRADED     0     0     0
            c1t0d0          FAULTED      0     0     0  cannot open
            c1t1d0          ONLINE       0     0     0
errors: No known data errors
```

此输出显示了池处于其当前状态的原因的完整说明，其中包括问题的易读说明，以及指向知识文章（用于了解更多信息）的链接。每篇知识文章都提供了有关从当前问题恢复的最佳方法的最新信息。使用详细的配置信息，应可确定损坏的设备以及修复池的方法。

在以上示例中，故障设备应该被替换。替换该设备后，请使用 zpool online 命令使设备恢复联机。例如：

```
# zpool online tank c1t0d0
Bringing device c1t0d0 online
# zpool status -x
all pools are healthy
```

如果池包含脱机设备，则命令输出将标识有问题的池。例如：

```
# zpool status -x
  pool: tank
 state: DEGRADED
status: One or more devices has been taken offline by the adminstrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using 'zpool online' or replace the device with
        'zpool replace'.
 scrub: none requested
config:

        NAME         STATE     READ WRITE CKSUM
        tank         DEGRADED     0     0     0
          mirror     DEGRADED     0     0     0
            c1t0d0   ONLINE       0     0     0
            c1t1d0   OFFLINE      0     0     0

errors: No known data errors
```

READ 和 WRITE 列提供了在设备上发现的 I/O 错误的计数，而 CKSUM 列则提供了在设备上出现的无法更正的校验和错误的计数。这两种错误计数可能会指示可能的设备故障，并且需要执行更正操作。如果针对顶层虚拟设备报告了非零错误，则表明部分数据可能无法访问。错误计数可标识任何已知的数据错误。

在以上示例输出中，脱机设备不会导致数据错误。

有关诊断和修复故障池和数据的更多信息，请参见第 9 章。

# 迁移 ZFS 存储池

有时，可能需要在计算机之间移动存储池。为此，必须将存储设备与原始计算机断开，然后将其重新连接到目标计算机。可以通过以下方法完成此任务：以物理方式重新为设备布线，或者使用多端口设备（如 SAN 中的设备）。使用 ZFS 可将池从一台计算机中导出，然后将其导入目标计算机，即使这两台计算机采用不同的字节存储顺序

(endianness)。有关在不同存储池（可能驻留在不同的计算机中）之间复制或迁移文件系统的信息，请参见第 103 页中的 “保存和恢复 ZFS 数据” 。



# 准备迁移 ZFS 存储池

应显式导出存储池，以表明可随时将其迁移。此操作会将任何未写入的数据刷新到磁盘，将数据写入磁盘以表明导出已完成，并从系统中删除有关池的所有信息。

如果不显式导出池，而是改为手动删除磁盘，则仍可以在其他系统中导入生成的池。但是，可能会丢失最后几秒的数据事务，并且由于设备不再存在，该池在原始计算机中可能会显示为处于故障状态。缺省情况下，目标计算机会拒绝导入未显式导出的池。对于防止意外导入包含仍在其他系统中使用的网络连接存储器的活动池，此条件是必要的。

# 导出 ZFS 存储池

要导出池，请使用 `zpool export` 命令。例如：

```
# zpool export tank
```

执行此命令后，池 `tank` 在系统中即不再可见。此命令将尝试取消挂载池中任何已挂载的文件系统，然后再继续执行。如果无法取消挂载任何文件系统，则可以使用 `-f` 选项强制取消挂载这些文件系统。例如：

```
# zpool export tank
cannot unmount '/export/home/eschrock': Device busy
# zpool export -f tank
```

如果在导出时设备不可用，则无法将磁盘指定为正常导出。如果之后将某个这样的设备附加到不包含任何工作设备的系统中，则该设备的状态会显示为 “可能处于活动状态” 。如果 ZFS 卷在池中处于使用状态，即使使用 `-f` 选项，也无法导出池。要导出包含 ZFS 卷的池，请首先确保卷的所有使用者都不再处于活动状态。

有关 ZFS 卷的更多信息，请参见第 131 页中的 “ZFS 卷” 。

# 确定要导入的可用存储池

从系统中删除池后（通过导出或通过强制删除设备），请立即将设备附加到目标系统。虽然 ZFS 可以处理仅有部分设备可用的一些情况，但是必须在系统之间移动池中的所有设备。没有必要使用相同的设备名称附加设备。ZFS 可检测任何移动的或重命名的设备，并相应地调整配置。要搜索可用的池，请运行不带任何选项的 `zpool import` 命令。例如：

```
# zpool import
  pool: tank
    id: 3778921145927357706
 state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

        tank        ONLINE
          mirror    ONLINE
            c1t0d0  ONLINE
            c1t1d0  ONLINE
```

在本示例中，池 `tank` 可用于在目标系统中导入。每个池都由一个名称以及唯一的数字标识符标识。如果可用于导入的多个池具有相同名称，则可以使用数字标识符对其进行区分。

与 `zpool status` 命令类似，`zpool import` 命令也会引用可在 Web 上获取的知识文章，其中包含有关禁止导入池这一问题的修复过程的最新信息。在此示例中，用户可以强制导入池。但是，如果导入当前正由其他系统通过存储网络使用的池，则可能导致数据损坏和出现紧急情况，因为这两个系统都尝试写入同一存储器。如果池中的某些设备不可用，但是存在足够的冗余，可确保池可用，则池会显示 `DEGRADED` 状态。例如：

```
# zpool import
  pool: tank
    id: 3778921145927357706
 state: DEGRADED
status: One or more devices are missing from the system.
action: The pool can be imported despite missing or damaged devices.  The
        fault tolerance of the pool may be compromised if imported.
   see: http://www.sun.com/msg/ZFS-8000-2Q
config:

        tank         DEGRADED
          mirror     DEGRADED
            c1t0d0   UNAVAIL   cannot open
            c1t1d0   ONLINE
```

在本示例中，第一个磁盘已损坏或缺失，但仍可以导入池，这是因为仍可以访问镜像数据。如果存在过多故障设备或缺失设备，则无法导入池。例如：

```
# zpool import
  pool: dozer
    id: 12090808386336829175
 state: FAULTED
action: The pool cannot be imported. Attach the missing
        devices and try again.
   see: http://www.sun.com/msg/ZFS-8000-6X
config:
        raidz               FAULTED
          c1t0d0    ONLINE
          c1t1d0    FAULTED
          c1t2d0    ONLINE
          c1t3d0    FAULTED
```

在本示例中，RAID-Z 虚拟设备中缺少两个磁盘，这意味着没有足够的可用冗余数据来重新构建池。在某些情况下，没有足够的设备就无法确定完整的配置。在这种情况下，虽然 ZFS 会尽可能多地报告有关该情况的信息，但是 ZFS 仍然无法知道池中包含的其他设备。例如：

```
# zpool import
pool: dozer
    id: 12090808386336829175
 state: FAULTED
status: One or more devices are missing from the system.
action: The pool cannot be imported. Attach the missing
        devices and try again.
   see: http://www.sun.com/msg/ZFS-8000-6X
config:
        dozer          FAULTED   missing device
          raidz       ONLINE
            c1t0d0    ONLINE
            c1t1d0    ONLINE
            c1t2d0    ONLINE
            c1t3d0    ONLINE
        Additional devices are known to be part of this pool, though their
        exact configuration cannot be determined.
```

# 从替换目录中查找 ZFS 存储池

缺省情况下，`zpool import` 命令仅在 `/dev/dsk` 目录中搜索设备。如果设备存在于其他目录中，或者使用的是文件支持的池，则必须使用 `-d` 选项搜索其他目录。例如：

```
# zpool create dozer mirror /file/a /file/b
# zpool export dozer
# zpool import -d /file
  pool: dozer
    id: 10952414725867935582
 state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

        dozer        ONLINE
          mirror     ONLINE
            /file/a  ONLINE
            /file/b  ONLINE
# zpool import -d /file dozer
```

如果设备存在于多个目录中，则可以指定多个 `-d` 选项。

# 导入 ZFS 存储池

确定要导入的池后，即可通过将该池的名称或者其数字标识符指定为 `zpool import` 命令的参数来将其导入。例如：

```
# zpool import tank
```

如果多个可用池具有相同名称，则可以使用数字标识符指定要导入的池。例如：

```
# zpool import
  pool: dozer
    id: 2704475622193776801
 state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

        dozer       ONLINE
          c1t9d0    ONLINE

  pool: dozer
    id: 6223921996155991199
 state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

        dozer       ONLINE
          c1t8d0    ONLINE
# zpool import dozer
cannot import 'dozer': more than one matching pool
```

```
import by numeric ID instead
# zpool import 6223921996155991199
```

如果该池的名称与现有的池名称冲突，则可以使用其他名称导入该池。例如：

```
# zpool import dozer zeepool
```

此命令使用新名称 zeepool 导入已导出的池 dozer。如果池未正常导出，则 ZFS 需要使用 -f 标志，以防止用户意外导入仍在其他系统中使用的池。例如：

```
# zpool import dozer
cannot import 'dozer': pool may be in use on another system
use '-f' to import anyway
# zpool import -f dozer
```

也可以使用 -R 选项在备用根下导入池。有关备用根池的更多信息，请参见第 137 页中的 “使用 ZFS 备用根池”。

## 恢复已销毁的 ZFS 存储池

可以使用 zpool import -D 命令恢复已销毁的存储池。例如：

```
# zpool destroy tank
# zpool import -D
pool: tank
    id: 3778921145927357706
 state: ONLINE (DESTROYED)
action: The pool can be imported using its name or numeric identifier.  The
        pool was destroyed, but can be imported using the '-Df' flags.
config:

        tank        ONLINE
          mirror    ONLINE
            c1t0d0  ONLINE
            c1t1d0  ONLINE
```

在以上的 zpool import 输出中，由于包含以下状态信息，因此可以将该池确定为已销毁的池：

```
state: ONLINE (DESTROYED)
```

要恢复已销毁的池，请再次执行 zpool import -D 命令，并指定要恢复的池和 -f 选项。例如：

```
# zpool import -Df tank
# zpool status tank
```

```
  pool: tank
 state: ONLINE
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror    ONLINE       0     0     0
            c1t0d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0

errors: No known data errors
```

如果已销毁池中的某个设备出现故障或不可用，但还是有可能恢复已销毁的池。在此情况下，请导入已降级的池，然后尝试修复设备故障。例如：

```
# zpool destroy dozer
# zpool import -D
pool: dozer
    id:
 state: DEGRADED (DESTROYED)
status: One or more devices are missing from the system.
action: The pool can be imported despite missing or damaged devices.  The
        fault tolerance of the pool may be compromised if imported.  The
        pool was destroyed, but can be imported using the '-Df' flags.
   see: http://www.sun.com/msg/ZFS-8000-2Q
config:

        dozer        DEGRADED
           raidz      ONLINE
            c1t0d0    ONLINE
            c1t1d0    ONLINE
            c1t2d0    UNAVAIL  cannot open
            c1t3d0    ONLINE
# zpool import -Df dozer
# zpool status -x
  pool: dozer
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-D3
 scrub: resilver completed with 0 errors on Fri Mar 17 16:11:35 2006
config:

        NAME                     STATE     READ WRITE CKSUM
        dozer                    DEGRADED     0     0     0
```

```
          raidz                 ONLINE       0     0     0
            c1t0d0              ONLINE       0     0     0
            c1t1d0              ONLINE       0     0     0
            c1t2d0              UNAVAIL      0     0     0  cannot open
            c1t3d0              ONLINE       0     0     0

errors: No known data errors
# zpool online dozer c1t2d0
Bringing device c1t2d0 online
# zpool status -x
all pools are healthy
```

# 升级 ZFS 存储池

如果您具有以前的 Solaris 发行版（如 Solaris 10 6/06 发行版）的 ZFS 存储池，则可使用 `zpool upgrade` 命令升级池，以利用 Solaris 10 11/06 发行版中的池功能。此外，`zpool status` 命令已经修改，可在池运行较早的版本时发出通知。例如：

```
# zpool status
  pool: test
 state: ONLINE
status: The pool is formatted using an older on-disk format.  The pool can
        still be used, but some features are unavailable.
action: Upgrade the pool using 'zpool upgrade'.  Once this is done, the
        pool will no longer be accessible on older software versions.
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        test        ONLINE       0     0     0
          c1t27d0   ONLINE       0     0     0

errors: No known data errors
```

可以使用以下语法来确定有关特殊版本和支持的发行版的其他信息。

```
# zpool upgrade -v
This system is currently running ZFS version 3.

The following versions are supported:

VER  DESCRIPTION
---  --------------------------------------------------------
 1   Initial ZFS version
 2   Ditto blocks (replicated metadata)
 3   Hot spares and double parity RAID-Z
```

```
For more information on a particular version, including supported releases, see:

http://www.opensolaris.org/os/community/zfs/version/N

Where 'N' is the version number.
```

然后，可通过运行 `zpool upgrade` 命令来升级池。例如：

```
# zpool upgrade -a
```

**注 –** 如果将池升级到最新版本，则在运行较早 ZFS 版本的系统中将无法访问这些池。

◆ ◆ ◆ 第 5 章

# 管理 ZFS 文件系统

本章提供有关管理 Solaris™ ZFS 文件系统的详细信息。本章包括分层文件系统布局、属性继承和自动挂载点管理以及共享交互等概念。

ZFS 文件系统是在存储池顶层生成的轻量 POSIX 文件系统。文件系统可以动态创建和销毁，而不需要分配或格式化任何基础空间。由于文件系统是轻量型的，并且是 ZFS 中的管理中心点，因此可能要创建许多文件系统。

使用 `zfs` 命令可以管理 ZFS 文件系统。`zfs` 命令提供了一组用于对文件系统执行特定操作的子命令。本章详细介绍了这些子命令。使用此命令还可以管理快照、卷和克隆，但本章仅对这些功能进行了简短介绍。有关快照和克隆的详细信息，请参见第 6 章。有关仿真卷的详细信息，请参见第 131 页中的 “ZFS 卷”。

**注 –** 术语**数据集**在本章中用作通称，表示文件系统、快照、克隆或卷。

本章包含以下各节：

# 创建和销毁 ZFS 文件系统

可以使用 zfs create 和 zfs destroy 命令来创建和销毁 ZFS 文件系统。



## 创建 ZFS 文件系统

使用 zfs create 命令可以创建 ZFS 文件系统。create 子命令仅使用一个参数：要创建的文件系统的名称。将文件系统名称指定为从池名称开始的路径名：

*pool-name/[filesystem-name/]filesystem-name*

路径中的池名称和初始文件系统名称标识分层结构中要创建新文件系统的位置。所有中间文件系统的名称必须已在池中存在。路径中的最后一个名称标识要创建的文件系统的名称。文件系统名称必须满足第 24 页中的 “ZFS 组件命名要求”中定义的命名约定。

在以下示例中，在 tank/home 文件系统中创建了一个名为 bonwick 的文件系统。

```
# zfs create tank/home/bonwick
```

如果新文件系统创建成功，则 ZFS 会自动挂载该文件系统。缺省情况下，文件系统将使用 create 子命令中为文件系统名称提供的路径挂载为 */dataset*。在本示例中，新创建的 bonwick 文件系统位于 /tank/home/bonwick 中。有关自动管理的挂载点的更多信息，请参见第 89 页中的 “管理 ZFS 挂载点”。

有关 zfs create 命令的更多信息，请参见 zfs(1M)。

可在创建文件系统时设置文件系统属性。

在以下示例中，指定并为 tank/home 文件系统创建了挂载点 /export/zfs。

```
# zfs create -o mountpoint=/export/zfs tank/home
```

有关文件系统属性的更多信息，请参见第 74 页中的 “ZFS 属性介绍”。

## 销毁 ZFS 文件系统

要销毁 ZFS 文件系统，请使用 zfs destroy 命令。销毁的文件系统将自动取消挂载，并取消共享。有关自动管理的挂载或自动管理的共享的更多信息，请参见第 90 页中的 “自动挂载点”。

在以下示例中，销毁了 tabriz 文件系统。

```
# zfs destroy tank/home/tabriz
```

**注意 –** 使用 `destroy` 子命令时不会出现确认提示。请务必谨慎使用该子命令。

如果要销毁的文件系统处于繁忙状态并因此无法取消挂载，则 `zfs destroy` 命令将失败。要销毁活动文件系统，请使用 `-f` 选项。由于此选项可取消挂载、取消共享和销毁活动文件系统，从而导致意外的应用程序行为，因此请谨慎使用此选项。

```
# zfs destroy tank/home/ahrens
cannot unmount 'tank/home/ahrens': Device busy

# zfs destroy -f tank/home/ahrens
```

如果文件系统具有子级，则 `zfs destroy` 命令也会失败。要以递归方式销毁文件系统及其所有后代，请使用 `-r` 选项。请注意，递归销毁同时会销毁快照，因此请谨慎使用此选项。

```
# zfs destroy tank/ws
cannot destroy 'tank/ws': filesystem has children
use '-r' to destroy the following datasets:
tank/ws/billm
tank/ws/bonwick
tank/ws/maybee

# zfs destroy -r tank/ws
```

如果要销毁的文件系统具有间接依赖项，那么即使是上述递归销毁命令也会失败。要强制销毁**所有**依赖项（包括目标分层结构外的克隆文件系统），必须使用 `-R` 选项。请务必谨慎使用此选项。

```
# zfs destroy -r tank/home/schrock
cannot destroy 'tank/home/schrock': filesystem has dependent clones
use '-R' to destroy the following datasets:
tank/clones/schrock-clone

# zfs destroy -R tank/home/schrock
```

**注意 –** 使用 `-f`、`-r` 或 `-R` 选项时不会出现确认提示，因此请谨慎使用这些选项。

有关快照和克隆的更多信息，请参见第 6 章。

## 重命名 ZFS 文件系统

使用 zfs rename 命令可重命名文件系统。使用 rename 子命令可以执行以下操作：

- 更改文件系统的名称
- 将文件系统重定位到 ZFS 分层结构中的新位置。
- 更改文件系统的名称并在 ZFS 分层结构中对其重定位

以下示例使用 rename 子命令对文件系统进行简单重命名：

```
# zfs rename tank/home/kustarz tank/home/kustarz_old
```

本示例将 kustarz 文件系统重命名为 kustarz_old。

以下示例说明如何使用 zfs rename 重定位文件系统。

```
# zfs rename tank/home/maybee tank/ws/maybee
```

在本示例中，maybee 文件系统从 tank/home 重定位到 tank/ws。通过重命名来重定位文件系统时，新位置必须位于同一池中，并且必须具有足够的空间来存放这一新文件系统。如果新位置没有足够空间（可能是因为已达到配额），则重命名将失败。

有关配额的更多信息，请参见第 95 页中的 “ZFS 配额和预留空间”。

重命名操作会尝试对文件系统以及任何后代文件系统顺序执行取消挂载/重新挂载。如果该操作无法取消挂载活动文件系统，则重命名将失败。如果出现这一问题，将需要强制取消挂载文件系统。

有关重命名快照的信息，请参见第 99 页中的 “重命名 ZFS 快照”。

# ZFS 属性介绍

属性是用来对文件系统、卷、快照和克隆的行为进行控制的主要机制。除非另行说明，否则本节中定义的属性适用于所有数据集类型。



属性分为两种类型：本机属性和用户定义的属性。本机属性用于导出内部统计信息或控制 ZFS 文件系统行为。此外，本机属性是可设置的或只读的。用户属性对 ZFS 文件系统行为没有影响，但可通过用户环境中有意义的方式来注释数据集。有关用户属性的更多信息，请参见第 81 页中的 “ZFS 用户属性”。

大多数可设置的属性也是可继承的。可继承属性是这样的属性：如果为父级设置了该属性，则该属性会向下传播给其所有后代。

所有可继承属性都有一个关联源。源用于指明获取属性的方法。属性的源可具有以下值：

| | |
|---|---|
| `local` | `local` 源表示属性是使用 `zfs set` 命令对数据集进行显式设置的，如第 85 页中的 “设置 ZFS 属性”中所述。 |
| `inherited from` *dataset-name* | 值为 `inherited from` *dataset-name* 表示属性是从指定的祖先继承的。 |
| `default` | 值为 `default` 表示属性设置不是继承或本地设置的。如果没有祖先具有属性源 `local`，则会使用此源。 |

下表介绍了只读的和可设置的本机 ZFS 文件系统属性。只读本机属性在表中注明为 “只读属性”。此表中列出的所有其他本机属性均为可设置的属性。有关用户属性的信息，请参见第 81 页中的 “ZFS 用户属性”。

**表 5–1** ZFS 本机属性说明

| 属性名 | 类型 | 缺省值 | 说明 |
|---|---|---|---|
| `aclinherit` | 字符串 | `secure` | 控制创建文件和目录时继承 ACL 项的方法。该属性的值包括 `discard`、`noallow`、`secure` 和 `passthrough`。有关这些值的说明，请参见第 112 页中的 “ACL 属性模式”。 |
| `aclmode` | 字符串 | `groupmask` | 控制在 `chmod` 操作过程中修改 ACL 项的方法。该属性的值包括 `discard`、`groupmask` 和 `passthrough`。有关这些值的说明，请参见第 112 页中的 “ACL 属性模式”。 |
| `atime` | 布尔值 | `on` | 控制文件被读取后是否更新该文件的访问时间。禁用该属性可避免在读取文件时产生写入流量，因此可显著提高性能，但可能会使邮件程序与其他相似的实用程序感到困惑。 |
| `available` | 数字 | N/A | 只读属性，用于确定可供数据集及其所有子级使用的空间量，假定池中没有其他活动。由于池中会共享空间，因此可用空间会受到多种因素的限制，包括物理池大小、配额、预留空间或池中的其他数据集。<br>该属性也可通过其简短列名 `avail` 来引用。<br>有关空间记帐的更多信息，请参见第 32 页中的 “ZFS 空间记帐”。 |
| `canmount` | 布尔值 | on | 控制是否可以使用 `zfs mount` 命令挂载给定的文件系统。在任意文件系统中均可设置该属性，该属性本身不可继承。不过，在设置该属性后，后代文件系统可以继承挂载点，但永远不会挂载文件系统本身。有关更多信息，请参见第 80 页中的 “canmount 属性”。 |

表 5–1 ZFS 本机属性说明 （续）

| 属性名 | 类型 | 缺省值 | 说明 |
| --- | --- | --- | --- |
| checksum | 字符串 | on | 控制用于验证数据完整性的校验和。缺省值为 on，这将自动选择合适的算法，当前算法为 fletcher2。该属性的值包括 on、off、fletcher2、fletcher4 和 sha256。值为 off 将禁用对用户数据的完整性检查。建议不要使用值 off。 |
| compression | 字符串 | off | 控制用于此数据集的压缩算法。目前，可以选择 lzjb、gzip 或 gzip-*N*。在包含现有数据的文件系统中启用压缩将只压缩新数据。现有数据保持未压缩状态。<br>该属性也可通过其简短列名 compress 来引用。 |
| compressratio | 数字 | N/A | 只读属性，用于标识针对此数据集实现的压缩比例，表示为乘数。通过运行 zfs set compression=on *dataset* 可以启用压缩。<br>根据所有文件的逻辑大小和引用的物理数据量进行计算。包括通过使用 compression 属性实现的明确的压缩节省量。 |
| copies | 数字 | 1 | 设置每个文件系统的用户数据副本数。可用值包括 1、2 或 3。这些副本是对任何池级别冗余的补充。用户数据的多个副本所使用的空间将在相应的文件和数据集中进行计费，并根据配额和预留空间进行计数。此外，启用多个副本时还会更新 used 属性。由于在现有文件系统中更改该属性只影响新写入的数据，因此请考虑在创建文件系统时设置该属性。 |
| creation | 数字 | N/A | 只读属性，用于标识创建此数据集的日期和时间。 |
| devices | 布尔值 | on | 控制在文件系统中打开设备文件的能力。 |
| exec | 布尔值 | on | 控制是否允许执行此文件系统中的程序。另外，设置为 off 时，将不允许执行带有 PROT_EXEC 的 mmap(2) 调用。 |
| mounted | 布尔值 | N/A | 只读属性，用于指明此文件系统、克隆或快照当前是否已挂载。该属性不适用于卷。值可以是 yes 或 no。 |
| mountpoint | 字符串 | N/A | 控制用于此文件系统的挂载点。当文件系统的 mountpoint 属性发生更改时，将取消挂载该文件系统以及继承挂载点的任何子级。如果新值为 legacy，则该文件系统和子级将保持取消挂载状态。否则，如果属性以前为 legacy 或 none，或者该文件系统和子级在属性发生更改之前处于挂载状态，则会自动在新位置重新挂载它们。此外，任何共享文件系统都将取消共享，并在新位置进行共享。<br>有关使用该属性的更多信息，请参见第 89 页中的 “管理 ZFS 挂载点” 。 |

表 5–1 ZFS 本机属性说明 （续）

| 属性名 | 类型 | 缺省值 | 说明 |
| --- | --- | --- | --- |
| origin | 字符串 | N/A | 克隆的文件系统或卷的只读属性，用于标识创建克隆所在的快照。只要克隆存在，便不能销毁克隆源（即使使用 -r 或 -f 选项也是如此）。<br>非克隆的文件系统其 origin 为 none。 |
| quota | 数字（或 none） | none | 限制数据集及其后代可占用的空间量。该属性可对已使用的空间量强制实施硬限制，包括后代（含文件系统和快照）占用的所有空间。对已有配额的数据集的后代设置配额不会覆盖祖先的配额，但会施加额外的限制。不能对卷设置配额，因为 volsize 属性可用作隐式配额。<br>有关设置配额的信息，请参见第 95 页中的 “设置 ZFS 文件系统的配额”。 |
| readonly | 布尔值 | off | 控制是否可以修改此数据集。设置为 on 时，无法对数据集进行任何修改。<br>该属性也可通过其简短列名 rdonly 来引用。 |
| recordsize | 数字 | 128K | 为文件系统中的文件指定建议的块大小。<br>该属性也可通过其简短列名 recsize 来引用。有关详细说明，请参见第 81 页中的 “recordsize 属性”。 |
| referenced | 数字 | N/A | 只读属性，用于标识此数据集可访问的数据量，这些数据可能会也可能不会与池中的其他数据集共享。<br>创建快照或克隆时，首先会引用与创建该属性时所在的文件系统或快照相同的空间量，因为其内容相同。<br>该属性也可通过其简短列名 refer 来引用。 |
| reservation | 数字（或 none） | none | 为数据集及其后代预留的最小空间量。如果使用的空间量低于该值，则认为数据集正在使用其预留空间指定的空间量。父数据集的使用空间中会包含预留空间，并会针对父数据集的配额和预留空间对其进行计数。<br>该属性也可通过其简短列名 reserv 来引用。<br>有关更多信息，请参见第 96 页中的 “设置 ZFS 文件系统的预留空间”。 |
| setuid | 布尔值 | on | 控制文件系统中是否会标记 setuid 位。 |

表 5–1 ZFS 本机属性说明 （续）

| 属性名 | 类型 | 缺省值 | 说明 |
| --- | --- | --- | --- |
| sharenfs | 字符串 | off | 控制文件系统是否可用于 NFS 中以及使用的选项。如果设置为 on，则会调用不带任何选项的 zfs share 命令。否则，将调用带有与该属性的内容等效的选项的 zfs share 命令。如果设置为 off，则使用传统的 share 和 unshare 命令以及 dfstab 文件来管理文件系统。<br>有关共享 ZFS 文件系统的更多信息，请参见第 93 页中的 “共享和取消共享 ZFS 文件系统”。 |
| snapdir | 字符串 | hidden | 控制 .zfs 目录在文件系统根目录中是隐藏还是可见。有关使用快照的更多信息，请参见第 97 页中的 “ZFS 快照概述”。 |
| type | 字符串 | N/A | 只读属性，用于将数据集类型标识为 filesystem（文件系统或克隆）、volume 或 snapshot。 |
| used | 数字 | N/A | 只读属性，用于标识数据集及其所有后代占用的空间量。<br>有关详细说明，请参见第 79 页中的 “used 属性”。 |
| volsize | 数字 | N/A | 可为卷指定卷的逻辑大小。<br>有关详细说明，请参见第 81 页中的 “volsize 属性”。 |
| volblocksize | 数字 | 8 KB | 可为卷指定卷的块大小。一旦写入卷后，块大小便不能更改，因此应在创建卷时设置块大小。卷的缺省块大小为 8 KB。范围位于 512 字节到 128 KB 之间的 2 的任意次幂都有效。<br>该属性也可通过其简短列名 volblock 来引用。 |
| zoned | 布尔值 | N/A | 指明是否已将此数据集添加至非全局区域。如果设置该属性，全局区域中将不会标记挂载点，因此 ZFS 在收到请求时不能挂载此类文件系统。首次安装区域时，会为添加的所有文件系统设置该属性。<br>有关将 ZFS 用于已安装的区域的更多信息，请参见第 133 页中的 “在安装了区域的 Solaris 系统中使用 ZFS”。 |
| xattr | 布尔值 | on | 指示对此文件系统启用还是禁用扩展属性。缺省值为 on。 |

# ZFS 只读本机属性

只读本机属性是可以检索但不能设置的属性。只读本机属性不可继承。有些本机属性特定于特殊类型的数据集。在这种情况下，表 5–1 的说明部分会注明特殊的数据集类型。

下面列出了只读本机属性，表 5–1 对其进行了说明。

- available
- creation
- mounted
- origin
- compressratio
- referenced
- type
- used

  有关详细信息，请参见第 79 页中的 “used 属性”。

有关空间记帐（包括 used、referenced 和 available 属性）的更多信息，请参见第 32 页中的 “ZFS 空间记帐”。

### used 属性

此数据集及其所有后代占用的空间量。可根据此数据集的配额和预留空间来检查该值。使用的空间不包括数据集的预留空间，但会考虑任何后代数据集的预留空间。数据集占用其父级的空间量以及以递归方式销毁该数据集时所释放的空间量应为其使用空间和预留空间的较大者。

创建快照时，其空间最初在快照与文件系统之间进行共享，还可能是与以前的快照进行共享。随着文件系统的变化，以前共享的空间将供快照专用，并会计算在快照的使用空间内。此外，删除快照可增加其他快照专用（和使用）的空间量。有关快照和空间问题的更多信息，请参见第 32 页中的 “空间不足行为”。

使用的空间量、可用的空间量或引用的空间量不会考虑暂挂更改。通常，暂挂更改仅占用几秒钟的时间。使用 fsync(3c) 或 O_SYNC 提交对磁盘的更改，不一定可以保证空间使用情况信息会立即更新。

## 可设置的 ZFS 本机属性

可设置的本机属性是其值可同时进行检索和设置的属性。可设置的本机属性可以使用 zfs set 命令或 zfs create 命令进行设置，请分别参见第 85 页中的 “设置 ZFS 属性”和第 72 页中的 “创建 ZFS 文件系统”中的说明。除了配额和预留空间外，可设置的本机属性均可继承。有关配额和预留空间的更多信息，请参见第 95 页中的 “ZFS 配额和预留空间”。

有些可设置的本机属性特定于特殊类型的数据集。在这种情况下，表 5–1 的说明部分会注明特殊的数据集类型。如果未明确注明，则表明属性适用于所有数据集类型：文件系统、卷、克隆和快照。

下面列出了可设置的属性，表 5–1 对其进行了说明。

- aclinherit

  有关详细说明，请参见第 112 页中的 “ACL 属性模式”。
- aclmode

  有关详细说明，请参见第 112 页中的 “ACL 属性模式”。
- atime
- canmount
- checksum
- compression
- copies
- devices
- exec
- mountpoint
- quota
- readonly
- recordsize

  有关详细说明，请参见第 81 页中的 “recordsize 属性”。
- reservation
- sharenfs
- setuid
- snapdir
- volsize

  有关详细说明，请参见第 81 页中的 “volsize 属性”。
- volblocksize
- zoned

## canmount 属性

如果该属性设置为 off，则不能使用 zfs mount 或 zfs mount -a 命令挂载文件系统。该属性与将 mountpoint 属性设置为 none 的效果相似，区别在于数据集仍有一个可继承的正常 mountpoint 属性。例如，可将该属性设置为 off，为后代文件系统建立可继承属性，但文件系统本身永远不会挂载，也无法供用户访问。在这种情况下，该属性设置为 off 的父文件系统将充当一个**容器**，这样便可以在容器中设置属性，但容器本身永远不可访问。

在以下示例中，创建了 userpool 并将 canmount 属性设置为 off。将后代用户文件系统的挂载点设置为一个公共挂载点 /export/home。在父文件系统中设置的属性可由后代文件系统继承，但永远不会挂载父文件系统本身。

```
# zpool create userpool mirror c0t5d0 c1t6d0
# zfs set canmount=off userpool
# zfs set mountpoint=/export/home userpool
# zfs set compression=on userpool
# zfs create userpool/user1
# zfs create userpool/user2
# zfs list -r userpool
NAME             USED  AVAIL  REFER  MOUNTPOINT
userpool         140K  8.24G  24.5K  /export/home
userpool/user1  24.5K  8.24G  24.5K  /export/home/user1
userpool/user2  24.5K  8.24G  24.5K  /export/home/user2
```

### recordsize 属性

为文件系统中的文件指定建议的块大小。

该属性专门设计用于对大小固定的记录中的文件进行访问的数据库工作负荷。ZFS 会根据为典型的访问模式优化的内部算法来自动调整块大小。对于创建很大的文件但访问较小的随机块中的文件的数据库而言，这些算法可能不是最优的。将 `recordsize` 指定为大于或等于数据库的记录大小的值可以显著提高性能。强烈建议不要将该属性用于一般用途的文件系统，否则可能会对性能产生不利影响。指定的大小必须是 2 的若干次幂，并且必须大于或等于 512 字节同时小于或等于 128 KB。更改文件系统的 `recordsize` 仅影响之后创建的文件。现有文件不会受到影响。

该属性也可通过其简短列名 `recsize` 来引用。

### volsize 属性

卷的逻辑大小。缺省情况下，创建卷会产生相同大小的预留空间。对 `volsize` 的任何更改都会反映为对预留空间的等效更改。这些检查用来防止用户产生的意外行为。如果卷包含的空间比其声明可用的空间少，则会导致未定义的行为或数据损坏，具体取决于卷的使用方法。如果在卷的使用过程中更改卷大小，特别是在收缩大小时，也会出现上述影响。调整卷大小时，需要格外小心。

尽管并不建议，但可以通过为 `zfs create -V` 指定 `-s` 标志或通过在创建卷后即更改预留空间来创建稀疏卷。**稀疏卷**的定义是预留空间与卷大小不相等的卷。对于稀疏卷，预留空间中不会反映对 `volsize` 的更改。

有关使用卷的更多信息，请参见第 131 页中的 “ZFS 卷” 。

## ZFS 用户属性

除了标准的本机属性外，ZFS 还支持任意用户属性。用户属性对 ZFS 行为没有影响，但可通过用户环境中有意义的信息来注释数据集。

用户属性名必须符合以下特征：

- 包含冒号字符 (':')，以与本机属性相区分。
- 包含小写字母、数字和以下标点字符：':'、+、'.'、'_'。
- 用户属性名最多可以包含 256 个字符。

预期约定是属性名分为以下两个部分，但 ZFS 不强制使用此名称空间：

*module:property*

在程序中使用用户属性时，请对属性名的 *module* 部分使用反向 DNS 域名，以尽量避免两个独立开发的软件包将同一属性名用于不同用途。以 "com.sun." 开头的属性名保留供 Sun Microsystems 使用。

用户属性的值具有以下特征：

- 是始终继承且从不进行验证的任意字符串。
- 用户属性值最多可以包含 1024 个字符。

例如：

```
# zfs set dept:users=finance userpool/user1
# zfs set dept:users=general userpool/user2
# zfs set dept:users=itops userpool/user3
```

对属性执行操作的所有命令（如 `zfs list`、`zfs get`、`zfs set` 等）都可用来处理本机属性和用户属性。

例如：

```
zfs get -r dept:users userpool
NAME            PROPERTY    VALUE           SOURCE
userpool        dept:users  all             local
userpool/user1  dept:users  finance         local
userpool/user2  dept:users  general         local
userpool/user3  dept:users  itops           local
```

要清除某一用户属性，请使用 `zfs inherit` 命令。例如：

```
# zfs inherit -r dept:users userpool
```

如果任意父数据集中均未定义该属性，则会将其完全删除。

# 查询 ZFS 文件系统信息

zfs list 命令提供了一种用于查看和查询数据集信息的可扩展机制。本节中对基本查询和复杂查询都进行了说明。

## 列出基本 ZFS 信息

通过使用不带任何选项的 zfs list 命令可以列出基本数据集信息。此命令可显示系统中所有数据集的名称，包括其 used、available、referenced 和 mountpoint 属性。有关这些属性的更多信息，请参见第 74 页中的 “ZFS 属性介绍”。

例如：

```
# zfs list
NAME                   USED  AVAIL  REFER  MOUNTPOINT
pool                   476K  16.5G    21K  /pool
pool/clone              18K  16.5G    18K  /pool/clone
pool/home              296K  16.5G    19K  /pool/home
pool/home/marks        277K  16.5G   277K  /pool/home/marks
pool/home/marks@snap      0      -   277K  -
pool/test               18K  16.5G    18K  /test
```

另外，还可使用此命令通过在命令行中提供数据集名称来显示特定数据集。此外，使用 -r 选项将以递归方式显示该数据集的所有后代。例如：

```
# zfs list -r pool/home/marks
NAME                   USED  AVAIL  REFER  MOUNTPOINT
pool/home/marks        277K  16.5G   277K  /pool/home/marks
pool/home/marks@snap      0      -   277K  -
```

可将 zfs list 命令与数据集、快照和卷的绝对路径名结合使用。例如：

```
# zfs list /pool/home/marks
NAME              USED  AVAIL  REFER  MOUNTPOINT
pool/home/marks   277K  16.5G   277K  /pool/home/marks
```

以下示例说明如何显示 tank/home/chua 及其所有后代数据集。

```
# zfs list -r tank/home/chua
NAME                       USED  AVAIL  REFER  MOUNTPOINT
tank/home/chua                26.0K  4.81G  10.0K  /tank/home/chua
tank/home/chua/projects       16K  4.81G   9.0K  /tank/home/chua/projects
tank/home/chua/projects/fs1    8K  4.81G     8K  /tank/home/chua/projects/fs1
tank/home/chua/projects/fs2    8K  4.81G     8K  /tank/home/chua/projects/fs2
```

有关 zfs list 命令的其他信息，请参见 zfs(1M)。

# 创建复杂的 ZFS 查询

通过使用 -o、-f 和 -H 选项可对 zfs list 输出进行自定义。

通过使用 -o 选项以及所需属性的逗号分隔列表可以自定义属性值输出。可将任何数据集属性作为有效值提供。有关所有受支持的数据集属性的列表，请参见第 74 页中的“ZFS 属性介绍”。除了其中定义的属性外，-o 选项列表还可以包含字符 name，以指明输出应包括数据集的名称。

以下示例使用 zfs list 来显示数据集名称以及 sharenfs 和 mountpoint 属性。

```
# zfs list -o name,sharenfs,mountpoint
NAME                  SHARENFS         MOUNTPOINT
tank                  off              /tank
tank/home             on               /tank/home
tank/home/ahrens      on               /tank/home/ahrens
tank/home/bonwick     on               /tank/home/bonwick
tank/home/chua        on               /tank/home/chua
tank/home/eschrock    on               legacy
tank/home/moore       on               /tank/home/moore
tank/home/tabriz      ro               /tank/home/tabriz
```

可以使用 -t 选项指定要显示的数据集的类型。下表中介绍了有效的类型。

**表 5–2** ZFS 数据集的类型

| 类型 | 说明 |
|---|---|
| filesystem | 文件系统和克隆 |
| volume | 卷 |
| snapshot | 快照 |

-t 选项可后跟要显示的数据集类型的逗号分隔列表。以下示例同时使用 -t 和 -o 选项来显示所有文件系统的名称和 used 属性：

```
# zfs list -t filesystem -o name,used
NAME              USED
pool              476K
pool/clone         18K
pool/home         296K
pool/home/marks   277K
pool/test          18K
```

使用 -H 选项可从生成的输出中省略 zfs list 标题。使用 -H 选项，所有空格都以制表符形式输出。当需要可解析的输出（例如编写脚本时），此选项可能很有用。以下示例显示了使用带有 -H 选项的 zfs list 命令所生成的输出：

```
# zfs list -H -o name
pool
pool/clone
pool/home
pool/home/marks
pool/home/marks@snap
pool/test
```

# 管理 ZFS 属性

数据集属性通过 zfs 命令的 set、inherit 和 get 子命令来管理。

- 第 85 页中的 “设置 ZFS 属性”
- 第 86 页中的 “继承 ZFS 属性”
- 第 86 页中的 “查询 ZFS 属性”

## 设置 ZFS 属性

可以使用 zfs set 命令修改任何可设置的数据集属性。或者，也可以使用 zfs create 命令在创建数据集时设置属性。有关可设置的数据集属性的列表，请参见第 79 页中的 “可设置的 ZFS 本机属性” 。zfs set 命令采用 *property=value* 格式的属性/值序列和数据集名称。

以下示例将 tank/home 的 atime 属性设置为 off。在每个 zfs set 调用过程中，只能设置或修改一个属性。

```
# zfs set atime=off tank/home
```

此外，任何文件系统属性均可在创建文件系统时设置。例如：

```
# zfs create -o atime=off tank/home
```

通过使用以下易于理解的后缀（按量值的顺序）可以指定数字属性：BKMGTPEZ。其中任一后缀都可后跟可选的 b，用于表示字节，但 B 后缀除外，因为它已表示了字节。以下四个 zfs set 调用是等效的数字表达式，指明在 tank/home/marks 文件系统中将 quota 属性设置为值 50 GB：

```
# zfs set quota=50G tank/home/marks
# zfs set quota=50g tank/home/marks
# zfs set quota=50GB tank/home/marks
# zfs set quota=50gb tank/home/marks
```

非数字属性的值区分大小写，并且必须为小写，但 mountpoint 和 sharenfs 除外。这两个属性的值既可以包含大写字母，也可以包含小写字母。

有关 zfs set 命令的更多信息，请参见 zfs(1M)。

# 继承 ZFS 属性

除非已对属性子级显式设置了配额或预留空间，否则除了配额和预留空间外，所有可设置的属性都从父级继承各自的值。如果没有祖先为继承的属性设置显式值，则使用该属性的缺省值。可以使用 `zfs inherit` 命令清除属性设置，从而导致从父级继承设置。

以下示例使用 `zfs set` 命令为 `tank/home/bonwick` 文件系统启用压缩。然后，使用 `zfs inherit` 取消设置 `compression` 属性，从而使该属性继承缺省设置 `off`。由于 `home` 和 `tank` 都未本地设置 `compression` 属性，因此会使用缺省值。如果两者都启用了压缩，则使用最直接的祖先中设置的值（在本示例中为 `home`）。

```
# zfs set compression=on tank/home/bonwick
# zfs get -r compression tank
NAME             PROPERTY      VALUE                  SOURCE
tank             compression   off                    default
tank/home        compression   off                    default
tank/home/bonwick compression   on                     local
# zfs inherit compression tank/home/bonwick
# zfs get -r compression tank
NAME             PROPERTY      VALUE                  SOURCE
tank             compression   off                    default
tank/home        compression   off                    default
tank/home/bonwick compression  off                    default
```

如果指定了 `-r` 选项，则会以递归方式应用 `inherit` 子命令。在以下示例中，该命令将使 `tank/home` 以及它可能具有的所有后代都继承 `compression` 属性的值。

```
# zfs inherit -r compression tank/home
```

**注 –** 请注意，使用 `-r` 选项会清除所有后代数据集的当前属性设置。

有关 `zfs` 命令的更多信息，请参见 `zfs(1M)`。

# 查询 ZFS 属性

查询属性值的最简单方法是使用 `zfs list` 命令。有关更多信息，请参见第 83 页中的 “列出基本 ZFS 信息”。但是，对于复杂查询和脚本编写，请使用 `zfs get` 命令以自定义格式提供更详细的信息。

可以使用 `zfs get` 命令检索任何数据集属性。以下示例说明如何在数据集中检索单个属性。

```
# zfs get checksum tank/ws
NAME            PROPERTY       VALUE                  SOURCE
tank/ws         checksum       on                     default
```

第四列 SOURCE 指明所设置的该属性值的源。下表定义了可能的源值的含义。

**表 5–3** 可能的 SOURCE 值 (zfs get)

| 源值 | 说明 |
|---|---|
| default | 从来不为数据集或其任何祖先显式设置该属性。使用的是该属性的缺省值。 |
| inherited from *dataset-name* | 该属性值继承自 *dataset-name* 所指定的父级。 |
| local | 使用 zfs set 可为此数据集显式设置该属性值。 |
| temporary | 该属性值是使用 zfs mount -o 选项设置的，并且仅在挂载的生命周期内有效。有关临时挂载点属性的更多信息，请参见第 92 页中的 “使用临时挂载属性”。 |
| -（无） | 该属性是只读属性。其值由 ZFS 生成。 |

可以使用特殊关键字 all 检索所有数据集属性。以下示例使用 all 关键字来检索所有现有的数据集属性：

```
# zfs get all tank
NAME  PROPERTY       VALUE                  SOURCE
tank  type           filesystem             -
tank  creation       Thu Feb 21 14:14 2008  -
tank  used           88K                    -
tank  available      49.5G                  -
tank  referenced     24.5K                  -
tank  compressratio  1.00x                  -
tank  mounted        yes                    -
tank  quota          none                   default
tank  reservation    none                   default
tank  recordsize     128K                   default
tank  mountpoint     /tank                  default
tank  sharenfs       off                    default
tank  checksum       on                     default
tank  compression    off                    default
tank  atime          on                     default
tank  devices        on                     default
tank  exec           on                     default
tank  setuid         on                     default
tank  readonly       off                    default
tank  zoned          off                    default
tank  snapdir        hidden                 default
tank  aclmode        groupmask              default
```

```
tank  aclinherit     secure                default
tank  canmount       on                    default
tank  shareiscsi     off                   default
tank  xattr          on                    default
```

通过 zfs get 的 -s 选项，可以按源值指定要显示的属性的类型。通过此选项可获取一个逗号分隔列表，用于指明所需的源类型。仅会显示具有指定源类型的属性。有效的源类型包括 local、default、inherited、temporary 和 none。以下示例显示了已对 pool 本地设置的所有属性。

```
# zfs get -s local all pool
NAME             PROPERTY      VALUE                     SOURCE
pool             compression   on                        local
```

以上任何选项均可与 -r 选项结合使用，以便以递归方式显示指定数据集的所有子级的指定属性。在以下示例中，以递归方式显示了 tank 中所有数据集的所有临时属性：

```
# zfs get -r -s temporary all tank
NAME             PROPERTY       VALUE                     SOURCE
tank/home          atime          off                     temporary
tank/home/bonwick  atime          off                     temporary
tank/home/marks    atime          off                     temporary
```

通过一项最新功能可以使用 zfs get 命令创建查询而无需指定目标文件系统，这意味着该命令作用于所有池或文件系统。例如：

```
# zfs get -s local all
tank/home               atime          off                   local
tank/home/bonwick       atime          off                   local
tank/home/marks         quota          50G                   local
```

有关 zfs get 命令的更多信息，请参见 zfs(1M)。

## 查询用于编写脚本的 ZFS 属性

zfs get 命令支持为编写脚本而设计的 -H 和 -o 选项。-H 选项指明应忽略所有标题信息，并且所有空格都显示为制表符形式。使用一致的空格可使数据便于分析。您可以使用 -o 选项自定义输出。通过此选项可获取要输出的值的逗号分隔列表。-o 列表中可提供第 74 页中的 “ZFS 属性介绍”中定义的所有属性，以及字符 name、value、property 和 source。

以下示例说明如何使用 zfs get 的 -H 和 -o 选项来检索单个值。

```
# zfs get -H -o value compression tank/home
on
```

-p 选项会将数字值报告为精确值。例如，1 MB 可能报告为 1000000。此选项可以按如下方式使用：

```
# zfs get -H -o value -p used tank/home
182983742
```

可以结合使用 -r 选项与以上任何选项，以递归方式为所有后代检索请求值。以下示例使用 -r、-o 和 -H 选项为 export/home 及其后代检索数据集名称和 used 属性值，同时忽略所有标题输出：

```
# zfs get -H -o name,value -r used export/home
export/home     5.57G
export/home/marks       1.43G
export/home/maybee      2.15G
```

# 挂载和共享 ZFS 文件系统

本节介绍如何在 ZFS 中管理挂载点和共享的文件系统。



## 管理 ZFS 挂载点

缺省情况下，所有 ZFS 文件系统都由 ZFS 通过使用 SMF 的 svc://system/filesystem/local 服务在引导时挂载。文件系统挂载在 */path* 下，其中 *path* 是文件系统的名称。

通过使用 zfs set 命令将 mountpoint 属性设置为特定路径，可以覆盖缺省挂载点。如果需要，ZFS 会自动创建此挂载点，并在调用 zfs mount -a 命令时自动挂载此文件系统，而无需编辑 /etc/vfstab 文件。

mountpoint 属性是继承的。例如，如果 pool/home 将 mountpoint 设置为 /export/stuff，则 pool/home/user 将继承 /export/stuff/user 的 mountpoint 属性。

可将 mountpoint 属性设置为 none，以防止挂载文件系统。此外，canmount 属性可用来确定某一文件系统是否可以挂载。有关 canmount 属性的更多信息，请参见第 80 页中的 “canmount 属性”。

如果需要，还可以使用 zfs set 将 mountpoint 属性设置为 legacy，从而通过传统挂载接口来显式管理文件系统。这样做可以防止 ZFS 自动挂载和管理此文件系统。不过必须改用包括 mount 和 umount 命令在内的传统工具以及 /etc/vfstab 文件。有关传统挂载的更多信息，请参见第 90 页中的 “传统挂载点”。

更改挂载点管理策略时，会应用以下行为：

- 自动挂载点行为
- 传统挂载点行为

## 自动挂载点

- 从 legacy 或 none 进行更改时，ZFS 将自动挂载文件系统。
- 如果 ZFS 当前正在管理文件系统，但该文件系统当前已取消挂载，并且 mountpoint 属性已更改，则文件系统将保持取消挂载状态。

另外，也可以在创建时使用 zpool create 的 -m 选项设置根数据集的缺省安装点。有关创建池的更多信息，请参见第 40 页中的 “创建 ZFS 存储池”。

mountpoint 属性不为 legacy 的所有数据集都由 ZFS 来管理。在以下示例中，创建了一个数据集，其挂载点由 ZFS 自动管理。

```
# zfs create pool/filesystem
# zfs get mountpoint pool/filesystem
NAME              PROPERTY      VALUE                      SOURCE
pool/filesystem   mountpoint    /pool/filesystem           default
# zfs get mounted pool/filesystem
NAME              PROPERTY      VALUE                      SOURCE
pool/filesystem   mounted       yes                        -
```

另外，也可按以下示例所示，显式设置 mountpoint 属性：

```
# zfs set mountpoint=/mnt pool/filesystem
# zfs get mountpoint pool/filesystem
NAME              PROPERTY      VALUE                      SOURCE
pool/filesystem   mountpoint    /mnt                       local
# zfs get mounted pool/filesystem
NAME              PROPERTY      VALUE                      SOURCE
pool/filesystem   mounted       yes                        -
```

mountpoint 属性更改时，文件系统将自动从旧挂载点取消挂载，并重新挂载到新挂载点。根据需要，可创建挂载点目录。如果 ZFS 由于处于活动状态而无法取消挂载文件系统，则会报告错误，并需要强制进行手动取消挂载。

## 传统挂载点

通过将 mountpoint 属性设置为 legacy，可以使用传统工具来管理 ZFS 文件系统。传统文件系统必须通过 mount 和 umount 命令以及 /etc/vfstab 文件来管理。ZFS 在引导时不会自动挂载传统文件系统，并且 ZFS mount 和 umount 命令不会对此类型的数据集执行操作。以下示例说明如何在传统模式下设置和管理 ZFS 数据集：

```
# zfs set mountpoint=legacy tank/home/eschrock
# mount -F zfs tank/home/eschrock /mnt
```

此外，还必须通过在 /etc/vfstab 文件中创建相应的项来挂载这些文件系统。否则，system/filesystem/local 服务在系统引导时将进入维护模式。

要在引导时自动挂载传统文件系统，必须向 /etc/vfstab 文件中添加一项。以下示例说明 /etc/vfstab 文件中的项的可能显示情况：

```
#device         device        mount           FS      fsck    mount   mount
#to mount       to fsck       point           type    pass    at boot options
#

tank/home/eschrock -        /mnt           zfs        -        yes        -
```

请注意，device to fsck 和 fsck pass 项设置为 -。使用此语法是因为 fsck 命令不适用于 ZFS 文件系统。有关 ZFS 中的数据完整性以及不需要 fsck 的更多信息，请参见第 21 页中的 “事务性语义”。

# 挂载 ZFS 文件系统

创建文件系统或系统引导时，ZFS 会自动挂载文件系统。仅当更改挂载点或显式挂载或取消挂载文件系统时，才需要使用 zfs mount 命令。

不带任何参数的 zfs mount 命令可以显示 ZFS 管理的当前已挂载的所有文件系统。传统管理的挂载点不会显示。例如：

```
# zfs mount
tank                            /tank
tank/home                       /tank/home
tank/home/bonwick               /tank/home/bonwick
tank/ws                         /tank/ws
```

可以使用 -a 选项挂载 ZFS 管理的所有文件系统。传统管理的文件系统不会挂载。例如：

```
# zfs mount -a
```

缺省情况下，ZFS 不允许在非空目录的顶层进行挂载。要强制在非空目录的顶层挂载，必须使用 -O 选项。例如：

```
# zfs mount tank/home/lalt
cannot mount '/export/home/lalt': directory is not empty
use legacy mountpoint to allow this behavior, or use the -O flag
# zfs mount -O tank/home/lalt
```

传统挂载点必须通过传统工具进行管理。尝试使用 ZFS 工具将产生错误。例如：

```
# zfs mount pool/home/billm
cannot mount 'pool/home/billm': legacy mountpoint
use mount(1M) to mount this filesystem
# mount -F zfs tank/home/billm
```

文件系统挂载时，将基于与数据集关联的属性值使用一组挂载选项。属性与挂载选项之间的相互关系如下：

| 属性 | 挂载选项 |
| --- | --- |
| devices | devices/nodevices |
| exec | exec/noexec |
| readonly | ro/rw |
| setuid | setuid/nosetuid |

挂载选项 nosuid 是 nodevices 和 nosetuid 的别名。

# 使用临时挂载属性

如果使用带有 -o 选项的 zfs mount 命令显式设置了以上任何选项，则会临时覆盖关联的属性值。zfs get 命令将这些属性值报告为 temporary，并在文件系统取消挂载时恢复为其初始设置。如果挂载数据集时更改某个属性值，更改将立即生效，并覆盖所有临时设置。

在以下示例中，对 tank/home/perrin 文件系统临时设置了只读挂载选项：

```
# zfs mount -o ro tank/home/perrin
```

在本示例中，假定文件系统已取消挂载。要临时更改当前已挂载的文件系统的属性，必须使用特殊的 remount 选项。在以下示例中，对于当前挂载的文件系统，atime 属性暂时更改为 off：

```
# zfs mount -o remount,noatime tank/home/perrin
# zfs get atime tank/home/perrin
NAME              PROPERTY       VALUE                      SOURCE
tank/home/perrin atime          off                        temporary
```

有关 zfs mount 命令的更多信息，请参见 zfs(1M)。

# 取消挂载 ZFS 文件系统

通过使用 zfs unmount 子命令可以取消挂载文件系统。unmount 命令可以采用挂载点或文件系统名作为参数。

在以下示例中，按文件系统名称取消挂载了一个文件系统：

```
# zfs unmount tank/home/tabriz
```

在以下示例中，按挂载点取消挂载了一个文件系统：

```
# zfs unmount /export/home/tabriz
```

如果文件系统处于活动或繁忙状态，则 unmount 命令将失败。要强制性取消挂载文件系统，可以使用 -f 选项。如果文件系统内容正处于使用状态，则强制性取消挂载该文件系统时请务必小心。否则，会产生不可预测的应用程序行为。

```
# zfs unmount tank/home/eschrock
cannot unmount '/export/home/eschrock': Device busy
# zfs unmount -f tank/home/eschrock
```

要提供向后兼容性，可以使用传统的 umount 命令来取消挂载 ZFS 文件系统。例如：

```
# umount /export/home/bob
```

有关 zfs umount 命令的更多信息，请参见 zfs(1M)。

# 共享和取消共享 ZFS 文件系统

与挂载点相似，ZFS 可以通过使用 sharenfs 属性来自动共享文件系统。如果使用此方法，则不必在添加新文件系统时修改 /etc/dfs/dfstab 文件。sharenfs 属性是要传递给 share 命令的选项的逗号分隔列表。特殊值 on 是缺省的共享选项的别名，这些选项是所有用户的 read/write 权限。特殊值 off 指明文件系统不是由 ZFS 进行管理，但可通过传统方法（如 /etc/dfs/dfstab 文件）来共享。在引导过程中将共享 sharenfs 属性不是 off 的所有文件系统。

## 控制共享语义

缺省情况下，所有文件系统都未进行共享。要共享新文件系统，请使用类似如下的 zfs set 语法：

```
# zfs set sharenfs=on tank/home/eschrock
```

该属性是继承的，如果文件系统继承的属性不为 off，则这些文件系统在创建时会自动进行共享。例如：

```
# zfs set sharenfs=on tank/home
# zfs create tank/home/bricker
# zfs create tank/home/tabriz
# zfs set sharenfs=ro tank/home/tabriz
```

tank/home/bricker 和 tank/home/tabriz 最初以可写方式共享，因为它们从 tank/home 继承了 sharenfs 属性。一旦该属性设置为 ro（只读），则无论为 tank/home 设置的 sharenfs 属性为何值，都会以只读方式共享 tank/home/tabriz。

## 取消共享 ZFS 文件系统

尽管大多数文件系统都可在引导、创建和销毁过程中自动共享和取消共享，但文件系统有时候需要显式取消共享。为此，请使用 zfs unshare 命令。例如：

```
# zfs unshare tank/home/tabriz
```

此命令会取消共享 tank/home/tabriz 文件系统。要取消共享系统中的所有 ZFS 文件系统，需要使用 -a 选项。

```
# zfs unshare -a
```

## 共享 ZFS 文件系统

在引导和创建过程中共享的 ZFS 的自动行为在大部分时间内对于正常操作而言是足够的。如果由于某些原因取消共享了某个文件系统，则可使用 zfs share 命令再次将其共享。例如：

```
# zfs share tank/home/tabriz
```

另外，还可以通过使用 -a 选项共享系统中的所有 ZFS 文件系统。

```
# zfs share -a
```

## 传统共享行为

如果 sharenfs 属性为 off，则 ZFS 在任何时候都不会尝试共享或取消共享文件系统。借助此设置，可以通过传统方法（如 /etc/dfs/dfstab 文件）来进行管理。

与传统的 mount 命令不同，传统的 share 和 unshare 命令在 ZFS 文件系统中仍然可以运行。因此，可以使用与 sharenfs 属性的设置不同的选项来手动共享文件系统。不鼓励使用这种管理模型。请选择完全通过 ZFS 或完全通过 /etc/dfs/dfstab 文件来管理 NFS 共享内容。ZFS 管理模型与传统模型相比，设计更为简单，所需进行的工作越少。但是，在某些情况下，可能仍然需要通过熟悉的模型来控制文件系统的共享行为。

# ZFS 配额和预留空间

ZFS 支持文件系统级别的配额和预留空间。可以使用 quota 属性对文件系统可以使用的空间量设置限制。此外，还可以使用 reservation 属性来保证一定的空间量可供文件系统使用。这两个属性将应用于设置了它们的数据集以及该数据集的所有后代。

也就是说，如果对 tank/home 数据集设置了配额，则 tank/home **及其所有后代**使用的总空间量不能超过该配额。同样，如果为 tank/home 指定了预留空间，则 tank/home **及其所有后代**都会使用该预留空间。数据集及其所有后代使用的空间量由 used 属性报告。

有关更多信息，请参见下面的示例。

## 设置 ZFS 文件系统的配额

通过使用 zfs set 和 zfs get 命令可以设置和显示 ZFS 配额。在以下示例中，对 tank/home/bonwick 设置了 10 GB 的配额。

```
# zfs set quota=10G tank/home/bonwick
# zfs get quota tank/home/bonwick
NAME              PROPERTY      VALUE                      SOURCE
tank/home/bonwick quota         10.0G                      local
```

ZFS 配额还会影响 zfs list 和 df 命令的输出。例如：

```
# zfs list
NAME                   USED  AVAIL  REFER  MOUNTPOINT
tank/home             16.5K  33.5G  8.50K  /export/home
tank/home/bonwick     15.0K  10.0G  8.50K  /export/home/bonwick
tank/home/bonwick/ws  6.50K  10.0G  8.50K  /export/home/bonwick/ws
# df -h /export/home/bonwick
Filesystem             size   used  avail capacity  Mounted on
tank/home/bonwick       10G     8K    10G     1%    /export/home/bonwick
```

请注意，虽然 tank/home 具有 33.5 GB 的可用空间，但由于 tank/home/bonwick 存在配额，tank/home/bonwick 和 tank/home/bonwick/ws 仅有 10 GB 的可用空间。

不能将配额设置为比数据集当前使用的空间小的数量。例如：

```
# zfs set quota=10K tank/home/bonwick
cannot set quota for 'tank/home/bonwick': size is less than current used or
reserved space
```

# 设置 ZFS 文件系统的预留空间

ZFS **预留空间**是从池中分配的保证可供数据集使用的空间。因此，如果空间当前在池中不可用，则不能为数据集预留该空间。所有未占用的预留空间的总量不能超出池中未使用的空间量。通过使用 `zfs set` 和 `zfs get` 命令可以设置和显示 ZFS 预留空间。例如：

```
# zfs set reservation=5G tank/home/moore
# zfs get reservation tank/home/moore
NAME             PROPERTY      VALUE                      SOURCE
tank/home/moore  reservation   5.00G                      local
```

ZFS 预留空间会影响 `zfs list` 命令的输出。例如：

```
# zfs list
NAME                   USED  AVAIL  REFER  MOUNTPOINT
tank/home             5.00G  33.5G  8.50K  /export/home
tank/home/moore       15.0K  10.0G  8.50K  /export/home/moore
```

请注意，`tank/home` 使用的空间为 5 GB，但 `tank/home` 及其后代涉及的总空间量远远小于 5 GB。使用的空间反映了为 `tank/home/moore` 预留的空间。在父数据集的已用空间中会考虑预留空间，并会针对其配额、预留空间或同时针对两者进行计数。

```
# zfs set quota=5G pool/filesystem
# zfs set reservation=10G pool/filesystem/user1
cannot set reservation for 'pool/filesystem/user1': size is greater than
available space
```

只要未预留的池中有可用空间，并且数据集的当前使用率低于其配额，数据集即可使用比其预留空间更多的空间。数据集不能占用为其他数据集预留的空间。

预留空间无法累积。也就是说，第二次调用 `zfs set` 来设置预留空间时，不会将该数据集的预留空间添加到现有预留空间中，而是使用第二个预留空间替换第一个预留空间。

```
# zfs set reservation=10G tank/home/moore
# zfs set reservation=5G tank/home/moore
# zfs get reservation tank/home/moore
NAME             PROPERTY      VALUE                      SOURCE
tank/home/moore  reservation   5.00G                      local
```

◆ ◆ ◆ 第 6 章

# 使用 ZFS 快照和克隆

本章介绍如何创建和管理 ZFS 快照和克隆。本章还介绍有关保存快照的信息。

本章包含以下各节：



## ZFS 快照概述

**快照**是文件系统或卷的只读副本。快照几乎可以即时创建，而且最初不占用池中的其他磁盘空间。但是，当活动数据集中的数据发生更改时，快照通过继续引用旧数据占用磁盘空间，从而阻止释放该空间。

ZFS 快照具有以下特征：

- 可在系统重新引导后存留下来。
- 理论最大快照数是 $2^{64}$。
- 不使用单独的后备存储。快照直接占用存储池（从中创建这些快照的文件系统所在的存储池）中的磁盘空间。
- 递归快照可作为一个原子操作快速创建。要么一起创建快照（一次创建所有快照），要么不创建任何快照。原子快照操作的优点是始终在一个一致的时间捕获快照数据，即使跨后代文件系统也是如此。

无法直接访问卷的快照，但是可以对它们执行克隆、备份、回滚等操作。有关备份 ZFS 快照的信息，请参见第 103 页中的 “保存和恢复 ZFS 数据”。

# 创建和销毁 ZFS 快照

快照是使用 `zfs snapshot` 命令创建的，该命令将要创建的快照的名称用作其唯一参数。快照名称按如下方式指定：

*filesystem@snapname*
*volume@snapname*

快照名称必须符合第 24 页中的 “ZFS 组件命名要求”中定义的命名约定。

在以下示例中，将创建 `tank/home/ahrens` 的快照，其名称为 `friday`。

```
# zfs snapshot tank/home/ahrens@friday
```

通过使用 `-r` 选项可为所有后代文件系统创建快照。例如：

```
# zfs snapshot -r tank/home@now
# zfs list -t snapshot
NAME                    USED  AVAIL  REFER  MOUNTPOINT
tank/home@now             0      -  29.5K  -
tank/home/ahrens@now      0      -  2.15M  -
tank/home/anne@now        0      -  1.89M  -
tank/home/bob@now         0      -  1.89M  -
tank/home/cindys@now      0      -  2.15M  -
```

快照没有可修改的属性。也不能将数据集属性应用于快照。

```
# zfs set compression=on tank/home/ahrens@tuesday
cannot set compression property for 'tank/home/ahrens@tuesday': snapshot
properties cannot be modified
```

使用 `zfs destroy` 命令可以销毁快照。例如：

```
# zfs destroy tank/home/ahrens@friday
```

如果数据集存在快照，则不能销毁该数据集。例如：

```
# zfs destroy tank/home/ahrens
cannot destroy 'tank/home/ahrens': filesystem has children
use '-r' to destroy the following datasets:
tank/home/ahrens@tuesday
tank/home/ahrens@wednesday
tank/home/ahrens@thursday
```

此外，如果已从快照创建克隆，则必须先销毁克隆，才能销毁快照。

有关 `destroy` 子命令的更多信息，请参见第 72 页中的 “销毁 ZFS 文件系统”。

## 重命名 ZFS 快照

可以重命名快照，但是必须在从中创建它们的池和数据集中对它们进行重命名。例如：

```
# zfs rename tank/home/cindys@083006 tank/home/cindys@today
```

此外，下面的快捷语法提供了与上例等效的快照重命名语法。

```
# zfs rename tank/home/cindys@083006 today
```

不支持以下快照重命名操作，因为目标池和文件系统名称与从中创建快照的池和文件系统不同。

```
# zfs rename tank/home/cindys@today pool/home/cindys@saturday
cannot rename to 'pool/home/cindys@today': snapshots must be part of same
dataset
```

可以使用 `zfs rename -r` 命令以递归方式重命名快照。例如：

```
# zfs list
NAME                         USED  AVAIL  REFER  MOUNTPOINT
users                        270K  16.5G    22K  /users
users/home                    76K  16.5G    22K  /users/home
users/home@yesterday            0      -    22K  -
users/home/markm              18K  16.5G    18K  /users/home/markm
users/home/markm@yesterday      0      -    18K  -
users/home/marks              18K  16.5G    18K  /users/home/marks
users/home/marks@yesterday      0      -    18K  -
users/home/neil               18K  16.5G    18K  /users/home/neil
users/home/neil@yesterday       0      -    18K  -
# zfs rename -r users/home@yesterday @2daysago
# zfs list -r users/home
NAME                        USED  AVAIL  REFER  MOUNTPOINT
users/home                   76K  16.5G    22K  /users/home
users/home@2daysago            0      -    22K  -
users/home/markm             18K  16.5G    18K  /users/home/markm
users/home/markm@2daysago      0      -    18K  -
users/home/marks             18K  16.5G    18K  /users/home/marks
users/home/marks@2daysago      0      -    18K  -
users/home/neil              18K  16.5G    18K  /users/home/neil
users/home/neil@2daysago       0      -    18K  -
```

# 显示和访问 ZFS 快照

在包含文件系统的根的 .zfs/snapshot 目录中，可以访问文件系统的快照。例如，如果在 /home/ahrens 上挂载了 tank/home/ahrens，则在 /home/ahrens/.zfs/snapshot/thursday 目录中可以访问 tank/home/ahrens@thursday 快照数据。

```
# ls /tank/home/ahrens/.zfs/snapshot
tuesday wednesday thursday
```

可以列出快照，如下所示：

```
# zfs list -t snapshot
NAME                           USED  AVAIL  REFER  MOUNTPOINT
pool/home/anne@monday             0      -   780K  -
pool/home/bob@monday              0      -  1.01M  -
tank/home/ahrens@tuesday       8.50K      -   780K  -
tank/home/ahrens@wednesday     8.50K      -  1.01M  -
tank/home/ahrens@thursday         0      -  1.77M  -
tank/home/cindys@today         8.50K      -   524K  -
```

可以列出为特定文件系统创建的快照，如下所示：

```
# zfs list -r -t snapshot -o name,creation tank/home
NAME                          CREATION
tank/home@now                  Wed Aug 30 10:53 2006
tank/home/ahrens@tuesday       Wed Aug 30 10:53 2006
tank/home/ahrens@wednesday     Wed Aug 30 10:54 2006
tank/home/ahrens@thursday      Wed Aug 30 10:53 2006
tank/home/cindys@now           Wed Aug 30 10:57 2006
```

## 快照空间记帐

创建快照时，最初在快照和文件系统之间共享其空间，还可能与以前的快照共享其空间。在文件系统发生更改时，以前共享的空间将变为该快照专用的空间，因此会将该空间算入快照的 used 属性。此外，删除快照可增加其他快照专用（**使用**）的空间量。

创建快照时，快照的空间 referenced 属性与文件系统的相同。

# 回滚到 ZFS 快照

可以使用 zfs rollback 命令废弃自创建特定快照之后所做的所有更改。文件系统恢复到创建快照时的状态。缺省情况下，该命令无法回滚到除最新快照以外的快照。

要回滚到早期快照，必须销毁所有的中间快照。可以通过指定 -r 选项销毁早期的快照。

如果存在任何中间快照的克隆，则还必须指定 -R 选项以销毁克隆。

**注 –** 如果要回滚的文件系统当前为挂载状态，则必须取消挂载再重新挂载。如果无法取消挂载该文件系统，则回滚将失败。-f 选项可强制取消挂载文件系统（如有必要）。

在以下示例中，会将 tank/home/ahrens 文件系统回滚到 tuesday 快照：

```
# zfs rollback tank/home/ahrens@tuesday
cannot rollback to 'tank/home/ahrens@tuesday': more recent snapshots exist
use '-r' to force deletion of the following snapshots:
tank/home/ahrens@wednesday
tank/home/ahrens@thursday
# zfs rollback -r tank/home/ahrens@tuesday
```

在上面的示例中，因为已回滚到以前的 tuesday 快照，所以删除了 wednesday 和 thursday 快照。

```
# zfs list -r -t snapshot -o name,creation tank/home/ahrens
NAME                       CREATION
tank/home/ahrens@tuesday  Wed Aug 30 10:53 2006
```

# ZFS 克隆概述

**克隆**是可写入的卷或文件系统，其初始内容与从中创建它的数据集的内容相同。与快照一样，创建克隆几乎是即时的，而且最初不占用其他磁盘空间。此外，还可以创建克隆的快照。

- 第 102 页中的 “创建 ZFS 克隆”
- 第 102 页中的 “销毁 ZFS 克隆”
- 第 102 页中的 “使用 ZFS 克隆替换 ZFS 文件系统”

克隆只能从快照创建。克隆快照时，会在克隆和快照之间建立隐式相关性。即使克隆是在数据集分层结构中的某个其他位置创建的，但只要克隆存在，就无法销毁原始快照。origin 属性显示此相关性，而 zfs destroy 命令会列出任何此类相关性（如果存在）。

克隆不继承从其中创建它的数据集的属性。使用 zfs get 和 zfs set 命令，可以查看和更改克隆数据集的属性。有关设置 ZFS 数据集属性的更多信息，请参见第 85 页中的 “设置 ZFS 属性”。

由于克隆最初与原始快照共享其所有磁盘空间，因此其 used 属性最初为零。随着不断对克隆进行更改，它使用的空间将越来越多。原始快照的 used 属性不考虑克隆所占用的磁盘空间。

# 创建 ZFS 克隆

要创建克隆，请使用 zfs clone 命令，指定从中创建克隆的快照以及新文件系统或卷的名称。新文件系统或卷可以位于 ZFS 分层结构中的任意位置。新数据集的类型（例如，文件系统或卷）与从中创建克隆的快照的类型相同。不能在原始文件系统快照所在池以外的池中创建该文件系统的克隆。

在以下示例中，将创建一个名为 tank/home/ahrens/bug123 的新克隆，其初始内容与快照 tank/ws/gate@yesterday 的内容相同。

```
# zfs snapshot tank/ws/gate@yesterday
# zfs clone tank/ws/gate@yesterday tank/home/ahrens/bug123
```

在以下示例中，将从 projects/newproject@today 快照为临时用户创建克隆工作区 projects/teamA/tempuser。然后，在克隆工作区上设置属性。

```
# zfs snapshot projects/newproject@today
# zfs clone projects/newproject@today projects/teamA/tempuser
# zfs set sharenfs=on projects/teamA/tempuser
# zfs set quota=5G projects/teamA/tempuser
```

# 销毁 ZFS 克隆

使用 zfs destroy 命令可以销毁 ZFS 克隆。例如：

```
# zfs destroy tank/home/ahrens/bug123
```

必须先销毁克隆，才能销毁父快照。

# 使用 ZFS 克隆替换 ZFS 文件系统

借助 zfs promote 命令可以用活动的 ZFS 文件系统的克隆来替换该文件系统。此功能简化了克隆并替换文件系统以使“源”文件系统变为指定文件系统之克隆的功能。此外，通过此功能还可以销毁最初创建克隆所基于的文件系统。如果没有克隆提升 (clone promotion) 功能，就无法销毁活动克隆的“源”文件系统。有关销毁克隆的更多信息，请参见第 102 页中的 “销毁 ZFS 克隆”。

在以下示例中，对 tank/test/productA 文件系统进行了克隆，然后克隆文件系统 tank/test/productAbeta 成为了 tank/test/productA 文件系统。

```
# zfs create tank/test
# zfs create tank/test/productA
# zfs snapshot tank/test/productA@today
# zfs clone tank/test/productA@today tank/test/productAbeta
```

```
# zfs list -r tank/test
NAME                   USED  AVAIL  REFER  MOUNTPOINT
tank/test              314K  8.24G  25.5K  /tank/test
tank/test/productA     288K  8.24G   288K  /tank/test/productA
tank/test/productA@today      0      -   288K  -
tank/test/productAbeta      0  8.24G   288K  /tank/test/productAbeta
# zfs promote tank/test/productAbeta
# zfs list -r tank/test
NAME                   USED  AVAIL  REFER  MOUNTPOINT
tank/test              316K  8.24G  27.5K  /tank/test
tank/test/productA        0  8.24G   288K  /tank/test/productA
tank/test/productAbeta   288K  8.24G   288K  /tank/test/productAbeta
tank/test/productAbeta@today      0      -   288K  -
```

在上面的 `zfs -list` 输出中，可以看到原始 `productA` 文件系统的空间记帐已替换为 `productAbeta` 文件系统。

通过重命名文件系统完成克隆替换过程。例如：

```
# zfs rename tank/test/productA tank/test/productAlegacy
# zfs rename tank/test/productAbeta tank/test/productA
# zfs list -r tank/test
NAME                   USED  AVAIL  REFER  MOUNTPOINT
tank/test              316K  8.24G  27.5K  /tank/test
tank/test/productA     288K  8.24G   288K  /tank/test/productA
tank/test/productA@today      0      -   288K  -
tank/test/productAlegacy      0  8.24G   288K  /tank/test/productAlegacy
```

或者，也可以删除传统的文件系统。例如：

```
# zfs destroy tank/test/productAlegacy
```

# 保存和恢复 ZFS 数据

`zfs send` 命令创建写入标准输出的快照流表示。缺省情况下，生成完整的流。可以将输出重定向到文件或其他系统。`zfs receive` 命令创建其内容在标准输入提供的流中指定的快照。如果接收了完整的流，那么同时会创建一个新文件系统。可通过这些命令来保存 ZFS 快照数据并恢复 ZFS 快照数据和文件系统。请参见下一节中的示例。


- 第 104 页中的 “使用其他备份产品保存 ZFS 数据”
- 第 104 页中的 “保存 ZFS 快照”
- 第 105 页中的 “恢复 ZFS 快照”
- 第 106 页中的 “远程复制 ZFS 数据”


以下是用于保存 ZFS 数据的解决方案：

- 保存 ZFS 快照和回滚快照（如有必要）。

- 保存 ZFS 快照的完整副本和增量副本以及恢复快照和文件系统（如有必要）。
- 通过保存和恢复 ZFS 快照及文件系统来远程复制 ZFS 文件系统。
- 用归档实用程序（如 `tar` 和 `cpio`）或第三方备份产品保存 ZFS 数据。

选择用于保存 ZFS 数据的解决方案时，请考虑以下事项：

- 文件系统快照和回滚快照－如果要轻松创建文件系统的副本并恢复到以前的文件系统版本（如有必要），请使用 `zfs snapshot` 和 `zfs rollback` 命令。例如，如果要从文件系统的早期版本恢复一个或多个文件，则可以使用此解决方案。

  有关创建快照和回滚到快照的更多信息，请参见第 97 页中的 “ZFS 快照概述”。
- 保存快照－使用 `zfs send` 和 `zfs receive` 命令可保存和恢复 ZFS 快照。可以保存快照之间的增量更改，但不能逐个恢复文件。必须恢复整个文件系统快照。
- 远程复制－如果要将文件系统从一个系统复制到另一个系统，请使用 `zfs send` 和 `zfs receive` 命令。此过程与可能跨 WAN 镜像设备的传统卷管理产品有所不同。不需要特殊的配置或硬件。复制 ZFS 文件系统的优点是，可以在其他系统的存储池上重新创建文件系统，并为新创建的池指定不同的配置级别（如 RAID-Z），但是新创建的池使用相同的文件系统数据。

# 使用其他备份产品保存 ZFS 数据

除 `zfs send` 和 `zfs receive` 命令外，还可以使用归档实用程序（如 `tar` 和 `cpio` 命令）保存 ZFS 文件。所有这些实用程序都可以保存和恢复 ZFS 文件属性和 ACL。请选中 `tar` 和 `cpio` 命令的适当选项。

有关 ZFS 和第三方备份产品的信息，请参见位于以下位置的 ZFS FAQ：`http://opensolaris.org/os/community/zfs/faq/#backupsoftware`

# 保存 ZFS 快照

`zfs send` 命令的最常见用法是在用于存储备份数据的另一个系统中保存快照副本和接收快照。例如：

```
host1# zfs send tank/dana@snap1 | ssh host2 zfs recv newtank/dana
```

发送完整的流时，目标文件系统必须不能存在。

使用 `zfs send -i` 选项可以保存增量数据。例如：

```
host1# zfs send -i tank/dana@snap1 tank/dana@snap2 | ssh host2 zfs recv newtank/dana
```

请注意，第一个参数是较早的快照，第二个参数是较晚的快照。在这种情况下，`newtank/dana` 文件系统必须存在，增量接收才能成功。

可将增量 *snapshot1* 源指定为快照名称的最后一个组成部分。此快捷方式意味着只需在 `@` 符号后指定 *snapshot1* 的名称，假定它与 *snapshot2* 都来自同一文件系统。例如：

```
host1# zfs send -i snap1 tank/dana@snap2 > ssh host2 zfs recv newtank/dana
```

此语法与上一示例中的增量语法等效。

尝试从其他文件系统 *snapshot1* 生成增量流时，将显示以下消息：

```
cannot send 'pool/fs@name': not an earlier snapshot from the same fs
```

如果需要存储许多副本，可以考虑使用 `gzip` 命令压缩 ZFS 快照流表示。例如：

```
# zfs send pool/fs@snap | gzip > backupfile.gz
```

# 恢复 ZFS 快照

恢复文件系统快照时，请牢记以下要点：

- 将恢复快照和文件系统。
- 将取消挂载文件系统和所有后代文件系统。
- 文件系统在恢复期间不可访问。
- 要恢复的原始文件系统在恢复期间必须不存在。
- 如果文件系统名称存在冲突，可以使用 `zfs rename` 重命名文件系统。

例如：

```
# zfs send tank/gozer@0830 > /bkups/gozer.083006
# zfs receive tank/gozer2@today < /bkups/gozer.083006
# zfs rename tank/gozer tank/gozer.old
# zfs rename tank/gozer2 tank/gozer
```

可以将 `zfs recv` 用作 `zfs receive` 命令的别名。

如果对文件系统进行更改并且要再次以增量方式发送快照，则必须先回滚接收文件系统。

例如，如果对文件系统进行如下更改：

```
host2# rm newtank/dana/file.1
```

并且要以增量方式发送 `tank/dana@snap3`，则必须先回滚接收文件系统，才能接收新的增量快照。使用 `-F` 选项可以取消回滚步骤。例如：

```
host1# zfs send -i tank/dana@snap2 tank/dana@snap3 | ssh host2 zfs recv -F newtank/dana
```

接收增量快照时，目标文件系统必须已存在。

如果对文件系统进行更改，但不回滚接收文件系统以接收新的增量快照，或者不使用 `-F` 选项，则会看到以下消息：

```
host1# zfs send -i tank/dana@snap4 tank/dana@snap5 | ssh host2 zfs recv newtank/dana
cannot receive: destination has been modified since most recent snapshot
```

在 -F 选项成功之前，会执行以下检查：

- 如果最新快照与增量源不匹配，那么回滚和接收都无法完成，并且会返回一条错误消息。
- 如果意外地向 zfs receive 命令提供了与增量源不匹配的其他文件系统的名称，那么回滚和接收都无法完成，并且会返回以下错误消息。

```
cannot send 'pool/fs@name': not an earlier snapshot from the same fs
```

# 远程复制 ZFS 数据

可以使用 zfs send 和 zfs recv 命令，将快照流表示从一个系统远程复制到另一个系统。例如：

```
# zfs send tank/cindy@today | ssh newsys zfs recv sandbox/restfs@today
```

此命令保存 tank/cindy@today 快照数据并将它恢复到 sandbox/restfs 文件系统，还在 newsys 系统上创建 restfs@today 快照。在此示例中，已将用户配置为在远程系统上使用 ssh。

◆ ◆ ◆ 第 7 章

# 7 使用 ACL 保护 ZFS 文件

本章介绍有关使用访问控制列表 (access control list, ACL) 通过提供比标准 UNIX 权限更详尽的权限来保护 ZFS 文件的信息。

本章包含以下各节：



## 新 Solaris ACL 模型

Solaris 的最近几种旧版本支持主要基于 POSIX 式 ACL 规范的 ACL 实现。基于 POSIX 样式的 ACL 用来保护 UFS 文件，并通过 NFSv4 之前的 NFS 版本进行转换。

引入 NFSv4 后，新 ACL 模型完全支持 NFSv4 在 UNIX 和非 UNIX 客户机之间提供的互操作性。如 NFSv4 规范中所定义，这一新的 ACL 实现提供了更丰富的基于 NT 样式 ACL 的语义。

与旧模型相比，新 ACL 模型的主要变化如下：

- 基于 NFSv4 规范并与 NT 样式的 ACL 相似。
- 提供了更详尽的访问权限集。有关更多信息，请参见表 7–2。
- 分别使用 `chmod` 和 `ls` 命令（而非 `setfacl` 和 `getfacl` 命令）进行设置和显示。
- 提供了更丰富的继承语义，用于指定如何将访问权限从目录应用到子目录等。有关更多信息，请参见第 111 页中的 “ACL 继承”。

两种 ACL 模型均可比标准文件权限提供更精细的访问控制。与 POSIX 式 ACL 非常相似，新 ACL 也由多个访问控制项 (Access Control Entry, ACE) 构成。

POSIX 样式的 ACL 使用单个项来定义允许和拒绝的权限。而新 ACL 模型包含两种类型的 ACE，用于进行访问检查：ALLOW 和 DENY。因此，不能根据任何定义一组权限的单个 ACE 来推断是否允许或拒绝该 ACE 中未定义的权限。

NFSv4 样式的 ACL 与 POSIX 式 ACL 之间的转换如下：

- 如果使用任何可识别 ACL 的实用程序（如 cp、mv、tar、cpio 或 rcp 命令）将具有 ACL 的 UFS 文件传送到 ZFS 文件系统，则 POSIX 式 ACL 会转换为等效的 NFSv4 样式的 ACL。
- 一些 NFSv4 样式的 ACL 会转换为 POSIX 式 ACL。如果 NFSv4 样式的 ACL 未转换为 POSIX 式 ACL，则会显示以下类似消息：

```
# cp -p filea /var/tmp
cp: failed to set acl entries on /var/tmp/filea
```

- 如果在运行当前 Solaris 发行版的系统上使用保留的 ACL 选项（tar -p 或 cpio -P）创建 UFS tar 或 cpio 归档文件，则在运行以前的 Solaris 发行版的系统中提取该归档文件时将丢失 ACL。

  所有文件都以正确的文件模式提取，但会忽略 ACL 项。
- 可以使用 ufsrestore 命令将数据恢复到 ZFS 文件系统中，但 ACL 将丢失。
- 如果尝试对 UFS 文件设置 NFSv4 样式的 ACL，则会显示以下类似消息：

```
chmod: ERROR: ACL type's are different
```

- 如果尝试对 ZFS 文件设置 POSIX 样式的 ACL，则会显示以下类似信息：

```
# getfacl filea
File system doesn't support aclent_t style ACL's.
See acl(5) for more information on Solaris ACL support.
```

有关对 ACL 和备份产品的其他限制信息，请参见第 104 页中的 “使用其他备份产品保存 ZFS 数据”。

# ACL 设置语法的说明

提供以下两种基本的 ACL 格式：

### 用于设置普通 ACL 的语法

```
chmod [options] A[index]{+|=}owner@ |group@ |everyone@:
access-permissions/...[:inheritance-flags]:deny | allow file
```

```
chmod [options] A-owner@, group@,
everyone@:access-permissions/...[:inheritance-flags]:deny | allow file ...
```

```
chmod [options] A[index]- file
```

### 用于设置非普通 ACL 的语法

`chmod [options] A[index]{+|=}user|group:name:`*access-permissions*
`/...[:`*inheritance-flags*`]:deny | allow` *file*

`chmod [options] A-user|group:name:`*access-permissions* `/...[:`*inheritance-flags*`]:deny | allow` *file ...*

`chmod [options] A[index]-` *file*

owner@, group@, everyone@
　标识用于普通 ACL 语法的 *ACL-entry-type*。有关 *ACL-entry-type* 的说明，请参见表 7–1。

user|group:*ACL-entry-ID*（*username* **或** *groupname*）
　标识用于显式 ACL 语法的 *ACL-entry-type*。用户和组的 *ACL-entry-type* 还必须包含 *ACL-entry-ID*、*username* 或 *groupname*。有关 *ACL-entry-type* 的说明，请参见表 7–1。

*access-permissions*/.../
　标识授予或拒绝的访问权限。有关 ACL 访问权限的说明，请参见表 7–2。

*inheritance-flags*
　标识一组可选的 ACL 继承标志。有关 ACL 继承标志的说明，请参见表 7–3。

`deny | allow`
　标识授予还是拒绝访问权限。

在以下示例中，*ACL-entry-ID* 值无意义。

```
group@:write_data/append_data/execute:deny
```

由于 ACL 中包括特定用户 (*ACL-entry-type*)，因此以下示例中包括 *ACL-entry-ID*。

```
0:user:gozer:list_directory/read_data/execute:allow
```

显示的 ACL 项与以下内容类似：

```
2:group@:write_data/append_data/execute:deny
```

本示例中指定的**2** 或**索引** *ID* 用于标识较大 ACL 中的 ACL 项，较大的 ACL 中可能包含对应于属主、特定 UID、组和各用户的多个项。可以使用 `chmod` 命令指定**索引** *ID*，以标识 ACL 要修改的部分。例如，可将索引 ID 3 标识为 `chmod` 命令中的 `A3`，与以下内容类似：

```
chmod A3=user:venkman:read_acl:allow filename
```

下表介绍了 ACL 项的类型，即属主、组和其他对象的 ACL 表示形式。

表 7–1 ACL 项类型

| ACL 项类型 | 说明 |
| --- | --- |
| owner@ | 指定授予对象属主的访问权限。 |
| group@ | 指定授予对象所属组的访问权限。 |
| everyone@ | 指定向不与其他任何 ACL 项匹配的任何用户或组授予的访问权限。 |
| user | 通过用户名指定向对象的其他用户授予的访问权限。必须包括 *ACL-entry-ID*，其中包含 *username* 或 *userID*。如果该值不是有效的数字 UID 或 *username*，则该 ACL 项的类型无效。 |
| group | 通过组名指定向对象的其他组授予的访问权限。必须包括 *ACL-entry-ID*，其中包含 *groupname* 或 *groupID*。如果该值不是有效的数字 UID 或 *groupname*，则该 ACL 项的类型无效。 |

下表介绍了 ACL 访问权限。

表 7–2 ACL 访问权限

| 访问权限 | 缩写访问权限 | 说明 |
| --- | --- | --- |
| add_file | w | 向目录中添加新文件的权限。 |
| add_subdirectory | p | 在目录中创建子目录的权限。 |
| append_data | p | 占位符。当前未实现。 |
| delete | d | 删除文件的权限。 |
| delete_child | D | 删除目录中的文件或目录的权限。 |
| execute | x | 执行文件或搜索目录内容的权限。 |
| list_directory | r | 列出目录内容的权限。 |
| read_acl | c | 读取 ACL 的权限 (ls)。 |
| read_attributes | a | 读取文件的基本属性（非 ACL）的权限。将基本属性视为状态级别属性。允许此访问掩码位意味着该实体可以执行 ls(1) 和 stat(2)。 |
| read_data | r | 读取文件内容的权限。 |
| read_xattr | R | 读取文件的扩展属性或在文件的扩展属性目录中执行查找的权限。 |
| synchronize | s | 占位符。当前未实现。 |

表 7–2 ACL 访问权限 （续）

| 访问权限 | 缩写访问权限 | 说明 |
| --- | --- | --- |
| write_xattr | W | 创建扩展属性或向扩展属性目录进行写入的权限。<br>向用户授予此权限意味着用户可为文件创建扩展属性目录。属性文件的权限可以控制用户对属性的访问。 |
| write_data | w | 修改或替换文件内容的权限。 |
| write_attributes | A | 将与文件或目录关联的时间更改为任意值的权限。 |
| write_acl | C | 编写 ACL 的权限或使用 `chmod` 命令修改 ACL 的能力。 |
| write_owner | o | 更改文件的属主或组的权限，或者对文件执行 `chown` 或 `chgrp` 命令的能力。<br>获取文件拥有权的权限或将文件的组拥有权更改为由用户所属组的权限。如果要将文件或组的拥有权更改为任意用户或组，则需要 `PRIV_FILE_CHOWN` 权限。 |

# ACL 继承

使用 ACL 继承的目的是使新创建的文件或目录可以继承其本来要继承的 ACL，但不忽略父目录的现有权限位。

缺省情况下，不会传播 ACL。如果设置某个目录的非普通 ACL，则该非普通 ACL 不会继承到任何后续目录。必须对文件或目录指定 ACL 的继承。

下表介绍了可选的继承标志。

表 7–3 ACL 继承标志

| 继承标志 | 缩写继承标志 | 说明 |
| --- | --- | --- |
| file_inherit | f | 仅将 ACL 从父目录继承到该目录中的文件。 |
| dir_inherit | d | 仅将 ACL 从父目录继承到该目录的子目录。 |
| inherit_only | i | 从父目录继承 ACL，但仅适用于新创建的文件或子目录，而不适用于该目录自身。该标志要求使用 `file_inherit` 标志或 `dir_inherit` 标志，或同时使用两者来表示要继承的内容。 |
| no_propagate | n | 仅将 ACL 从父目录继承到该目录的第一级内容，而不是第二级或后续内容。该标志要求使用 `file_inherit` 标志或 `dir_inherit` 标志，或同时使用两者来表示要继承的内容。 |

此外，还可以使用 `aclinherit` 文件系统属性对文件系统设置更为严格或更为宽松的缺省 ACL 继承策略。有关更多信息，请参见下一节。

## ACL 属性模式

ZFS 文件系统包括与 ACL 相关的两种属性模式：

- `aclinherit`－此属性可确定 ACL 继承的行为。包括以下属性值：
  - `discard`－对于新对象，创建文件或目录时不会继承任何 ACL 项。文件或目录的 ACL 等效于该文件或目录的权限模式。
  - `noallow`－对于新对象，仅继承访问类型为 `deny` 的可继承 ACL 项。
  - `secure`－对于新对象，继承 ACL 项时将删除 `write_owner` 和 `write_acl` 权限。
  - `passthrough`－对于新对象，将继承可继承的 ACL 项，并且不会对其进行更改。实际上，此模式会禁用 `secure` 模式。

  `aclinherit` 的缺省模式为 `secure`。
- `aclmode`－每次 `chmod` 命令修改文件或目录的模式或者最初创建文件时，此属性都将修改 ACL 的行为。包括以下属性值：
  - `discard`－删除所有 ACL 项，但定义文件或目录的模式所需的项除外。
  - `groupmask`－除非用户项与文件或目录的属主具有相同的 UID，否则将减少用户或组的 ACL 权限，以使其不会大于组权限位。然后，减少 ACL 权限，以使其不会大于属主权限位。
  - `passthrough`－对于新对象，将继承可继承的 ACL 项，并且不会对其进行更改。

  `aclmode` 属性的缺省模式为 `groupmask`。

# 设置 ZFS 文件的 ACL

正如 ZFS 所实现的那样，ACL 由 ACL 项的数组构成。ZFS 提供了一个**纯** ACL 模型，其中所有文件都包括 ACL。通常，ACL 很**普通**，因为它仅表示传统的 UNIX `owner/group/other` 项。

ZFS 文件仍然具有权限位和模式，但这些值大部分是 ACL 所表示内容的高速缓存。因此，如果更改文件的权限，该文件的 ACL 也会相应地更新。此外，如果删除授予用户对文件或目录的访问权限的非普通 ACL，则由于该文件或目录的权限位会将访问权限授予组或各用户，因此该用户仍可访问这一文件或目录。所有访问控制决策都由文件或目录的 ACL 中表示的权限来管理。

对于 ZFS 文件，ACL 访问权限的主要规则如下：

- ZFS 按照 ACL 项在 ACL 中的排列顺序从上至下对其进行处理。
- 仅处理具有与访问权限的请求者匹配的“对象”的 ACL 项。
- 一旦授予允许权限，同一 ACL 权限集当中的后续 ACL 拒绝项即不能拒绝此权限。
- 无条件地授予文件属主 `write_acl` 权限，即使显式拒绝此权限时也是如此。否则，将拒绝仍未指定的所有权限。

如果拒绝权限或缺少访问权限，权限子系统将确定为文件属主或超级用户授予的访问请求。此机制可以防止文件属主无法访问其文件，并允许超级用户修改文件以进行恢复。

如果设置某个目录的非普通 ACL，则该目录的子目录不会自动继承该 ACL。如果设置了非普通 ACL 并希望目录的子目录继承该 ACL，则必须使用 ACL 继承标志。有关更多信息，请参见表 7–3 和第 120 页中的 “以详细格式对 ZFS 文件设置 ACL 继承” 。

创建新文件时，根据 umask 值将应用类似如下的缺省的普通 ACL：

```
$ ls -v file.1
-r--r--r--   1 root     root      206663 May  4 11:52 file.1
     0:owner@:write_data/append_data/execute:deny
     1:owner@:read_data/write_xattr/write_attributes/write_acl/write_owner
         :allow
     2:group@:write_data/append_data/execute:deny
     3:group@:read_data:allow
     4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

请注意，本示例中的每个用户类别（owner@、group@、everyone@）都有两个 ACL 项。一项用于 deny 权限，另一项用于 allow 权限。

此文件 ACL 的说明如下：

| | |
|---|---|
| 0:owner@ | 拒绝属主对文件的执行权限 (execute:deny)。 |
| 1:owner@ | 属主可以读取和修改文件的内容 (read_data/write_data/append_data)。属主还可以修改文件的属性，如时间标记、扩展属性和 ACL (write_xattr/write_attributes/write_acl)。此外，属主还可以修改文件的拥有权 (write_owner:allow)。 |
| 2:group@ | 拒绝组对文件的修改和执行权限 (write_data/append_data/execute:deny)。 |
| 3:group@ | 授予组对文件的读取权限 (read_data:allow)。 |
| 4:everyone@ | 拒绝用户或组之外的所有人员对文件内容的执行或修改权限以及对文件任何属性的修改权限 (write_data/append_data/write_xattr/execute/write_attributes/write_acl/write_owner:deny)。 |
| 5:everyone@ | 向用户或组之外的所有人员授予对文件以及文件属性的读取权限 (read_data/read_xattr/read_attributes/read_acl/synchronize:allow)。synchronize 访问权限当前未实现。 |

创建新目录时，根据 umask 值，缺省目录 ACL 将类似如下：

```
$ ls -dv dir.1
drwxr-xr-x   2 root     root           2 Feb 23 10:37 dir.1
     0:owner@::deny
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     2:group@:add_file/write_data/add_subdirectory/append_data:deny
     3:group@:list_directory/read_data/execute:allow
     4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

此目录 ACL 的说明如下：

`0:owner@` 目录的属主拒绝列表为空 (`::deny`)。

`1:owner@` 属主可以读取和修改目录内容 (`list_directory/read_data/add_file/write_data/add_subdirectory/append_data`)、搜索内容 (`execute`)，修改时间戳、扩展属性和 ACL 等文件属性 (`write_xattr/write_attributes/write_acl`)。此外，属主还可以修改目录的拥有权 (`write_owner:allow`)。

`2:group@` 组不能添加或修改目录内容 (`add_file/write_data/add_subdirectory/append_data:deny`)。

`3:group@` 组可以列出并读取目录内容。此外，组还具有搜索目录内容的执行权限 (`list_directory/read_data/execute:allow`)。

`4:everyone@` 拒绝用户或组之外的所有人员对目录内容的添加或修改权限 (`add_file/write_data/add_subdirectory/append_data`)。此外，还会拒绝修改目录的任何属性的权限 (`write_xattr/write_attributes/write_acl/write_owner:deny`)。

`5:everyone@` 向用户或组之外的所有人员授予对目录内容和目录属性的读取和执行权限 (`list_directory/read_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow`)。`synchronize` 访问权限当前未实现。

# 以详细格式设置和显示 ZFS 文件的 ACL

可以使用 `chmod` 命令修改 ZFS 文件的 ACL。以下用于修改 ACL 的 `chmod` 语法使用 *acl* **规范**来确定 ACL 的格式。有关 *acl* **规范**的说明，请参见第 108 页中的 “ACL 设置语法的说明”。

- 添加 ACL 项
  - 为用户添加 ACL 项

```
% chmod A+acl-specification filename
```

- 按 *index-ID* 添加 ACL 项

```
% chmod Aindex-ID+acl-specification filename
```

此语法用于在指定的 *index-ID* 位置插入新的 ACL 项。

- 替换 ACL 项

```
% chmod Aindex-ID=acl-specification filename
```

```
% chmod A=acl-specification filename
```

- 删除 ACL 项
  - 按 *index-ID* 删除 ACL 项

```
% chmod Aindex-ID- filename
```

  - 由用户删除 ACL 项

```
% chmod A-acl-specification filename
```

  - 从文件中删除所有非普通 ACE

```
% chmod A- filename
```

详细 ACL 信息是通过使用 `ls -v` 命令来显示的。例如：

```
# ls -v file.1
-rw-r--r--   1 root     root      206663 Feb 16 11:00 file.1
     0:owner@:execute:deny
     1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     2:group@:write_data/append_data/execute:deny
     3:group@:read_data:allow
     4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

有关使用缩写 ACL 格式的信息，请参见第 127 页中的 “以缩写格式设置和显示 ZFS 文件的 ACL” 。

**示例 7–1** 修改 ZFS 文件的普通 ACL

本节提供了设置和显示普通 ACL 的示例。

在以下示例中，普通 ACL 存在于 `file.1` 中：

示例 7–1 修改 ZFS 文件的普通 ACL （续）

```
# ls -v file.1
-rw-r--r--   1 root     root      206663 Feb 16 11:00 file.1
     0:owner@:execute:deny
     1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     2:group@:write_data/append_data/execute:deny
     3:group@:read_data:allow
     4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

在以下示例中，为 group@ 授予了 write_data 权限。

```
# chmod A2=group@:append_data/execute:deny file.1
# chmod A3=group@:read_data/write_data:allow file.1
# ls -v file.1
-rw-rw-r--   1 root     root           206663 May  3 16:36 file.1
     0:owner@:execute:deny
     1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     2:group@:append_data/execute:deny
     3:group@:read_data/write_data:allow
     4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

在以下示例中，对 file.1 的权限重新设置为 644。

```
# chmod 644 file.1
# ls -v file.1
-rw-r--r--   1 root     root           206663 May  3 16:36 file.1
     0:owner@:execute:deny
     1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     2:group@:write_data/append_data/execute:deny
     3:group@:read_data:allow
     4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

**示例 7–2** 设置 ZFS 文件的非普通 ACL

本节提供了设置和显示非普通 ACL 的示例。

在以下示例中，为用户 gozer 添加了对 test.dir 目录的 read_data/execute 权限。

```
# chmod A+user:gozer:read_data/execute:allow test.dir
# ls -dv test.dir
drwxr-xr-x+  2 root     root           2 Feb 16 11:12 test.dir
     0:user:gozer:list_directory/read_data/execute:allow
     1:owner@::deny
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     3:group@:add_file/write_data/add_subdirectory/append_data:deny
     4:group@:list_directory/read_data/execute:allow
     5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

在以下示例中，为用户 gozer 删除了 read_data/execute 权限。

```
# chmod A0- test.dir
# ls -dv test.dir
drwxr-xr-x   2 root     root           2 Feb 16 11:12 test.dir
     0:owner@::deny
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     2:group@:add_file/write_data/add_subdirectory/append_data:deny
     3:group@:list_directory/read_data/execute:allow
     4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

**示例 7–3** 与 ZFS 文件权限的 ACL 交互

以下 ACL 示例说明设置 ACL 和随后更改文件或目录的权限位两者之间的相互关系。

在以下示例中，普通 ACL 存在于 file.2 中：

```
# ls -v file.2
-rw-r--r--   1 root     root        2703 Feb 16 11:16 file.2
     0:owner@:execute:deny
```

示例 7–3 与 ZFS 文件权限的 ACL 交互 （续）

```
     1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     2:group@:write_data/append_data/execute:deny
     3:group@:read_data:allow
     4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

在以下示例中，将从 everyone@ 中删除 ACL 允许权限。

```
# chmod A5- file.2
# ls -v file.2
-rw-r-----   1 root     root        2703 Feb 16 11:16 file.2
     0:owner@:execute:deny
     1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     2:group@:write_data/append_data/execute:deny
     3:group@:read_data:allow
     4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
```

在此输出中，文件的权限位从 655 重置为 650。为 everyone@ 删除 ACL 允许权限时，已有效地从文件的权限位中删除了对 everyone@ 的读取权限。

在以下示例中，现有 ACL 将替换为 everyone@ 的 read_data/write_data 权限。

```
# chmod A=everyone@:read_data/write_data:allow file.3
# ls -v file.3
-rw-rw-rw-+  1 root     root        1532 Feb 16 11:18 file.3
     0:everyone@:read_data/write_data:allow
```

在此输出中，chmod 语法有效地将现有 ACL 中的 read_data/write_data:allow 权限替换为对属主、组和 everyone@ 的读取/写入权限。在此模型中，everyone@ 用于指定对任何用户或组的访问权限。由于不存在用以覆盖属主和组的权限的 owner@ 或 group@ ACL 项，因此权限位会设置为 666。

在以下示例中，现有 ACL 将替换为用户 gozer 的读取权限。

```
# chmod A=user:gozer:read_data:allow file.3
# ls -v file.3
----------+  1 root     root        1532 Feb 16 11:18 file.3
     0:user:gozer:read_data:allow
```

**示例 7–3** 与 ZFS 文件权限的 ACL 交互 （续）

在此输出中，文件权限计算结果为 000，这是因为不存在对应 owner@、group@ 或 everyone@ 的 ACL 项，这些项用于表示文件的传统权限组成部分。文件属主可通过重置权限（和 ACL）来解决此问题，如下所示：

```
# chmod 655 file.3
# ls -v file.3
-rw-r-xr-x+  1 root     root           0 Mar  8 13:24 file.3
     0:user:gozer::deny
     1:user:gozer:read_data:allow
     2:owner@:execute:deny
     3:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     4:group@:write_data/append_data:deny
     5:group@:read_data/execute:allow
     6:everyone@:write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:deny
     7:everyone@:read_data/read_xattr/execute/read_attributes/read_acl
         /synchronize:allow
```

**示例 7–4** 恢复 ZFS 文件的普通 ACL

可以使用 chmod 命令来删除文件或目录的所有非普通 ACL。

在以下示例中，test5.dir 中存在 2 个非普通 ACE。

```
# ls -dv test5.dir
drwxr-xr-x+  2 root     root           2 Feb 16 11:23 test5.dir
     0:user:gozer:read_data:file_inherit:deny
     1:user:lp:read_data:file_inherit:deny
     2:owner@::deny
     3:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     4:group@:add_file/write_data/add_subdirectory/append_data:deny
     5:group@:list_directory/read_data/execute:allow
     6:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     7:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

在以下示例中，删除了用户 gozer 和 lp 的非普通 ACL。剩余的 ACL 包含用于 owner@、group@ 和 everyone@ 的六个缺省值。

示例 7–4 恢复 ZFS 文件的普通 ACL （续）

```
# chmod A- test5.dir
# ls -dv test5.dir
drwxr-xr-x   2 root     root           2 Feb 16 11:23 test5.dir
     0:owner@::deny
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     2:group@:add_file/write_data/add_subdirectory/append_data:deny
     3:group@:list_directory/read_data/execute:allow
     4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

# 以详细格式对 ZFS 文件设置 ACL 继承

可以确定如何在文件和目录中继承或不继承 ACL。缺省情况下，不会传播 ACL。如果设置某个目录的非普通 ACL，则任何后续目录都不会继承该 ACL。必须对文件或目录指定 ACL 的继承。

此外，还提供了可在文件系统中进行全局设置的两个 ACL 属性：`aclinherit` 和 `aclmode`。缺省情况下，`aclinherit` 设置为 `secure`，`aclmode` 设置为 `groupmask`。

有关更多信息，请参见第 111 页中的 “ACL 继承” 。

示例 7–5 缺省 ACL 继承

缺省情况下，ACL 不通过目录结构传播。

在以下示例中，为用户 `gozer` 应用了针对 `test.dir` 的非普通 ACE `read_data/write_data/execute`。

```
# chmod A+user:gozer:read_data/write_data/execute:allow test.dir
# ls -dv test.dir
drwxr-xr-x+  2 root     root           2 Feb 17 14:45 test.dir
     0:user:gozer:list_directory/read_data/add_file/write_data/execute:allow
     1:owner@::deny
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     3:group@:add_file/write_data/add_subdirectory/append_data:deny
     4:group@:list_directory/read_data/execute:allow
     5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
```

**示例 7–5** 缺省 ACL 继承 （续）

```
     6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

如果创建了 `test.dir` 子目录，则不会传播用户 `gozer` 的 ACE。如果对 `sub.dir` 的权限授予用户 `gozer` 作为文件属主、组成员或 `everyone@` 进行访问的权限，则该用户只能访问 `sub.dir`。

```
# mkdir test.dir/sub.dir
# ls -dv test.dir/sub.dir
drwxr-xr-x   2 root     root           2 Feb 17 14:46 test.dir/sub.dir
     0:owner@::deny
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     2:group@:add_file/write_data/add_subdirectory/append_data:deny
     3:group@:list_directory/read_data/execute:allow
     4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

**示例 7–6** 对文件和目录授予 ACL 继承

以下一系列示例标识了设置 `file_inherit` 标志时应用的文件和目录的 ACE。

在以下示例中，为用户 `gozer` 添加了对 `test.dir` 目录中的文件的 `read_data/write_data` 权限，以便该用户对于任何新创建的文件都具有读取访问权限。

```
# chmod A+user:gozer:read_data/write_data:file_inherit:allow test2.dir
# ls -dv test2.dir
drwxr-xr-x+  2 root     root           2 Feb 17 14:47 test2.dir
     0:user:gozer:read_data/write_data:file_inherit:allow
     1:owner@::deny
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     3:group@:add_file/write_data/add_subdirectory/append_data:deny
     4:group@:list_directory/read_data/execute:allow
     5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

**示例 7–6** 对文件和目录授予 ACL 继承 （续）

在以下示例中，用户 gozer 的权限应用于新创建的 test2.dir/file.2 文件。授予 ACL 继承 read_data:file_inherit:allow 意味着用户 gozer 可以读取任何新创建的文件的内容。

```
# touch test2.dir/file.2
# ls -v test2.dir/file.2
-rw-r--r--+  1 root     root           0 Feb 17 14:49 test2.dir/file.2
     0:user:gozer:write_data:deny
     1:user:gozer:read_data/write_data:allow
     2:owner@:execute:deny
     3:owner@:read_data/write_data/append_data/write_xattr/write_attributes+
         /write_acl/write_owner:allow
     4:group@:write_data/append_data/execute:deny
     5:group@:read_data:allow
     6:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     7:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

由于此文件的 aclmode 设置为缺省模式 groupmask，因此用户 gozer 对 file.2 不具有 write_data 权限，这是因为该文件的组权限不允许使用此权限。

请注意，设置 file_inherit 或 dir_inherit 标志时所应用的 inherit_only 权限用来通过目录结构传播 ACL。因此，除非用户 gozer 是文件的属主或文件所属组的成员，否则仅授予或拒绝该用户 everyone@ 权限中的权限。例如：

```
# mkdir test2.dir/subdir.2
# ls -dv test2.dir/subdir.2
drwxr-xr-x+  2 root     root           2 Feb 17 14:50 test2.dir/subdir.2
     0:user:gozer:list_directory/read_data/add_file/write_data:file_inherit
         /inherit_only:allow
     1:owner@::deny
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     3:group@:add_file/write_data/add_subdirectory/append_data:deny
     4:group@:list_directory/read_data/execute:allow
     5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

以下一系列示例标识了同时设置 file_inherit 和 dir_inherit 标志时所应用的文件和目录的 ACL。

**示例 7–6** 对文件和目录授予 ACL 继承 （续）

在以下示例中，向用户 gozer 授予了继承用于新创建的文件和目录的读取、写入和执行权限。

```
# chmod A+user:gozer:read_data/write_data/execute:file_inherit/dir_inherit:allow test3.dir
# ls -dv test3.dir
drwxr-xr-x+  2 root     root           2 Feb 17 14:51 test3.dir
     0:user:gozer:list_directory/read_data/add_file/write_data/execute
         :file_inherit/dir_inherit:allow
     1:owner@::deny
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     3:group@:add_file/write_data/add_subdirectory/append_data:deny
     4:group@:list_directory/read_data/execute:allow
     5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow

                # touch test3.dir/file.3
                # ls -v test3.dir/file.3
                -rw-r--r--+  1 root     root           0 Feb 17 14:53 test3.dir/file.3
                     0:user:gozer:write_data/execute:deny
                     1:user:gozer:read_data/write_data/execute:allow
                     2:owner@:execute:deny
                     3:owner@:read_data/write_data/append_data/write_xattr/write_attributes
                         /write_acl/write_owner:allow
                     4:group@:write_data/append_data/execute:deny
                     5:group@:read_data:allow
                     6:everyone@:write_data/append_data/write_xattr/execute/write_attributes
                         /write_acl/write_owner:deny
                     7:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
                         :allow

                # mkdir test3.dir/subdir.1
                # ls -dv test3.dir/subdir.1
                drwxr-xr-x+  2 root     root           2 May  4 15:00 test3.dir/subdir.1
                     0:user:gozer:list_directory/read_data/add_file/write_data/execute
                         :file_inherit/dir_inherit/inherit_only:allow
                     1:user:gozer:add_file/write_data:deny
                     2:user:gozer:list_directory/read_data/add_file/write_data/execute:allow
                     3:owner@::deny
                     4:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
                         /append_data/write_xattr/execute/write_attributes/write_acl
                         /write_owner:allow
```

**示例 7–6** 对文件和目录授予 ACL 继承 （续）

```
     5:group@:add_file/write_data/add_subdirectory/append_data:deny
     6:group@:list_directory/read_data/execute:allow
     7:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     8:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

在以下示例中，由于 group@ 和 everyone@ 的父目录的权限位拒绝写入和执行权限，因此拒绝了用户 gozer 的写入和执行权限。缺省的 aclmode 属性为 secure，这意味着未继承 write_data 和 execute 权限。

在以下示例中，向用户 gozer 授予了继承用于新创建的文件的读取、写入和执行权限，但未将这些权限传播给该目录的后续内容。

```
# chmod A+user:gozer:read_data/write_data/execute:file_inherit/no_propagate:allow test4.dir
# ls -dv test4.dir
drwxr-xr-x+  2 root     root           2 Feb 17 14:54 test4.dir
     0:user:gozer:list_directory/read_data/add_file/write_data/execute
         :file_inherit/no_propagate:allow
     1:owner@::deny
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     3:group@:add_file/write_data/add_subdirectory/append_data:deny
     4:group@:list_directory/read_data/execute:allow
     5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

如以下示例所示，创建新子目录时，用户 gozer 对文件的 read_data/write_data/execute 权限不会传播给新的 sub4.dir 目录。

```
# mkdir test4.dir/sub4.dir
# ls -dv test4.dir/sub4.dir
drwxr-xr-x   2 root     root           2 Feb 17 14:57 test4.dir/sub4.dir
     0:owner@::deny
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     2:group@:add_file/write_data/add_subdirectory/append_data:deny
     3:group@:list_directory/read_data/execute:allow
     4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
```

**示例 7–6** 对文件和目录授予 ACL 继承 （续）

```
     5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

如以下示例所示，gozer 对文件的 read_data/write_data/execute 权限将传播给新创建的文件。

```
# touch test4.dir/file.4
# ls -v test4.dir/file.4
-rw-r--r--+  1 root     root           0 May  4 15:02 test4.dir/file.4
     0:user:gozer:write_data/execute:deny
     1:user:gozer:read_data/write_data/execute:allow
     2:owner@:execute:deny
     3:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     4:group@:write_data/append_data/execute:deny
     5:group@:read_data:allow
     6:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     7:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

**示例 7–7** ACL 模式设置为 Passthrough 时的 ACL 继承

如果 tank/cindy 文件系统的 aclmode 属性设置为 passthrough，则用户 gozer 将继承为新创建的 file.4 应用于 test4.dir 的 ACL，如下所示：

```
# zfs set aclmode=passthrough tank/cindy
# touch test4.dir/file.4
# ls -v test4.dir/file.4
-rw-r--r--+  1 root     root           0 Feb 17 15:15 test4.dir/file.4
     0:user:gozer:read_data/write_data/execute:allow
     1:owner@:execute:deny
     2:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     3:group@:write_data/append_data/execute:deny
     4:group@:read_data:allow
     5:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     6:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

此输出说明对父目录 test4.dir 设置的 read_data/write_data/execute:allow:file_inherit/dir_inherit ACL 会传递给用户 gozer。

**示例 7–8** ACL 模式设置为 Discard 时的 ACL 继承

如果将文件系统的 `aclmode` 属性设置为 `discard`，则目录的权限位更改时，可能会废弃 ACL。例如：

```
# zfs set aclmode=discard tank/cindy
# chmod A+user:gozer:read_data/write_data/execute:dir_inherit:allow test5.dir
# ls -dv test5.dir
drwxr-xr-x+  2 root     root           2 Feb 16 11:23 test5.dir
     0:user:gozer:list_directory/read_data/add_file/write_data/execute
         :dir_inherit:allow
     1:owner@::deny
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     3:group@:add_file/write_data/add_subdirectory/append_data:deny
     4:group@:list_directory/read_data/execute:allow
     5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

如果以后决定要加强目录的权限位，则会废弃非普通 ACL。例如：

```
# chmod 744 test5.dir
# ls -dv test5.dir
drwxr--r--   2 root     root           2 Feb 16 11:23 test5.dir
     0:owner@::deny
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     2:group@:add_file/write_data/add_subdirectory/append_data/execute:deny
     3:group@:list_directory/read_data:allow
     4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /execute/write_attributes/write_acl/write_owner:deny
     5:everyone@:list_directory/read_data/read_xattr/read_attributes/read_acl
         /synchronize:allow
```

**示例 7–9** ACL 继承模式设置为 Noallow 时的 ACL 继承

在以下示例中，设置了两个包含文件继承的非普通 ACL。一个 ACL 允许 `read_data` 权限，一个 ACL 拒绝 `read_data` 权限。此示例还说明了如何可在同一 `chmod` 命令中指定两个 ACE。

```
# zfs set aclinherit=nonallow tank/cindy
# chmod A+user:gozer:read_data:file_inherit:deny,user:lp:read_data:file_inherit:allow test6.dir
```

示例 7–9 ACL 继承模式设置为 Noallow 时的 ACL 继承 （续）

```
# ls -dv test6.dir
drwxr-xr-x+  2 root     root           2 May  4 14:23 test6.dir
     0:user:gozer:read_data:file_inherit:deny
     1:user:lp:read_data:file_inherit:allow
     2:owner@::deny
     3:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/write_xattr/execute/write_attributes/write_acl
         /write_owner:allow
     4:group@:add_file/write_data/add_subdirectory/append_data:deny
     5:group@:list_directory/read_data/execute:allow
     6:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
         /write_attributes/write_acl/write_owner:deny
     7:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

如以下示例所示，创建新文件时，将废弃允许 `read_data` 权限的 ACL。

```
# touch test6.dir/file.6
# ls -v test6.dir/file.6
-rw-r--r--+  1 root     root           0 May  4 13:44 test6.dir/file.6
     0:user:gozer:read_data:deny
     1:owner@:execute:deny
     2:owner@:read_data/write_data/append_data/write_xattr/write_attributes
         /write_acl/write_owner:allow
     3:group@:write_data/append_data/execute:deny
     4:group@:read_data:allow
     5:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     6:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

# 以缩写格式设置和显示 ZFS 文件的 ACL

可通过使用 14 个唯一字母表示权限的缩写格式来设置和显示 ZFS 文件的权限。表 7–2 和表 7–3 中列出了表示缩写权限的字母。

可以使用 `ls -V` 命令显示用于文件和目录的缩写 ACL 列表。例如：

```
# ls -V file.1
-rw-r--r--   1 root     root      206663 Feb 16 11:00 file.1
           owner@:--x-----------:------:deny
           owner@:rw-p---A-W-Co-:------:allow
           group@:-wxp----------:------:deny
```

```
            group@:r-------------:------:allow
         everyone@:-wxp---A-W-Co-:------:deny
         everyone@:r-----a-R-c--s:------:allow
```

以下介绍了缩写的 ACL 输出：

| | |
|---|---|
| `owner@` | 拒绝属主对文件的执行权限 (`x=execute`)。 |
| `owner@` | 属主可以读取和修改文件的内容（`rw=read_data/write_data`、`p=append_data`）。属主还可以修改文件的属性，如时间戳、扩展属性和 ACL（`A=write_xattr`、`W=write_attributes`、`C=write_acl`）。此外，属主还可以修改文件的拥有权 (`O=write_owner`)。 |
| `group@` | 拒绝组对文件的修改和执行权限（`rw=read_data/write_data`、`p=append_data`、`x=execute`）。 |
| `group@` | 授予组对文件的读取权限 (`r=read_data`)。 |
| `everyone@` | 拒绝用户或组之外的所有人员对文件内容的执行或修改权限以及对文件任何属性的修改权限（`w=write_data`、`x=execute`、`p=append_data`、`A=write_xattr`、`W=write_attributes`、`C=write_acl`、`o=write_owner`）。 |
| `everyone@` | 向用户或组之外的所有人员授予对文件以及文件属性的读取权限（`r=read_data`、`a=append_data`、`R=read_xattr`、`c=read_acl`、`s=synchronize`）。`synchronize` 访问权限当前未实现。 |

与详细 ACL 格式相比，缩写 ACL 格式具有以下优点：

- 可将权限指定为 `chmod` 命令的位置参数。
- 可以删除用于标识无权限的连字符 (-) 字符，并且只需指定必需的字母。
- 可以同一方式设置权限和继承标志。

有关使用详细 ACL 格式的信息，请参见第 114 页中的 “以详细格式设置和显示 ZFS 文件的 ACL”。

**示例 7–10** 以缩写格式设置和显示 ACL

在以下示例中，普通 ACL 存在于 `file.1` 中：

```
# ls -V file.1
-rw-r-xr-x   1 root     root      206663 Feb 16 11:00 file.1
            owner@:--x-----------:------:deny
            owner@:rw-p---A-W-Co-:------:allow
            group@:-w-p----------:------:deny
            group@:r-x-----------:------:allow
         everyone@:-w-p---A-W-Co-:------:deny
         everyone@:r-x---a-R-c--s:------:allow
```

**示例 7–10** 以缩写格式设置和显示 ACL （续）

在本示例中，为用户 gozer 添加了对 file.1 的 read_data/execute 权限。

```
# chmod A+user:gozer:rx:allow file.1
# ls -V file.1
-rw-r-xr-x+  1 root     root      206663 Feb 16 11:00 file.1
        user:gozer:r-x-----------:------:allow
            owner@:--x-----------:------:deny
            owner@:rw-p---A-W-Co-:------:allow
            group@:-w-p----------:------:deny
            group@:r-x-----------:------:allow
         everyone@:-w-p---A-W-Co-:------:deny
         everyone@:r-x---a-R-c--s:------:allow
```

为用户 gozer 添加相同权限的另一种方法是在特定位置（例如 4）插入新 ACL。这样，位于位置 4-6 的现有 ACL 将被下推。例如：

```
# chmod A4+user:gozer:rx:allow file.1
# ls -V file.1
-rw-r-xr-x+  1 root     root      206663 Feb 16 11:00 file.1
            owner@:--x-----------:------:deny
            owner@:rw-p---A-W-Co-:------:allow
            group@:-w-p----------:------:deny
            group@:r-x-----------:------:allow
        user:gozer:r-x-----------:------:allow
         everyone@:-w-p---A-W-Co-:------:deny
         everyone@:r-x---a-R-c--s:------:allow
```

在以下示例中，通过使用缩写 ACL 格式向用户 gozer 授予了继承用于新创建的文件和目录的读取、写入和执行权限。

```
# chmod A+user:gozer:rwx:fd:allow dir.2
# ls -dV dir.2
drwxr-xr-x+  2 root     root           2 Aug 28 13:21 dir.2
        user:gozer:rwx-----------:fd----:allow
            owner@:--------------:------:deny
            owner@:rwxp---A-W-Co-:------:allow
            group@:-w-p----------:------:deny
            group@:r-x-----------:------:allow
         everyone@:-w-p---A-W-Co-:------:deny
         everyone@:r-x---a-R-c--s:------:allow
```

另外，还可以剪切 ls -V 输出中的权限和继承标志并将其粘贴到缩写的 chmod 格式中。例如，要将用户 gozer 对 dir.1 的权限和继承标志复制给用户 cindys，可将权限和继承标志 (rwx-----------:f-----:allow) 复制并粘贴到 chmod 命令中。例如：

示例 7–10 以缩写格式设置和显示 ACL （续）

```
# chmod A+user:cindys:rwx-----------:fd----:allow dir.2
# ls -dv dir.2
drwxr-xr-x+  2 root     root           2 Aug 28 14:12 dir.2
       user:cindys:rwx-----------:fd----:allow
        user:gozer:rwx-----------:fd----:allow
            owner@:--------------:------:deny
            owner@:rwxp---A-W-Co-:------:allow
            group@:-w-p----------:------:deny
            group@:r-x-----------:------:allow
         everyone@:-w-p---A-W-Co-:------:deny
         everyone@:r-x---a-R-c--s:------:allow
```

◆ ◆ ◆ 第 8 章

# ZFS 高级主题

本章介绍仿真卷、在安装了区域的 Solaris 系统中使用 ZFS、ZFS 备用根池以及 ZFS 权限配置文件。

本章包含以下各节：



## ZFS 卷

ZFS 卷是表示块设备的数据集，其使用方式与任何块设备类似。ZFS 卷被标识为 `/dev/zvol/{dsk,rdsk}/path` 目录中的设备。

以下示例将创建 5 GB 的 ZFS 卷 `tank/vol`。

```
# zfs create -V 5gb tank/vol
```

创建卷时，会自动将预留空间设置为卷的初始大小。预留空间大小一直等于卷的大小，因此可防止出现意外行为。例如，如果卷大小减小，则可能导致数据受损。更改卷大小时请务必小心。

此外，如果对大小发生更改的卷创建快照，并且尝试回滚该快照或从该快照中创建克隆，则可能会引入文件系统不一致性。

有关可应用于卷的文件系统属性的信息，请参见表 5–1。

如果使用安装了区域的 Solaris 系统，则不能在非全局区域中创建或克隆 ZFS 卷。在非全局区域中创建或克隆卷的任何尝试都将失败。有关在全局区域中使用 ZFS 卷的信息，请参见第 135 页中的 “向非全局区域中添加 ZFS 卷” 。

# 使用 ZFS 卷作为交换设备或转储设备

要设置交换区域，请创建一个特定大小的 ZFS 卷，然后在该设备中启用交换。在 ZFS 文件系统中，不要交换到文件。不支持 ZFS 交换文件配置。

在以下示例中，会添加一个 5 GB 的 tank/vol 卷作为交换设备。

```
# swap -a /dev/zvol/dsk/tank/vol
# swap -l
swapfile                 dev  swaplo blocks   free
/dev/dsk/c0t0d0s1      32,33      16 1048688  1048688
/dev/zvol/dsk/tank/vol 254,1      16 10485744 10485744
```

不支持使用 ZFS 卷作为转储设备。请使用 dumpadm 命令设置转储设备。

# 使用 ZFS 卷作为 Solaris iSCSI 目标

该 Solaris 发行版支持 Solaris iSCSI 目标和启动器。

此外，通过设置卷的 shareiscsi 属性，可以轻松创建 ZFS 卷作为 iSCSI 目标。例如：

```
# zfs create -V 2g tank/volumes/v2
# zfs set shareiscsi=on tank/volumes/v2
# iscsitadm list target
Target: tank/volumes/v2
    iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
    Connections: 0
```

创建 iSCSI 目标后，设置 iSCSI 启动器。有关 Solaris iSCSI 目标和启动器的更多信息，请参见《系统管理指南：设备和文件系统》中的第 15 章 “配置 Solaris iSCSI 启动器（任务）”。

---

**注 –** 也可以使用 iscsitadm 命令来创建和管理 Solaris iSCSI 目标。如果对 ZFS 卷设置 shareiscsi 属性，请勿使用 iscsitadm 命令再创建同一目标设备。否则，同一设备最终将具有重复的目标信息。

---

如果将 ZFS 卷作为 iSCSI 目标，则可以像管理其他 ZFS 数据集一样来管理 ZFS 卷。不过，对于 iSCSI 目标而言，重命名、导出和导入操作的工作方式略有不同。

- 重命名 ZFS 卷时，iSCSI 目标名将保持不变。例如：

```
# zfs rename tank/volumes/v2 tank/volumes/v1
# iscsitadm list target
Target: tank/volumes/v1
    iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
    Connections: 0
```

- 导出包含共享 ZFS 卷的池会导致目标被删除。导入包含共享 ZFS 卷的池会导致目标被共享。例如：

```
# zpool export tank
# iscsitadm list target
# zpool import tank
# iscsitadm list target
Target: tank/volumes/v1
    iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
    Connections: 0
```

所有 iSCSI 目标配置信息都存储在数据集内。与 NFS 共享文件系统相似，在其他系统中导入的 iSCSI 目标也会相应进行共享。

# 在安装了区域的 Solaris 系统中使用 ZFS

以下各节介绍如何在具有 Solaris 区域的系统中使用 ZFS。


- 第 134 页中的 “向非全局区域中添加 ZFS 文件系统”
- 第 134 页中的 “将数据集委托给非全局区域”
- 第 135 页中的 “向非全局区域中添加 ZFS 卷”
- 第 135 页中的 “在区域中使用 ZFS 存储池”
- 第 135 页中的 “在区域内管理 ZFS 属性”
- 第 136 页中的 “了解 zoned 属性”


将 ZFS 数据集与区域关联时，请牢记以下要点：

- 可将 ZFS 文件系统或 ZFS 克隆添加至非全局区域（可以委托或不委托管理控制）。
- 可将 ZFS 卷作为设备添加至非全局区域。
- 此时不能将 ZFS 快照与区域关联。
- 在 Solaris 10 发行版中，请勿将 ZFS 文件系统用作全局区域根路径或非全局区域根路径。在 Solaris Express 发行版中，可以将 ZFS 用作区域根路径，但请记住，不支持对这些区域进行修补或升级。

在以下各节中，ZFS 数据集是指文件系统或克隆。

通过添加数据集，非全局区域可与全局区域共享空间，但区域管理员不能在基础文件系统分层结构中控制属性或创建新文件系统。这与向区域中添加其他任何类型的文件系统相同，应该在主要目的只是为了共享公用空间时才这样做。

使用 ZFS，还可将数据集委托给非全局区域，从而授予区域管理员对数据集及其所有子级的完全控制。区域管理员可以在此数据集中创建和销毁文件系统或克隆，并可修改数据集的属性。区域管理员无法影响尚未添加到区域中的数据集，也不能超过对所引入区域的数据集所设置的任何顶层配额。

在安装了 Solaris 区域的系统中使用 ZFS 时，请考虑以下相互影响：

- 添加到非全局区域的 ZFS 文件系统必须将其 mountpoint 属性设置为 legacy。
- 由于 Solaris 升级进程存在问题，因此 ZFS 文件系统不能充当区域根目录。请勿在委托给非全局区域的 ZFS 文件系统中包括修补程序或升级进程所访问的与系统相关的任何软件。

# 向非全局区域中添加 ZFS 文件系统

如果目标只是与全局区域共享空间，则可添加 ZFS 文件系统作为通用文件系统。添加到非全局区域的 ZFS 文件系统必须将其 mountpoint 属性设置为 legacy。

可以使用 zonecfg 命令的 add fs 子命令将 ZFS 文件系统添加到非全局区域中。例如：

在以下示例中，全局区域中的全局管理员会向非全局区域中添加一个 ZFS 文件系统。

```
# zonecfg -z zion
zonecfg:zion> add fs
zonecfg:zion:fs> set type=zfs
zonecfg:zion:fs> set special=tank/zone/zion
zonecfg:zion:fs> set dir=/export/shared
zonecfg:zion:fs> end
```

此语法可向挂载在 /export/shared 且已配置的 zion 区域中添加 ZFS 文件系统 tank/zone/zion。文件系统的 mountpoint 属性必须设置为 legacy，并且该文件系统不能已在其他位置挂载。区域管理员可在文件系统中创建和销毁文件。不能在其他位置重新挂载文件系统，区域管理员也不能更改该文件系统的属性，如 atime、readonly、compression 等。全局区域管理员负责设置和控制文件系统的属性。

有关 zonecfg 命令以及使用 zonecfg 配置资源类型的更多信息，请参见《系统管理指南：Solaris Containers－资源管理和 Solaris Zones》中的第 II 部分, “Zones”。

# 将数据集委托给非全局区域

如果主要目标是将存储管理委托给区域，则 ZFS 支持通过使用 zonecfg 命令的 add dataset 子命令向非全局区域中添加数据集。

在以下示例中，全局区域中的全局管理员会将一个 ZFS 文件系统委托给非全局区域。

```
# zonecfg -z zion
zonecfg:zion> add dataset
zonecfg:zion:dataset> set name=tank/zone/zion
zonecfg:zion:dataset> end
```

与添加文件系统不同，此语法会使 ZFS 文件系统 tank/zone/zion 在已配置的 zion 区域中可见。区域管理员可以设置文件系统属性，也可以创建其子级。此外，区域管理员还可以拍摄快照、创建克隆或控制整个文件系统分层结构。

有关区域内所允许的操作的更多信息，请参见第 135 页中的 “在区域内管理 ZFS 属性”。

## 向非全局区域中添加 ZFS 卷

不能使用 zonecfg 命令的 add dataset 子命令向非全局区域中添加 ZFS 卷。如果检测到要尝试添加 ZFS 卷，则区域将无法引导。但是，可以使用 zonecfg 命令的 add device 子命令向区域中添加卷。

在以下示例中，全局区域中的全局管理员会向非全局区域添加一个 ZFS 卷：

```
# zonecfg -z zion
zion: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:zion> create
zonecfg:zion> add device
zonecfg:zion:device> set match=/dev/zvol/dsk/tank/vol
zonecfg:zion:device> end
```

此语法可将 tank/vol 卷导出到区域中。请注意，即使卷不与物理设备对应，向区域中添加原始卷仍然可能存在安全风险。具体来说，区域管理员可能会创建格式错误的文件系统，这在尝试挂载时会使系统发出警告音。有关向区域中添加设备以及相关的安全风险的更多信息，请参见第 136 页中的 “了解 zoned 属性”。

有关向区域中添加设备的更多信息，请参见《系统管理指南：Solaris Containers－资源管理和 Solaris Zones》中的第 II 部分, “Zones”。

## 在区域中使用 ZFS 存储池

不能在区域中创建或修改 ZFS 存储池。委托的管理模型可将对全局区域内的物理存储设备的控制以及对虚拟存储的控制集中到非全局区域。尽管可向区域中添加池级别数据集，但区域内不允许使用用于修改该池的物理特征的任何命令，如创建、添加或删除设备。即使使用 zonecfg 命令的 add device 子命令向区域中添加物理设备或是已使用了文件，zpool 命令也不允许在该区域内创建任何池。

## 在区域内管理 ZFS 属性

将数据集添加到区域后，区域管理员便可控制特定的数据集属性。将数据集添加到区域时，该数据集所有祖先都显示为只读数据集，而该数据集本身则与其所有子级一样是可写的。例如，请参考以下配置：

```
global# zfs list -Ho name
tank
tank/home
```

```
tank/data
tank/data/matrix
tank/data/zion
tank/data/zion/home
```

如果将 tank/data/zion 添加到区域中，则每个数据集都将具有以下属性。

| 数据集 | 可见 | 可写 | 不变属性 |
| --- | --- | --- | --- |
| tank | 是 | 否 | - |
| tank/home | 否 | - | - |
| tank/data | 是 | 否 | - |
| tank/data/matrix | 否 | - | - |
| tank/data/zion | 是 | 是 | sharenfs、zoned、quota、reservation |
| tank/data/zion/home | 是 | 是 | sharenfs、zoned |

请注意，tank/zone/zion 的每个父级都会显示为只读，所有子级都可写，并且不属于父级分层结构的数据集完全不可见。由于非全局区域不能充当 NFS 服务器，因此区域管理员无法更改 sharenfs 属性。区域管理员也不能更改 zoned 属性，否则将产生下节介绍的安全风险。

除了 quota 属性和数据集本身之外，其他所有的可设置属性均可更改。全局区域管理员借助此行为可以控制非全局区域所使用的所有数据集的空间占用情况。

此外，一旦将数据集添加到非全局区域中，全局区域管理员便无法更改 sharenfs 和 mountpoint 属性。

# 了解 zoned 属性

将数据集添加到非全局区域中时，必须对该数据集进行特殊标记，以便不在全局区域的上下文中解释特定属性。将数据集添加到受区域管理员控制的非全局区域之后，便不能再信任其内容。与任何文件系统一样，可能存在 setuid 二进制命令、符号链接或可能对全局区域的安全性造成不利影响的可疑内容。此外，不能在全局区域的上下文中解释 mountpoint 属性。否则，区域管理员可能会影响全局区域的名称空间。为解决后一个问题，ZFS 使用 zoned 属性来指示已在某一时刻将数据集委托给非全局区域。

zoned 属性是在首次引导包含 ZFS 数据集的区域时自动启用的布尔值。区域管理员将无需手动启用此属性。如果设置了 zoned 属性，则不能在全局区域中挂载或共享数据集，并且执行 zfs share -a 命令或 zfs mount -a 命令时将忽略该数据集。以下示例会将 tank/zone/zion 添加至区域中，但不能添加 tank/zone/global：

```
# zfs list -o name,zoned,mountpoint -r tank/zone
NAME                  ZONED  MOUNTPOINT
tank/zone/global        off  /tank/zone/global
tank/zone/zion           on  /tank/zone/zion
# zfs mount
tank/zone/global           /tank/zone/global
tank/zone/zion             /export/zone/zion/root/tank/zone/zion
```

请注意 mountpoint 属性与当前挂载 tank/zone/zion 数据集的目录之间的差异。mountpoint 属性反映的是磁盘上存储的属性，而不是数据集当前在系统中的挂载位置。

从区域中删除数据集或销毁区域时，**不会**自动清除 zoned 属性。此行为是由与这些任务关联的固有安全风险引起的。由于不受信任的用户已对数据集及其子级具有完全访问权限，因此 mountpoint 属性可能会设置为错误值，或在文件系统中可能存在 setuid 二进制命令。

为了防止意外的安全风险，要通过任何方式重用数据集时必须由全局管理员手动清除 zoned 属性。将 zoned 属性设置为 off 之前，请确保数据集及其所有子级的 mountpoint 属性均已设置为合理值并且不存在 setuid 二进制命令，或禁用 setuid 属性。

确定没有任何安全漏洞后，即可使用 zfs set 或 zfs inherit 命令禁用 zoned 属性。如果在区域正在使用数据集时禁用 zoned 属性，则系统的行为方式可能无法预测。仅当确定非全局区域不再使用数据集时，才能更改该属性。

# 使用 ZFS 备用根池

创建池时，该池将固定绑定到主机系统。主机系统一直掌握着池的状况信息，以便可以检测到池何时不可用。此状况信息虽然对于正常操作很有用，但在从备用介质引导或在可移除介质上创建池时则会成为障碍。为解决此问题，ZFS 提供了**备用根**池功能。系统重新引导之后备用根池不会保留，并且所有挂载点都会被修改以与该池的根相关。

## 创建 ZFS 备用根池

创建备用根池的最常见目的是为了与可移除介质结合使用。在这些情况下，用户通常需要一个单独的文件系统，并且希望在目标系统中选择的任意位置挂载该系统。使用 -R 选项创建备用根池时，根文件系统的挂载点将自动设置为 /，这与备用根本身等效。

在以下示例中，名为 morpheus 的池是通过将 /mnt 作为备用根路径来创建的：

```
# zpool create -R /mnt morpheus c0t0d0
# zfs list morpheus
NAME                   USED  AVAIL  REFER  MOUNTPOINT
morpheus              32.5K  33.5G     8K  /mnt/
```

请注意单个文件系统 morpheus，其挂载点是池 /mnt 的备用根。存储在磁盘上的挂载点是 /，/mnt 的全路径仅在备用根池的上下文中才会进行解释。然后，可在不同系统的任意备用根池下导出和导入此文件系统。

## 导入备用根池

也可以使用备用根来导入池。如果挂载点不是在当前根的上下文中而是在可以执行修复的某个临时目录下解释的，则可以使用此功能进行恢复。在挂载可移除介质时（如上所述），也可以使用此功能。

在以下示例中，名为 morpheus 的池是通过将 /mnt 作为备用根路径来导入的。本示例假定之前已导出了 morpheus。

```
# zpool import -R /mnt morpheus
# zpool list morpheus
NAME                    SIZE    USED   AVAIL    CAP  HEALTH     ALTROOT
morpheus               33.8G   68.0K   33.7G     0%  ONLINE     /mnt
# zfs list morpheus
NAME                   USED  AVAIL  REFER  MOUNTPOINT
morpheus              32.5K  33.5G     8K  /mnt/morpheus
```

# ZFS 权限配置文件

如果要在不使用超级用户 (root) 帐户的情况下执行 ZFS 管理任务，则可采用具有以下任一配置文件的角色来执行 ZFS 管理任务：

- ZFS 存储管理－提供了在 ZFS 存储池中创建、销毁和处理设备的能力
- ZFS 文件系统管理－提供了创建、销毁和修改 ZFS 文件系统的能力

有关创建或分配角色的更多信息，请参见《系统管理指南：安全性服务》。

◆ ◆ ◆ 第 9 章

# 9 ZFS 疑难解答和数据恢复

本章介绍如何确定 ZFS 故障模式以及如何从相应故障模式中恢复。还提供了有关预防故障的信息。

本章包含以下各节：



## ZFS 故障模式

作为组合的文件系统和卷管理器，ZFS 可以呈现许多不同的故障模式。本章首先概述各种故障模式，然后讨论如何在正运行的系统上确定各种故障。本章最后讨论如何修复问题。ZFS 可能会遇到以下三种基本类型的错误：



请注意，单个池可能会遇到所有这三种错误，因此完整的修复过程依次查找和更正各个错误。

## ZFS 存储池中缺少设备

如果某设备已从系统中彻底删除，则 ZFS 会检测到该设备无法打开，并将其置于 FAULTED 状态。这可能会导致整个池变得不可用，但也可能不会，具体取决于池的数据复制级别。如果镜像设备或 RAID-Z 设备中的一个磁盘被删除，仍可以继续访问池。如果删除了镜像的所有组件，删除了 RAID-Z 设备中的多个设备，或删除了单磁盘顶层设备，则池将变成 FAULTED。在重新连接设备之前，无法访问任何数据。

## ZFS 存储池中的设备已损坏

“损坏”一词涵盖各种可能出现的错误。以下是错误示例：

- 由于损坏的磁盘或控制器而导致的瞬态 I/O 错误
- 磁盘上的数据因宇宙射线而损坏
- 导致数据传输至错误目标或从错误源位置传输的驱动程序错误
- 只是另一个用户意外地覆写了物理设备的某些部分

在一些情况下，这些错误是瞬态的，如控制器出现问题时的随机 I/O 错误。在另外一些情况下，损坏是永久性的，如磁盘损坏。但是，若损坏是永久性的，则并不一定表明该错误很可能会再次出现。例如，如果管理员意外覆写了磁盘的一部分，且未出现某种硬盘故障，则不需要替换该设备。准确确定设备出现的错误不是一项轻松的任务，在稍后的一节中将对此进行更详细的介绍。

## ZFS 数据已损坏

一个或多个设备错误（指示缺少设备或设备已损坏）影响顶层虚拟设备时，将出现数据损坏。例如，镜像的一半可能会遇到数千个绝不会导致数据损坏的设备错误。如果在镜像另一面的完全相同位置中遇到错误，则会导致数据损坏。

数据损坏始终是永久性的，因此在修复期间需要特别注意。即使修复或替换基础设备，也将永远丢失原始数据。这种情况通常要求从备份恢复数据。在遇到数据错误时会记录错误，并可以通过常规磁盘清理对错误进行控制，如下一节所述。删除损坏的块后，下一遍清理会识别出数据损坏已不再存在，并从系统中删除该错误的任何记录。

# 检查 ZFS 数据完整性

对于 ZFS，不存在与 `fsck` 等效的实用程序。此实用程序一直以来用于两个目的：数据修复和数据验证。

# 数据修复

对于传统的文件系统，写入数据的方法本身容易出现导致数据不一致的意外故障。由于传统的文件系统不是事务性的，因此可能会出现未引用的块、错误的链接计数或其他不一致的数据结构。添加日志记录确实解决了其中的一些问题，但是在无法回滚日志时可能会带来其他问题。对于 ZFS，这些问题都不存在。磁盘上存在不一致数据的唯一原因是出现硬盘故障（在这种情况下，应该已创建冗余池）或 ZFS 软件中存在错误。

假定 `fsck` 实用程序设计用于修复特定于单独文件系统的已知异常，为没有已知反常的文件系统编写这样的实用程序就是不可能的。以后，可能会有大量经验证明某些数据损坏问题是足够常见、足够简单的，以至可以开发出修复用的实用程序来加以解决，但是使用冗余池总能避免这些问题。

如果池不是冗余池，则始终可能会因数据损坏而造成无法访问某些或所有数据。

# 数据验证

除了数据修复外，`fsck` 实用程序还验证磁盘上的数据是否没有问题。过去，此任务是通过取消挂载文件系统再运行 `fsck` 实用程序执行的，在该过程中可能会使系统进入单用户模式。此情况导致的停机时间的长短与所检查文件系统的大小成比例。ZFS 提供了一种对所有数据执行常规检查的机制，而不是要求显式实用程序执行必要的检查。此功能称为**清理**，在内存和其他系统中经常将它用作一种在错误导致硬盘或软件故障之前检测和防止错误的方法。

# 控制 ZFS 数据清理

每当 ZFS 遇到错误时（不管是在清理中还是按需访问文件时），都会在内部记录该错误，以便您可以快速查看池中所有已知错误的概览信息。

## 显式 ZFS 数据清理

检查数据完整性的最简单的方法是，对池中所有数据启动显式清理操作。此操作对池中的所有数据遍历一次，并验证是否可以读取所有块。尽管任何 I/O 的优先级一直低于常规操作的优先级，但是清理以设备所允许的最快速度进行。虽然进行清理时文件系统应该保持可用而且几乎都做出响应，但是此操作可能会对性能产生负面影响。要启动显式清理，请使用 `zpool scrub` 命令。例如：

```
# zpool scrub tank
```

可以在 `zpool status` 输出中显示当前清理的状态。例如：

```
# zpool status -v tank
  pool: tank
 state: ONLINE
```

```
 scrub: scrub completed with 0 errors on Wed Aug 30 14:02:24 2006
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror    ONLINE       0     0     0
            c1t0d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0

errors: No known data errors
```

请注意，每个池一次只能发生一个活动的清理操作。

可通过使用 `-s` 选项来停止正在进行的清理操作。例如：

```
# zpool scrub -s tank
```

在大多数情况下，旨在确保数据完整性的清理操作应该一直执行到完成。如果清理操作影响了系统性能，您可以自行决定停止清理操作。

执行常规清理还可保证对系统上所有磁盘执行连续的 I/O。常规清理具有副作用，即阻止电源管理将空闲磁盘置于低功耗模式。如果系统通常一直执行 I/O，或功耗不是重要的考虑因素，则可以安全地忽略此问题。

有关解释 `zpool status` 输出的更多信息，请参见第 56 页中的 “查询 ZFS 存储池的状态”。

### ZFS 数据清理和重新同步

替换设备时，将启动重新同步操作，以便将正确副本中的数据移动到新设备。此操作是一种形式的磁盘清理。因此，在给定的时间，池中只能发生一个这样的操作。如果清理操作正在进行，则重新同步操作会暂停当前清理，并在重新同步完成后重新启动清理操作。

有关重新同步的更多信息，请参见第 152 页中的 “查看重新同步状态”。

## 确定 ZFS 中的问题

以下各节介绍如何确定 ZFS 文件系统或存储池中的问题。



可以使用以下功能来确定 ZFS 配置所存在的问题：

- 使用 `zpool status` 命令可提供详细的 ZFS 存储池信息

- 通过 ZFS/FMA 诊断消息报告池和设备故障
- 使用 zpool history 命令可以显示以前修改了池状态信息的 ZFS 命令

大多数 ZFS 疑难解答都可以使用 zpool status 命令解决。此命令对系统中的各种故障进行分析并确定最严重的问题，同时为您提供建议的操作和指向知识文章（用于获取更多信息）的链接。请注意，虽然池可能存在多个问题，但是此命令仅确定其中的一个问题。例如，出现数据损坏错误时总是指示设备之一出现了故障。但替换该故障设备并不能修复数据损坏问题。

此外，提供了 ZFS 诊断引擎，用于诊断和报告池故障及设备故障。另外，还可与池或设备的故障关联的校验和 I/O 设备和池错误。fmd 报告的 ZFS 故障在控制台上以及系统消息文件中显示。在大多数情况下，fmd 消息指导您查看 zpool status 命令中的进一步恢复说明。

基本的恢复过程如下所示：

- 如果合适，请使用 zpool history 命令确定导致错误情况的以前的 ZFS 命令。例如：

```
# zpool history
History for 'tank':
2007-04-25.10:19:42 zpool create tank mirror c0t8d0 c0t9d0 c0t10d0
2007-04-25.10:19:45 zfs create tank/erick
2007-04-25.10:19:55 zfs set checksum=off tank/erick
```

  注意，在上面的输出中，对 tank/erick 文件系统禁用了校验和。建议不要使用此配置。
- 通过在系统控制台上或 /var/adm/messages 文件中显示的 fmd 消息来确定错误。
- 在 zpool status -x 命令中查找进一步的修复说明。
- 修复故障，如：
  - 替换故障设备或缺少的设备，并使其联机。
  - 从备份恢复故障配置或损坏的数据。
  - 使用 zpool status -x 命令验证恢复。
  - 备份所恢复的配置（如果适用）。

本章介绍如何解释 zpool status 输出以便诊断故障类型，并会将您导向后续有关如何修复该问题的相应部分。尽管大多数工作是由命令自动执行的，但是准确了解所确定的问题以便诊断故障类型是很重要的。

# 确定 ZFS 存储池中是否存在问题

确定系统上是否存在任何已知问题的最简单的方法是使用 zpool status -x 命令。此命令仅对出现问题的池进行说明。如果系统上不存在错误池，则该命令显示一条简单的消息，如下所示：

```
# zpool status -x
all pools are healthy
```

如果没有 -x 标志，则该命令显示所有池（如果在命令行上指定了池，则为请求的池）的完整状态，即使池的运行状况良好也是如此。

有关 zpool status 命令的命令行选项的更多信息，请参见第 56 页中的 “查询 ZFS 存储池的状态” 。

# 查看 zpool status 输出

完整的 zpool status 输出与以下内容类似：

```
# zpool status tank
  pool: tank
 state: DEGRADED
status: One or more devices has been taken offline by the administrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using 'zpool online' or replace the device with
        'zpool replace'.
 scrub: none requested
 config:

        NAME         STATE     READ WRITE CKSUM
        tank         DEGRADED     0     0     0
          mirror     DEGRADED     0     0     0
            c1t0d0   ONLINE       0     0     0
            c1t1d0   OFFLINE      0     0     0

errors: No known data errors
```

此输出分为以下几部分：

## 总体池状态信息

zpool status 输出中的开始部分包含以下字段（其中一些字段仅针对出现问题的池显示）：

pool　池的名称。

state　池的当前运行状况。此信息仅指池提供必要复制级别的能力。处于 ONLINE 状态的池可能仍存在故障设备或数据损坏。

status　对池故障的说明。如果未发现问题，则省略此字段。

action　建议用于修复错误的操作。此字段为缩写形式，使用户转到以下节之一。如果未发现问题，则省略此字段。

| | |
|---|---|
| see | 对包含详细修复信息的知识文章的引用。联机文章的更新频率比本指南要高，因此应始终参考其中的最新修复过程。如果未发现问题，则省略此字段。 |
| scrub | 确定清理操作的当前状态，它可能包括完成上一清理的日期和时间、正在进行的清理或者是否未请求清理。 |
| errors | 确定是否存在已知的数据错误。 |

## 配置信息

zpool status 输出中的 config 字段说明构成池的设备的配置布局，以及设备的状态和设备产成的任何错误。其状态可以是以下状态之一：ONLINE、FAULTED、DEGRADED、UNAVAILABLE 或 OFFLINE。如果状态是除 ONLINE 之外的任何状态，则说明池的容错能力已受到损害。

配置输出的第二部分显示错误统计信息。这些错误分为以下三类：

- READ－发出读取请求时出现 I/O 错误。
- WRITE－发出写入请求时出现 I/O 错误。
- CKSUM－校验和错误。设备将损坏的数据作为读取请求的结果返回。

这些错误可用于确定损坏是否是永久性的。小量 I/O 错误数可能指示临时故障，而大量 I/O 错误则可能指示设备出现了永久性问题。这些错误不一定对应于应用程序所解释的数据损坏。如果设备处于冗余配置中，则磁盘设备可能显示无法更正的错误，而镜像或 RAID-Z 设备级别上不显示错误。如果是这种情况，则说明 ZFS 成功检索了正确数据，并尝试从现有副本修复损坏的数据。

有关解释这些错误以确定设备故障的更多信息，请参见第 149 页中的 “确定设备故障的类型”。

最后，在 zpool status 输出的最后一列中显示其他辅助信息。此信息是对 state 字段的详述，以帮助诊断故障模式。如果设备处于 FAULTED 状态，则此字段指示是否无法访问设备或者设备上的数据是否已损坏。如果设备正在进行重新同步，则此字段显示当前的进度。

有关监视重新同步进度的更多信息，请参见第 152 页中的 “查看重新同步状态”。

## 清理状态

zpool status 输出的第三部分说明任何显式清理的当前状态。此信息不是用于指示系统上是否检测到任何错误，但是可以利用此信息来判定数据损坏错误报告的准确性。如果上一清理是最近结束的，则很可能已发现任何已知的数据损坏。

有关数据清理以及如何解释此信息的更多信息，请参见第 140 页中的 “检查 ZFS 数据完整性”。

### 数据损坏错误

zpool status 命令还显示是否有已知错误与池关联。在磁盘清理或常规操作期间，可能已发现这些错误。ZFS 将与池关联的所有数据错误记录在持久性日志中。每当系统的完整清理完成时，都会轮转此日志。

数据损坏错误始终是致命的。出现这种错误表明至少一个应用程序因池中的数据损坏而遇到 I/O 错误。冗余池中的设备错误不会导致数据损坏，而且不会被记录在此日志中。缺省情况下，仅显示发现的错误数。使用 zpool status -v 选项可以列出带有详细说明的完整错误列表。例如：

```
# zpool status -v
  pool: tank
 state: DEGRADED
status: One or more devices has experienced an error resulting in data
        corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
        entire pool from backup.
   see: http://www.sun.com/msg/ZFS-8000-8A
 scrub: resilver completed with 1 errors on Fri Mar 17 15:42:18 2006
config:

        NAME         STATE     READ WRITE CKSUM
        tank         DEGRADED     0     0     1
          mirror     DEGRADED     0     0     1
            c1t0d0   ONLINE       0     0     2
            c1t1d0   UNAVAIL      0     0     0  corrupted data

errors: The following persistent errors have been detected:

          DATASET  OBJECT  RANGE
          5        0       lvl=4294967295 blkid=0
```

也可使用 fmd 在系统控制台上和 /var/adm/messages 文件中显示类似的消息。还可以使用 fmdump 命令跟踪这些消息。

有关解释数据损坏错误的更多信息，请参见第 154 页中的 “确定数据损坏的类型”。

## ZFS 错误消息的系统报告

除了持久跟踪池中的错误外，ZFS 还在发生相关事件时显示系统日志消息。以下情况将生成事件以通知管理员：

- **设备状态转换**－如果设备变为 FAULTED 状态，则 ZFS 将记录一条消息，指出池的容错能力可能已受到危害。如果稍后将设备联机，将池恢复正常，则将发送类似的消息。

- **数据损坏**－如果检测到任何数据损坏，则 ZFS 将记录一条消息，说明检测到数据损坏的时间和位置。仅在首次检测到数据损坏时才记录此消息。后续访问不生成消息。
- **池故障和设备故障**－如果出现池故障或设备故障，则 Fault Manager 守护进程将通过系统日志消息以及 `fmdump` 命令报告这些错误。

如果 ZFS 检测到设备错误并自动从其恢复，则不进行通知。这样的错误不会造成池冗余或数据完整性方面的故障。并且，这样的错误通常是由伴随有自己的一组错误消息的驱动程序问题导致的。

# 修复损坏的 ZFS 配置

ZFS 在根文件系统上维护活动池及其配置的高速缓存。如果此文件已损坏或者不知何故变得与磁盘上所存储的内容不同步，则无法再打开池。虽然基础文件系统和存储的质量始终可能会带来任意的损坏，但是 ZFS 会尽量避免出现此情况。此情况通常会导致池从系统中消失（它原本应该是可用的）。此情况还可能表明其本身并不是一个完整的配置，缺少一定数目（具体数目未知）的顶层虚拟设备。在这两种情况下，都可以通过先导出池（如果它确实是可见的）再重新导入它来恢复配置。

有关导入和导出池的更多信息，请参见第 61 页中的 “迁移 ZFS 存储池”。

# 修复缺少的设备

如果设备无法打开，则它在 `zpool status` 输出中显示为 `UNAVAILABLE`。此状态表示在首次访问池时 ZFS 无法打开设备，或者设备自那时以来已变得不可用。如果设备导致顶层虚拟设备不可用，则无法访问池中的任何内容。此外，池的容错能力可能已受到损害。无论哪种情况，只需要将设备重新附加到系统即可恢复正常操作。

例如，设备出现故障后，可能会在 `fmd` 的输出中看到与以下内容类似的消息：

```
SUNW-MSG-ID: ZFS-8000-D3, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Thu Aug 31 11:40:59 MDT 2006
PLATFORM: SUNW,Sun-Blade-1000, CSN: -, HOSTNAME: tank
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: e11d8245-d76a-e152-80c6-e63763ed7e4e
DESC: A ZFS device failed.  Refer to http://sun.com/msg/ZFS-8000-D3 for more information.
AUTO-RESPONSE: No automated response will occur.
IMPACT: Fault tolerance of the pool may be compromised.
REC-ACTION: Run 'zpool status -x' and replace the bad device.
```

下一步是使用 `zpool status -x` 命令查看有关设备问题和解决方法的更详细的信息。例如：

```
# zpool status -x
  pool: tank
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-D3
 scrub: resilver completed with 0 errors on Thu Aug 31 11:45:59 MDT 2006
config:

        NAME         STATE     READ WRITE CKSUM
        tank         DEGRADED     0     0     0
          mirror     DEGRADED     0     0     0
            c0t1d0   UNAVAIL      0     0     0  cannot open
            c1t1d0   ONLINE       0     0     0
```

从此输出中可以看到，缺少的设备 `c0t1d0` 不起作用。如果确定驱动器有故障，请替换该设备。

然后，使用 `zpool online` 命令将替换的设备联机。例如：

```
# zpool online tank c0t1d0
```

确认池在替换设备后运行状况良好。

```
# zpool status -x tank
pool 'tank' is healthy
```

# 以物理方式重新附加设备

重新附加缺少的设备的具体方式取决于相关设备。如果设备是网络连接驱动器，则应该恢复连通性。如果设备是 USB 或其他可移除介质，则应该将它重新附加到系统。如果设备是本地磁盘，则控制器可能已出现故障，以致设备对于系统不再可见。在这种情况下，应该替换控制器，以使磁盘重新可用。可能存在其他反常，具体取决于硬件的类型及其配置。如果驱动器出现故障，且对系统不再可见（不大可能的事件），则应该将该设备视为损坏的设备。按照第 149 页中的 “修复损坏的设备”中概述的过程操作。

# 将设备可用性通知 ZFS

将设备重新附加到系统后，ZFS 可能会也可能不会自动检测其可用性。如果池以前是有故障的，或者在附加过程中重新引导了系统，则 ZFS 在尝试打开池时会自动地重新扫描所有驱动器。如果在系统启动时池的性能降低且设备已替换，则必须通知 ZFS 设备现在是可用的并可以使用 `zpool online` 命令重新打开。例如：

```
# zpool online tank c0t1d0
```

有关使设备联机的更多信息，请参见第 50 页中的 “使设备联机”。

# 修复损坏的设备

本节介绍如何确定设备故障类型、清除瞬态错误和替换设备。

## 确定设备故障的类型

术语**损坏的设备**概念相当含糊，它可以用来描述许多可能的情况：

- **位损坏**－随着时间的推移，随机事件（如电磁感应和宇宙射线）可能会导致存储在磁盘上的位在不可预知的事件中发生翻转。这些事件相对少见，但是通常足以导致大系统或长时间运行的系统出现潜在的数据损坏。这些错误通常是瞬态的。
- **误导的读取或写入**－固件错误或硬件故障可以导致整个块的读取或写入引用磁盘上的不正确位置。这些错误通常是瞬态的，尽管大量此类错误可能指示驱动器有故障。
- **管理员错误**－管理员可能无意中用错误的数据覆写了部分磁盘（如在部分磁盘上复制 `/dev/zero`），从而导致磁盘上出现永久性损坏。这些错误始终是瞬态的。
- **临时故障**－磁盘可能在某段时间内变得不可用，从而导致 I/O 失败。此情况通常与网络连接设备相关联，尽管本地磁盘也可能遇到临时故障。这些错误可能是也可能不是瞬态的。
- **损坏或反常的硬件**－此情况是损坏的硬件所呈现的各种问题的集中体现。这可能是一致的 I/O 错误、导致随机损坏的有故障传输或任何数目的故障。这些错误通常是永久性的。
- **脱机的设备**－如果设备处于脱机状态，则假定是管理员因推测该设备有故障而将它置于此状态。将设备置于此状态的管理员可以确定此假定是否正确。

准确确定出现的错误可能是一个很困难的过程。第一步是检查 `zpool status` 输出中的错误计数，如下所示：

```
# zpool status -v pool
```

错误分为 I/O 错误与校验和错误，这两种错误都指示可能的故障类型。典型操作可预知的错误数非常少（在很长一段时间内只能预知几个错误）。如果看到大量的错误，则此情况可能指示即将出现或已出现设备故障。但是，管理员错误的反常可能会导致大的错误计数。另一信息源是系统日志。如果日志显示大量的 SCSI 或光纤通道驱动程序消息，则此情况可能指示出现了严重的硬件问题。如果未生成系统日志消息，则损坏很可能是瞬态的。

目的是回答以下问题：

**此设备上是否可能出现另一错误？**

仅出现一次的错误被认为是**瞬态的**，不指示存在潜在的故障。其持久性或严重性足以指明潜在硬件故障的错误被认为是“致命的”。由于确定错误类型的行为已超出当前可用于 ZFS 的任何自动化软件的功能范围，因此如此多的操作必须由您（即管理员）手动执行。在确定后，可以执行相应的操作。清除瞬态错误，或者替换出现致命错误的设备。以下几节将介绍这些修复过程。

即使设备错误被认为是瞬态的，它仍然可能导致池中出现了无法更正的数据错误。这些错误需要特殊的修复过程，即使认为基础设备运行状况良好或已进行修复也是如此。有关修复数据错误的更多信息，请参见第 153 页中的 “修复损坏的数据”。

# 清除瞬态错误

如果认为设备错误是瞬态的（因为它们不大可能影响设备将来的运行状况），则可以安全地清除设备错误，以指示未出现致命错误。要将 RAID-Z 或镜像设备的错误计数器清零，请使用 `zpool clear` 命令。例如：

```
# zpool clear tank c1t0d0
```

此语法清除与设备关联的任何错误，并清除与设备关联的任何数据错误计数。

要清除与池中虚拟设备关联的所有错误，并清除与池关联的任何数据错误计数，请使用以下语法：

```
# zpool clear tank
```

有关清除池错误的更多信息，请参见第 50 页中的 “清除存储池设备”。

# 替换 ZFS 存储池中的设备

如果设备损坏是永久性的，或者将来很可能出现永久性损坏，则必须替换该设备。是否可以替换设备取决于配置。

## 确定是否可以替换设备

对于要替换的设备，池必须处于 ONLINE 状态。设备必须是冗余配置的一部分，或者其运行状况必须良好（处于 ONLINE 状态）。如果磁盘是冗余配置的一部分，则必须存在可以从其中检索正确数据的足够副本。如果四向镜像中有两个磁盘是有故障的，则可以替换其中任一磁盘（因为运行状况良好的副本是可用的）。但是，如果四向 RAID-Z 设备中有两个磁盘是有故障的，则两个磁盘都不能替换（因为不存在从其中检索数据的足够副本）。如果设备已损坏但处于联机状态，则只要池不处于 FAULTED 状态就可以替换它。但是，除非存在包含正确数据的足够副本，否则会将设备上的任何错误数据复制到新设备。

在以下配置中，可以替换磁盘 c1t1d0，而且将从完好的副本 c1t0d0 复制池中的任何数据。

```
mirror            DEGRADED
    c1t0d0              ONLINE
    c1t1d0              FAULTED
```

虽然因没有可用的正确副本而无法对数据进行自我修复，但是还可以替换磁盘 c1t0d0。

在以下配置中，无法替换任一有故障磁盘。也无法替换 ONLINE 磁盘，因为池本身是有故障的。

```
raidz             FAULTED
    c1t0d0              ONLINE
    c2t0d0              FAULTED
    c3t0d0              FAULTED
    c3t0d0              ONLINE
```

在以下配置中，尽管已将磁盘上存在的错误数据复制到新磁盘，但是任一顶层磁盘都可替换。

```
c1t0d0         ONLINE
c1t1d0         ONLINE
```

如果其中一个磁盘是有故障的，则无法执行替换操作，因为池本身是有故障的。

## 无法替换的设备

如果设备缺失导致池出现故障，或者设备在非冗余配置中包含太多的数据错误，则无法安全地替换设备。如果没有足够的冗余，则不存在可用来恢复损坏设备的正确数据。在这种情况下，唯一的选择是销毁池再重新创建配置，在该过程中恢复数据。

有关恢复整个池的更多信息，请参见第 156 页中的 “修复 ZFS 存储池范围内的损坏”。

## 替换 ZFS 存储池中的设备

确定可以替换设备后，可以使用 zpool replace 命令替换设备。如果要将损坏的设备替换为另一个不同设备，请使用以下命令：

```
# zpool replace tank c1t0d0 c2t0d0
```

此命令首先将数据从损坏的设备或从池中的其他设备（如果处于冗余配置中）迁移到新设备。此命令完成后，将从配置中拆离损坏的设备，此时可以将该设备从系统中移除。如果已移除设备并在同一位置中将它替换为新设备，请使用命令的单设备形式。例如：

```
# zpool replace tank c1t0d0
```

此命令接受未格式化的磁盘，适当地将它格式化，然后开始重新同步其余配置中的数据。

有关 zpool replace 命令的更多信息，请参见第 51 页中的 “替换存储池中的设备”。

## 查看重新同步状态

替换驱动器这一过程可能需要很长一段时间，具体取决于驱动器的大小和池中的数据量。将数据从一个设备移动到另一个设备的过程称为**重新同步**，可以使用 zpool status 命令监视此过程。

传统的文件系统在块级别上重新同步数据。由于 ZFS 消除了卷管理器的人为分层，因此它能够以更强大的受控方式执行重新同步。此功能的两个主要优点如下：

- ZFS 仅重新同步最少量的必要数据。如果发生暂时故障（与彻底替换设备相对），整个磁盘的重新同步在几分钟或几秒钟内即可完成，而无需重新同步整个磁盘，或者通过一些卷管理器支持的“脏区”日志记录使问题复杂化。替换整个磁盘时，重新同步过程所用的时间与磁盘上所用的数据量成比例。如果只使用了池中几 GB 的空间，则替换 500 GB 的磁盘可能只需要几秒的时间。
- 重新同步是可中断的和安全的。如果系统断电或者进行重新引导，则重新同步过程会准确地从它停止的位置继续，而无需手动干预。

要查看重新同步过程，请使用 zpool status 命令。例如：

```
zpool status tank
  pool: tank
 state: ONLINE
status: One or more devices is currently being resilvered.  The pool will
        continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
 scrub: resilver in progress, 56.40% done, 0h0m to go
config:
```

```
        NAME           STATE     READ WRITE CKSUM
        tank           ONLINE       0     0     0
          mirror       ONLINE       0     0     0
            replacing  ONLINE       0     0     0
              c1t0d0   ONLINE       0     0     0
              c2t0d0   ONLINE       0     0     0
            c1t1d0     ONLINE       0     0     0

errors: No known data errors
```

在本示例中，磁盘 `c1t0d0` 被替换为 `c2t0d0`。通过查看状态输出的配置部分中是否显示有 *replacing*，可观察到此替换虚拟设备的事件。此设备不是真正的设备，不可能使用此虚拟设备类型创建池。此设备的用途仅仅是显示重新同步过程，以及准确确定被替换的设备。

请注意，当前正进行重新同步的任何池都处于 `DEGRADED` 状态，这是因为在重新同步过程完成之前，池无法提供所需的冗余级别。虽然 I/O 始终是按照比用户请求的 I/O 更低的优先级调度的（以最大限度地减少对系统的影响），但是重新同步会尽可能快地进行。重新同步完成后，该配置将恢复为新的完整配置。例如：

```
# zpool status tank
  pool: tank
 state: ONLINE
 scrub: scrub completed with 0 errors on Thu Aug 31 11:20:18 2006
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror    ONLINE       0     0     0
            c2t0d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0

errors: No known data errors
```

池再次处于 `ONLINE` 状态，而且原始的坏磁盘 (`c1t0d0`) 已从配置中删除。

# 修复损坏的数据

以下各节介绍如何确定数据损坏的类型以及如何修复数据（如有可能）。


- 第 154 页中的 “确定数据损坏的类型”
- 第 155 页中的 “修复损坏的文件或目录”
- 第 156 页中的 “修复 ZFS 存储池范围内的损坏”

ZFS 使用校验和、冗余和自我修复数据来最大限度地减少出现数据损坏的可能性。但是，如果池不是冗余池，如果将池降级时出现损坏，或者不大可能发生的一系列事件协同损坏数据段的多个副本，则可能会出现数据损坏。不管是什么原因，结果都是相同的：数据被损坏，因此无法再进行访问。所执行的操作取决于被损坏数据的类型及其相对值。可能损坏以下两种基本类型的数据：

- 池元数据－ZFS 需要解析一定量的数据才能打开池和访问数据集。如果此数据被损坏，则整个池或数据集分层结构的整个部分将变得不可用。
- 对象数据－在这种情况下，损坏发生在特定的文件或目录中。此问题可能会导致无法访问该文件或目录的一部分，或者此问题可能导致对象完全损坏。

数据是在常规操作期间和清理过程中验证的。有关如何验证池数据完整性的更多信息，请参见第 140 页中的 “检查 ZFS 数据完整性”。

# 确定数据损坏的类型

缺省情况下，`zpool status` 命令仅说明已出现损坏，而不说明出现此损坏的位置。例如：

```
# zpool status
   pool: monkey
state: ONLINE
status: One or more devices has experienced an error resulting in data
         corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
         entire pool from backup.
    see: http://www.sun.com/msg/ZFS-8000-8A
scrub: none requested
config:

         NAME        STATE     READ WRITE CKSUM
         monkey      ONLINE       0     0     0
           c1t1d0s6  ONLINE       0     0     0
           c1t1d0s7  ONLINE       0     0     0

errors: 8 data errors, use '-v' for a list
```

每个错误仅指示在给定时间点出现了错误。每个错误不一定仍存在于系统上。在正常情况下，会出现此状况。某些临时故障可能会导致数据损坏（在故障结束后将得到自动修复）。完整的池清理可保证检查池中的每个活动块，因此每当清理完成后都会重置错误日志。如果确定错误不再存在，并且不希望等待清理完成，则使用 `zpool online` 命令重置池中的所有错误。

如果数据损坏位于池范围内的元数据中，则输出稍有不同。例如：

```
# zpool status -v morpheus
  pool: morpheus
    id: 1422736890544688191
 state: FAULTED
status: The pool metadata is corrupted.
action: The pool cannot be imported due to damaged devices or data.
   see: http://www.sun.com/msg/ZFS-8000-72
config:

        morpheus    FAULTED   corrupted data
          c1t10d0   ONLINE
```

如果出现池范围的损坏，池将被置于 FAULTED 状态，这是因为池可能无法提供所需的冗余级别。

# 修复损坏的文件或目录

如果文件或目录被损坏，则系统也许仍然能够正常工作，具体取决于损坏的类型。如果系统中的任何位置都不存在完好的数据副本，则任何损坏实际上都是不可恢复的。如果数据很重要，则只能选择从备份恢复受影响的数据。尽管这样，您也许能够从此损坏恢复而不必恢复整个池。

如果损坏出现在文件数据块中，则可以安全地删除该文件，从而清除系统中的错误。使用 zpool status -v 命令可以显示包含持久性错误的文件名列表。例如：

```
# zpool status -v
   pool: monkey
state: ONLINE
status: One or more devices has experienced an error resulting in data
         corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
         entire pool from backup.
    see: http://www.sun.com/msg/ZFS-8000-8A
scrub: none requested
config:

         NAME        STATE     READ WRITE CKSUM
         monkey      ONLINE       0     0     0
           c1t1d0s6  ONLINE       0     0     0
           c1t1d0s7  ONLINE       0     0     0

errors: Permanent errors have been detected in the following files:

/monkey/a.txt
/monkey/bananas/b.txt
/monkey/sub/dir/d.txt
```

```
/monkey/ghost/e.txt
/monkey/ghost/boo/f.txt
```

下面介绍上述输出：

- 如果找到文件的全路径并且已挂载数据集，则会显示该文件的全路径。例如：

  ```
  /monkey/a.txt
  ```

- 如果找到文件的全路径但未挂载数据集，则会显示不带前导斜杠 (/) 的数据集名称，后面是数据集中文件的路径。例如：

  ```
  monkey/ghost:/e.txt
  ```

- 如果由于错误或由于对象没有与之关联的实际文件路径而导致文件路径的对象编号无法成功转换（dnode_t 便是这种情况），则会显示数据集名称，后跟该对象的编号。例如：

  ```
  monkey/dnode:<0x0>
  ```

- 如果元对象集 (Meta-object Set, MOS) 中的对象已损坏，则会显示特殊标签 <metadata>，后跟该对象的编号。

如果损坏发生在目录或文件的元数据中，则唯一的选择是将文件移动到别处。可以安全地将任何文件或目录移动到不太方便的位置，以允许恢复原始对象。

## 修复 ZFS 存储池范围内的损坏

如果损坏出现在池元数据中（该损坏防碍打开池），则必须从备份恢复池及其所有数据。所用的机制通常随池配置和备份策略的不同而不同。首先，保存 zpool status 所显示的配置，以便在销毁池后可以重新创建它。然后，使用 zpool destroy -f 销毁池。此外，将说明数据集的布局和在本地设置的各种属性的文件保存在某个安全的位置（因为在使池无法访问后此信息将变得无法访问）。使用池配置和数据集布局，可以在销毁池后重新构造完整的配置。然后可以使用任何备份或恢复策略填充数据。

# 修复无法引导的系统

根据设计，即使在出错时 ZFS 也是强健而稳定的。尽管这样，在访问池时，软件错误或某些意外反常可能导致系统发出警告音。在引导过程中，必须打开每个池，这意味着这样的故障将导致系统进入应急重新引导循环。为了从此情况恢复，必须通知 ZFS 不要在启动时查找任何池。

ZFS 在 /etc/zfs/zpool.cache 中维护可用池及其配置的内部高速缓存。此文件的位置和内容是专用的，有可能更改。如果系统变得无法引导，则使用 -m milestone=none 引导选项引导到 none 里程碑。系统启动后，将根文件系统重新挂载为可写入，然后删除

/etc/zfs/zpool.cache。这些操作使 ZFS 忘记系统上存在池，从而阻止它尝试访问导致问题的损坏池。然后可以通过发出 svcadm milestone all 命令进入正常系统状态。从备用根引导时，可以使用类似的过程执行修复。

系统启动后，可以尝试使用 zpool import 命令导入池。但是，这样做很可能会导致在引导期间出现的相同错误，因为该命令使用相同机制访问池。如果系统上有多个池，并且您希望导入某个特定池而不访问任何其他池，则必须重新初始化已损坏池中的设备，然后才能安全地导入完好的池。

# 索引

## Z

## 保

## 虚



## 疑



## 硬



## 预



## 整



## 重



## 术

## 传



## 自



## 组