QR-B10-02

**HW&NWPU【秘】**

**信息接受者**

**LVM文件系统格式分析**

**功能及详细设计说明书 R1.0**

**文件编号：**

**作成部门：西工大计算机学院二系**

**西北工业大学**

改版履历：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 改版履历 | | 文件名: | | | | 制定部门：计算机学院二系 |
| 版数 | 承认/日期 | | 查阅/日期 | 作成/日期 | 改版内容 | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |
|  |  | |  |  |  | |

# 目录

[改版履历： 2](#_Toc503096049)

[目录 3](#_Toc503096050)

[1 LVM简介 4](#_Toc503096051)

[1.1 LVM工作原理 4](#_Toc503096052)

[1.1.1 LVM架构 4](#_Toc503096053)

[1.1.2 LVM实现机制 5](#_Toc503096054)

[1.2 Physical Partition物理分区 8](#_Toc503096055)

[1.3 Physical Volume物理卷 8](#_Toc503096056)

[1.4 Physical Extents物理卷基本单元 8](#_Toc503096057)

[1.5 Volume Group卷组 9](#_Toc503096058)

[1.6 Logical Extent逻辑卷基本单元 9](#_Toc503096059)

[1.7 Logical Volume逻辑卷 9](#_Toc503096060)

[2 创建和管理LVM命令 9](#_Toc503096061)

[2.1 创建LVM分区 10](#_Toc503096062)

[2.2 磁盘扩容 12](#_Toc503096063)

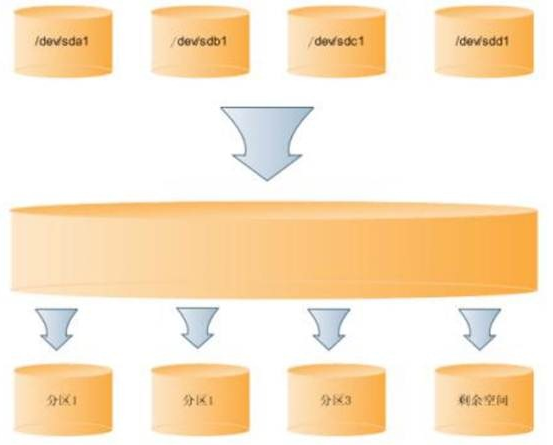
[3 分析步骤及方法 13](#_Toc503096064)

[4 引用文献 14](#_Toc503096065)

# LVM简介

LVM是 Logical Volume Manager(逻辑卷管理)的简写，它由Heinz Mauelshagen在Linux 2.4内核上实现。LVM将一个或多个硬盘的分区在逻辑上集合，相当于一个大硬盘来使用，当硬盘的空间不够使用的时候，可以继续将其它的硬盘的分区加入其中，这样可以实现磁盘空间的动态管理，相对于普通的磁盘分区有很大的灵活性。

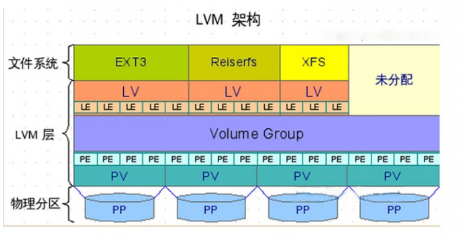
与传统的磁盘与分区相比，LVM为计算机提供了更高层次的磁盘存储。它使系统管理员可以更方便的为应用与用户分配存储空间。在LVM管理下的存储卷可以按需要随时改变大小与移除(可能需对文件系统工具进行升级)。LVM也允许按用户组对存储卷进行管理，允许管理员用更直观的名称(如"sales'、 'development')代替物理磁盘名(如'sda'、'sdb')来标识存储卷。

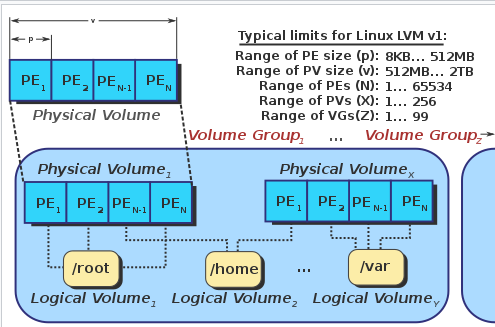


## LVM工作原理

### LVM架构

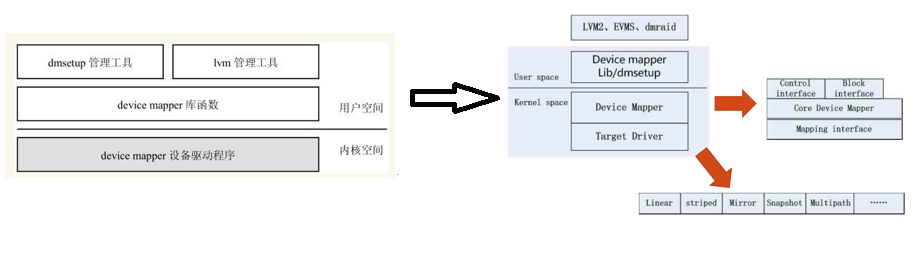
LVM是在磁盘分区和文件系统之间添加的一个逻辑层如下图所示，来为文件系统屏蔽下层磁盘分区布局，提供一个抽象的盘卷，在盘卷上建立文件系统。





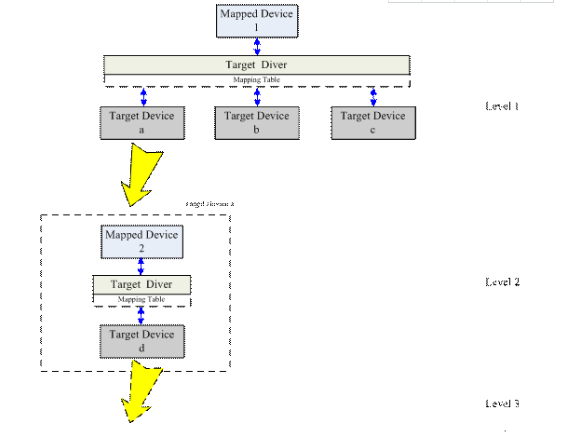
### LVM实现机制

在Linux中，LVM是基于内核模块DM（Device Mapper）的管理工具。如下图所示，处在用户空间的LVM制定磁盘管理策略，将策略发给内核空间的DM进行IO的操作。对于硬件设备的地址空间映射都是DM进行处理，因此有必要进行DM的分析。



#### DM原理介绍

Device mapper 在内核中作为一个块设备驱动被注册的，它包含三个重要的对象概念，mapped device、映射表、target device。Mapped device 是一个逻辑抽象，可以理解成为内核向外提供的逻辑设备，它通过映射表描述的映射关系和 target device 建立映射。从 Mapped device 到一个 target device 的映射表由一个多元组表示，该多元组由表示 mapped device 逻辑的起始地址、范围、和表示在 target device 所在物理设备的地址偏移量以及target 类型等变量组成（这些地址和偏移量都是以磁盘的扇区为单位的，即 512 个字节大小）。Target device 表示的是 mapped device 所映射的物理空间段，对 mapped device 所表示的逻辑设备来说，就是该逻辑设备映射到的一个物理设备。Device mapper 中这三个对象和 target driver 插件一起构成了一个可迭代的设备树。在该树型结构中的顶层根节点是最终作为逻辑设备向外提供的 mapped device，叶子节点是 target device 所表示的底层物理设备。最小的设备树由单个 mapped device 和 target device 组成。每个 target device 都是被mapped device 独占的，只能被一个 mapped device 使用。一个 mapped device 可以映射到一个或者多个 target device 上，而一个 mapped device 又可以作为它上层 mapped device的 target device 被使用，该层次在理论上可以在 device mapper 架构下无限迭代下去。



#### LVM

透过 PV, VG, LV 的规划之后，再利用 mkfs 就可以将你的 LV 格式化成为可以利用的文件系统了！而且这个文件系统的容量在未来还能够进行扩充或减少， 而且里面的数据还不会被影响！实在是很『福气啦！』那实作方面要如何进行呢？很简单呢！ 整个流程由基础到最终的结果可以这样看：

图 3.1.2、 LVM 各组件的实现流程图示

如此一来，我们就可以利用 LV 这个玩意儿来进行系统的挂载了。不过，你应该要觉得奇怪的是， 那么我的数据写入这个 LV 时，到底他是怎么写入硬盘当中的？ 呵呵！好问题～其实，依据写入机制的不同，而有两种方式：

- 线性模式 (linear)：假如我将 /dev/hda1, /dev/hdb1 这两个 partition 加入到 VG 当中，并且整个 VG 只有一个 LV 时，那么所谓的线性模式就是：当 /dev/hda1 的容量用完之后，/dev/hdb1 的硬盘才会被使用到， 这也是我们所建议的模式。

- 交错模式 (triped)：

那什么是交错模式？很简单啊，就是我将一笔数据拆成两部分，分别写入 /dev/hda1 与 /dev/hdb1 的意思，感觉上有点像 RAID 0 啦！如此一来，一份数据用两颗硬盘来写入，理论上，读写的效能会比较好。

基本上，LVM 最主要的用处是在实现一个可以弹性调整容量的文件系统上， 而不是在创建一个效能为主的磁碟上

目前使用的lvm版本为2比lvm1增加了在线修改容量

lvm2(rhel4以后)

实现逻辑过程

线性模式（linear）：

假如将 /dev/sdb1、/dev/sdb2 这两个分区加入到 VG 中，并且 VG 只有一个 LV。那么，线性模式就是，当 /dev/sdb1 的容量用完之后，/dev/sdb2 的硬盘才会使用。

交错模式（triped）：

交错的意思是将一份数据拆成两部分，分别写入 /dev/sdb1 和 /dev/sdb2 。如此一来，一份数据写入两个磁盘。理论上，读写性能会较好。

通常，LVM 最主要的用处就是产生一个大磁盘，而不是在建立性能好的磁盘。所以，我们使用的是 LVM 可以管理整个分区大小的灵活性，而不是着眼在性能上。

因此，LVM 默认的读写模式是线性模式。

如果使用交错模式，要注意，当任何一个分区“归天”时，所有的数据都会“损坏”。所以，不是很合适使用这种模式。

如果要强调性能与备份，那么就直接使用 RAID 即可，不需要使用 LVM。

## Physical Partition物理分区

卷组(VG)中的物理卷划分成固定大小的块（缺省为4MB），这样的块称为物理分区（Physical Partition）。

## Physical Volume物理卷

一个物理卷指一块硬盘。

## Physical Extents物理卷基本单元

物理卷组成的基本单位，大小可以指定为8KB至512MB。

## Volume Group卷组

卷组是可用物理硬盘的集合，可以逻辑地看成一块大硬盘。一个卷组由一个或多个物理卷组成，最多可达32 个（AIX4.3.2 版本下该值已经增至128）。

## Logical Extent逻辑卷基本单元

逻辑卷也被划分为被称为LE(Logical Extents) 的可被寻址的基本单位。在同一个卷组中，多个PE对应于一个LE。

## Logical Volume逻辑卷

每个卷组中可以定义一个或多个逻辑卷，逻辑卷是位于物理分区上的信

息集合（可以是一个文件系统、调页空间、日志或转储设备空间等）。

# LVM格式说明

LVM创建逻辑卷“LVM”的时候，主要包含有三部分即PV，VG和LV等。PV

抽象为逻辑设备提供上层进行调用，数据格式都写在PV这一层上。需要调查研究物理卷组成，其主要包含物理卷标，元数据域，数据域这三个主要信息区域。

### 物理卷标

这个物理卷标会在物理卷的第二个扇区进行数据记录。物理卷标主要包含有两个部分，物理卷标头部信息和物理卷头部信息。

1. 物理卷标头部信息，大小32字节：

|  |  |  |  |
| --- | --- | --- | --- |
| Offset | Size | Value | Description |
| 0 | 8 | "LABELONE" | Signature (or identifier) |
| 8 | 8 |  | Sector number :The sector number of the physical volume label header |
| 16 | 4 |  | Checksum:CRC-32 for offset 20 to end of the physical volume label sector |
| 20 | 4 |  | Data offset (or header size):The offset, in bytes, relative from the start of the physical volume label header |
| 24 | 8 | "LVM2\x20001" | Type indicator |

1. 物理卷头部：

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset** | **Size** | **Value** | **Description** |
| 0 | 32 |  | Physical volume identifier |
| Contains a UUID stored as an ASCII string. |
| 32 | 8 |  | Physical volume size |
| Value in bytes |
| 40 | …​ |  | List of data area descriptors |
| The last descriptor in the list is terminator and consists of 0-byte values. |
| …​ | …​ |  | List of metadata area descriptors |
| The last descriptor in the list is terminator and consists of 0-byte values. |

### 数据域描述

数据域描述主要内容是物理卷的数据域描述，地址偏移和数据数据域大小。

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset** | **Size** | **Value** | **Description** |
| 0 | 8 |  | Data area offset:The offset, in bytes, relative from the start of the physical volume |
|
| 8 | 8 |  | Data area size:Value in bytes |
|

### 元数据域

元数据域包含有两部分，元数据域头部信息和元数据。

#### 元数据域头

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset** | **Size** | **Value** | **Description** |
| 0 | 4 |  | Checksum |
| CRC-32 for offset 4 to end of the metadata area header |
| 4 | 16 | "\x20LVM2\x20x[5A%r0N\*>" | Signature |
| 20 | 4 | 1 | Version |
| 24 | 8 |  | Metadata area offset Value in bytes The offset, in bytes, of the metadata area relative from the start of the physical volume |
|
|
| 32 | 8 |  | Metadata area size |
| The size of the metadata area in bytes |
| 40 | …​ |  | List of raw location descriptors |
| The last descriptor in the list is terminator and consists of 0-byte values. |

#### 元数据

主要包含LVM分层后的各项信息，以及全局信息等。比如物理卷子部信息、卷组子部信息和逻辑卷的子部信息等。

参数定义规则：

1）公式：<identifier> = <value>

2）<value>可以成为如下值:

|  |  |
| --- | --- |
| **Value** | **Description** |
| [0-9]+ | An integer |
| "…​" | A string |
| ["…​", "…​", …​] | A list (or array) of strings |

1. 卷组信息

<name> {<parameters>

<sub sections>}

<name> ：卷组名字

<parameters>：可以是如下值：

|  |  |
| --- | --- |
| **Value** | **Description** |
| id | Volume group identifier (VG UUID) |
| Contains an ASCII string in the following format: fg1fKZ-xoHz-CfAD-yQPx-l2HL-Y7kA-9kJ9LD |
| seqno | Metadata sequence number |
| status | The status flags |
|
| flags | The flags Contains a list of strings. |
|
| extent\_size | The size of an extent |
| The value contains the number of sectors |
| max\_lv | Maximum number of logical volumes |
| max\_pv | Maximum number of physical volumes |
| metadata\_copies | The number of metadata copies |

<sub sections>

|  |  |
| --- | --- |
| **Value** | **Description** |
| physical\_volumes | The physical volumes sub sections |
| logical\_volumes | The logical volumes sub sections |

1. 物理卷子部信息

physical\_volumes {<sub sections>}

|  |  |
| --- | --- |
| **Value** | **Description** |
| pv# | Individual physical volume sub section |
| Where # is a place holder for a the physical volume number e.g. pv0. 0 appears to be the first number that is used |

1. 逻辑卷子部信息

logical\_volumes {<sub sections>}

|  |  |
| --- | --- |
| **Value** | **Description** |
| <name> | Individual physical volume sub section |
| Where <name> is a place holder for a the logical volume name |

# 创建和管理LVM命令

1. 查看

|  |  |
| --- | --- |
| 命令 | 说明 |
| pes、pedisplay | 查看pe的大小(pes==pescan) |
| pvs、pvdisplay | 查看物理卷 |
| vgs、vgdisplay | 查看卷组 |
| lvs、lvdisplay | 查看逻辑卷 |
| fdisk | 查看磁盘分区 |

1. 创建

|  |  |
| --- | --- |
| 命令 | 说明 |
| pvcreate | 创建物理卷 |
| vgcreate | 创建卷组 |
| lvcreate | 创建逻辑卷 |

1. 逻辑卷删除

|  |  |
| --- | --- |
| 命令 | 说明 |
| lvremove | lv完整路径 |
| vgremove | vg名 |
| pvremove | 设备完整路径 去硬盘 |

1. 逻辑卷扩展

|  |  |
| --- | --- |
| 命令 | 说明 |
| vgextend | 扩展vg |
| lvextend | 扩展lv |
| resize2fs | 刷新文件系统 |

1. 逻辑卷的缩小

|  |  |
| --- | --- |
| 命令 | 说明 |
| resize2fs | 减少文件系统 |
| lvreduce | 减少lv卷大小 |

## 创建LVM分区

LVM:逻辑分区管理，可基于动态的扩展缩小硬件设备的使用空间。设备文件：硬件文件,如一块硬盘、一块U盘、一个(硬盘)分区等。当将他们格式化的时候就是选择把他们格式成那种文件系统属性（在windows里有fat32、NTFS等，在linux里有ext2、ext3，unix里有jfs、jfs2等） 。

linux是直接将硬件设备文件在dev下显示出来，并将其mount到某个地方（某个目录）才能使用，就是将这个硬件设备（比如磁盘或者磁盘的一个分区的容量）分配给了一个或者多个目录使用。

“/“分区下的目录是因为linux和unix的目录结构所致，与层级关系没有直接关系，比如你有2快SCSI硬盘，在dev下就是sda（主分区就是sda1-sda3，sda5开始时扩展分区，sda4就是sda5、sda6等扩展分区的集合类似windows）和sdb（同sda分区），你将sda的空间分配（挂载）给了/目录，而将sdb的空间分配（挂载）给了/oracle（/目录下的oracle目录）这个目录，虽然oracle的目录结构是/oracle，在“/”目录下，但是空间确实sdb的空间和"/"目录使用的sda空间没有任何关系，坏了也不会影响到“/”目录。反过来“/”坏了不影响/oracle目录。

LVM管理是基于Unix的,在IBM的小机AIX上常用。可以动态的管理硬件设备空间。

整个思路：在一个Unix或者linux系统中，有硬件设备称之为PV，而将PV的空间集合在一个就形成了VG，在VG中划分出多个lv分区（如同windows里的C、D、E等分区）挂载到多个目录中使用（lv与目录是一一对应的关系），可以在VG中动态的调整这些lv的容量，只要vg有空间，那么在vg下的所有lv就可以随意的扩大容量，直到vg的容量满了，vg的容量满了可以在vg中添加pv增加vg的容量

unix中：df -g 用来查看lv

linux中：df -h 用来查看lv

|  |
| --- |
| 步骤：  1.创建pv  2.创建vg，将pv加入vg  3.在vg中创建lv  4.将lv分区格式化  5.将lv分区挂载到某个目录使用 |

具体步骤如下

1. 创建PV

[root@z1~]# pvcreate /dev/sdb1           创建pv /dev/sdb1 将SCSI第二块硬盘的第一个主分区做成pv

[root@z1~]# pvcreate /dev/sdc1           将SCSI第三块硬盘的第一个主分区做成pv（linux操作系统应该在第一个硬盘中）  
查看PV

[root@z1~]# pvdisplay

1. 创建卷组VG

[root@z1~]# vgcreate datavg /dev/sdb1  /dev/sdc1     创建名为datavg 的卷组，将/dev/sdb1和/dev/sdc1两个pv加入到这个卷组  
查看VG  
[root@z1~]#vgdisplay

1. 创建逻辑卷LV

[root@z1~]# lvcreate –L 40G –n lv1 datavg   在datavg中创建一个大小40G,名为lv1的逻辑卷  
查看LV  
[root@z1~]# lvdisplay

1. LV格式化成ext3文件格式

[root@z1~]# mkfs .ext3 /dev/datavg/lv1       将lv1格式化成ext3文件系统类型  
[root@z1~]# partprobe                        在不重新启动机器的情况下系统能够识别这些分区

1. 挂载

创建目录

[root@z1~]# mkdir /oracle  
挂载  
[root@z1~]# mount /dev/datavg/lv1  /oracle       将lv1挂载到/oralce目录  
查看容量 Ls –lh  
对挂载的逻辑卷进行扩容  
[root@z1~]# lvextend –L +50M /dev/rootvg/lv1     将lv扩容50M（datavg有50M空间的前提下）   
查看分区使用情况 [root@z1~]# df –h  
查看磁盘使用情况 fdisk –l

## 磁盘扩容

对lv1进行在线(动态)扩容  
[root@z1~]#resize2fs /dev/datavg/lv1           
#resize2fs是ext2文件系统大小的调整工具，ext3只是多了journal的ext2，也可以用。  
   
对为挂载的逻辑卷进行扩容  
[root@z1~]#lvextend –L +70M /dev/datavg/lv1       给lv1增加70M（70从datavg中来）   
[root@z1~]#e2fsck –f /dev/datavg/lv1              强制检查ext2、ext3、ext4等文件系统的正确性(清理磁盘碎片,让数据不分散,这个在缩小LV的时候至关重要)  
[root@z1~]#lvdipaly  
    
对VG进行扩容(一)   
[root@z1~]# pvcreate /dev/sdb2                     创建pv /dev/sdb2  
[root@z1~]# vgextend datavg /dev/sdb2              将/dev/sdb2这个pv加入卷组datavg  
对VG进行扩容(二)  
[root@z1~]# pvcreate /dev/sdb2                     创建pv /dev/sdb2  
[root@z1~]# pvcreate /dev/sdc3                     创建pv /dev/sdb3  
[root@z1~]# vgcreate datavg /dev/sdb2              创建datavg（里面是/dev/sdb2）  
[root@z1~]# vgcreate rootvg /dev/sdc3              创建rootvg（里面是/dev/sdb3）  
[root@z1~]# vgmerge rootvg datavg                  将rootvg与datavg整合到rootvg  
　　卸载卷的方法:  
　　卸载物理卷:pvremove PVDEVICE  
　　卸载卷组:vgremove VGNAME  
　　卸载逻辑卷:lvremove LVDEVICE  
　　卸载的顺序:先逻辑—卷组—物理卷  
　　卸载前别忘了备份

# 独立冗余磁盘整列(软RAID)

## 简介

RAID:Redundant Arrays of Inexpensive(Independent)Disks，1988年由加利福尼亚大学伯克利分校（University of California-Berkeley） “A Case for Redundant Arrays of Inexpensive Disks”提出。多个磁盘合成一个“阵列”来提供更好的性能、冗余，或者两者都提供，就叫做独立磁盘冗余整列。

## 特点

1）提高IO性能，提升磁盘读写

2）提高耐用性，磁盘冗余来实现

3）多块磁盘组织在一起的工作方式有所不同

## Raid级别

RAID-0：将数据切割成等分，然后按次序存储在磁盘中

RAID-1：将数据切割成等分，然后分别存入各个磁盘中

RAID-5：将数据切割成等分，然后按次序存储在磁盘中，并且每个磁盘轮流存储校验信息

......

RAID-6: 将数据切割成等分，然后按次序存储在磁盘中，并且每个磁盘轮流存储校验信息，且存两份校验信息

RAID-10: 先做raid1，然后做raid0即可

RAID-01: 先做raid0，然后做raid1即可

## raid级别及其特性

RAID-0；读写性能提升

可用空间n\*min(s1,s2...)

    无容错能力

    最少磁盘数量：2个

RAID-1：读性能提升，写性能有所下降

    1\*min(s1,s2,...)

    有冗余能力

    最少磁盘数量：2,2N

RAID-5: 读写性能提升

    可用空间(N-1)\*min(s1,s2)

    有容错能力，只允许坏一块磁盘

    最少磁盘数量：3,3+

RAID-6：读写性能提升

    可用空间(N-2)\*min(s1,s2,...)

    有容错能力允许坏两块磁盘

    最少磁盘数量：4,4+

RAID-10：读、写性能提升

    可用空间：N\*min(S1,S2,...)/2

     有容错能力：每组镜像最多只能坏一块

    最少磁盘数：4, 4+

RAID-01：读、写性能提升

    可用空间：N\*min(S1,S2,...)/2

     有容错能力：每组镜像最多只能坏一块

    最少磁盘数：4, 4+

# 分析步骤及方法

# 引用文献

<https://en.wikipedia.org/wiki/Logical_Volume_Manager_(Linux)>

<http://www.doc88.com/p-675124528501.html>

<https://www.ibm.com/developerworks/cn/linux/l-devmapper/index.html>

<http://blog.51cto.com/purify/1844274>