跟我一起写 udev 规则

udev 面向 2.6 以上的 linux 内核在用户空间提供动态的/dev 下固定设备命名方案. 之前的/dev 实现: devfs 现在已被废弃, udev 成为继任者. udev vs devfs 是一个敏感的谈话内容,在进行比较之前你应该读一下这个文档(http://kernel.org/pub/linux/utils /kernel/hotplug/udev_vs_devfs).

几年间你为之使用 udev 规则的设备发生改变了,如同规则自身的弹性一样. 在现代系统中 udev 为系统外的类型设备提供了固定的命名方法, 避免了为这些设备提供定制规则. 但是一些用户仍然需要额外的定制级别.
本文档假设你已经安装了 udev 并使用缺省配置运行 ok. 这通常通过你的 linux 发行版做到的.

语义: devfs, sysfs, nodes 等

在典型的基于 linux 的系统中,/dev 目录用来存储文件一样的设备节点, 它们指向系统中特定的设备. 每一个节点指向系统的一部分(一个设备), 可能存在也可能不存在. 用户空间应用程序可以使用这些设备节点跟系统硬件打交道,例如, X 服务器"监听"/dev/input/mice 来根据用户的鼠标移动来移动可视鼠标指针.

原来的/dev 目录仅仅在设备可能在系统中出现时产生,因此/dev 目录一般非常大. 随之而来的 devfs 提供了一种易于管理的途径(注意它仅仅在硬件插入到系统中时产生/dev)以及其他功能,但系统会出现无法容易修复的问题. udev 是一种新的管理/dev 目录的方法,它的设计清除了以前的/dev 实现的一些问题并提供了鲁棒的路径向后兼容. 为了创建并命名系统中相应的/dev 设备结点,udev 需要依赖于根据用户提供的规则从 sysfs 中得到的匹配信息. 本文着重规则书写的过程,udev 相关的任务由用户自己完成.

sysfs 是 2.6 内核中一个新的文件系统, 它由内核管理,并导出当前系统中插入的设备基本信息. udev 可使用这些信息创建对应的硬件设备结点. sysfs 挂载在/sys 下而且是可浏览的. 你可能很希望在使用 udev 之前刺探下存储在那儿的有关文件. 本文中我将交替使用/sys 和 sysfs 术语.

为什么？

udev 规则具有弹性非常强大，这里是一些你使用规则可以达到的结果：

1. 重命名设备节点的缺省名字为其他名字
2. 通过创建符号链接到缺省设备节点来提供一个可选的固定的设备节点名字
3. 基于程序的输出命名设备节点
4. 改变设备节点的权限和所有权
5. 但设备节点被创建或删除时(通常是添加设备或拔出设备时)执行一个脚本
6. 重命名网络接口

当存在的特定设备没有设备节点时,这不是书写规则的工作范围. 即使没有匹配的规则,udev 也会利用内核提供的缺省名字来创建设备节点.
拥有固定命名设备节点有很多好处. 假设你有两个 USB 存储设备:一个数码相机,一个是 USB 闪存盘. 这些设备通过被赋予/dev/sda 和/dev/sdb 设备节点,准确的赋值取决于它们连接到系统的顺序. 这可能为一些用户造成麻烦,如果每个设备每次都可以固定命名,比如/dev/camera 和/dev/flashdisk,用户就会获益.

内置固定命名方法

udev 为系统外的一些设备类型提供了固定命名,这是一个很有用的特征,在某些情况下意味着你不用书写任何规则. udev 为存储设备在/dev/disk 目录下提供了系统外命名方法. 要查看它为你的存储硬件创建的固定命名,你可以使用下列命名:

```
#ls -lR /dev/disk
```

所有存储类型都可以这么用. 例如 udev 为我的根分区创建了固定命名链接:/dev/disk/by-id/scsi-SATA_ST3120827AS_4MS1NDXZ- part3. 但我插入我的 USB 闪存盘 udev 就会创建另外一个固定命名节点:/dev/disk/by-id/usb- Prolific_Technology_Inc._USB_Mass_Storage_Device-part1.

规则书写

规则文件和语义

为决定如何命名设备以及执行什么另外动作,udev 会读取一系列规则文件. 这些文件保存在/etc/udev/rules.d 目录下并且都必须有.rules 后缀名.

缺省 udev 规则存储在/etc/udev/rules.d/50-udev.rules 里面. 你可能发现整个文件很有意思, 它包含了少量例子, 一些缺省规则提供了 devfs 风格的/dev 布局, 但是你不应该直接在这个文件里面书写规则.

/etc/udev/rules.d/下面的文件通过 lexical 顺序解析,在某些情况下规则的解析顺序很重要. 通常来说你希望你的规则可以在缺省规则之前解析, 所以我建议你创建一个文件/etc/udev/rules.d/10-local.rules 并把自己的所有规则写到这里面去.

在一个规则文件中, 以"#"开头的行被认为是注释. 每一个非空的行都是一条规则. 规则不能跨越多行.

一个设备可以被多条规则匹配到, 这有着很实用的优点, 例如, 我们可以写两个匹配同一个设备的规则, 每一个规则为设备提供了它自己的可选命名. 即使分开在不同的文件种, 两个可选命名也都会被创建, 要明白 udev 在找到一个匹配规则后不会停止处理其他规则, 它仍然会继续查找并尝试应用已知的每条规则, 这很重要.

规则语法

每条规则通过一系列键值对创建,这些键值对通过逗号分隔. 匹配键是用来识别要应用规则的设备的条件, 但规则中对应设备的所有匹配键被处理后,就会应用规则并且赋值键的行为也会触发. 每条规则应该包含至少一个匹配键和至少一个赋值键.

这是用来阐述上面内容的一个例子规则:

KERNEL=="hdb",NAME="my_spare_disk"

上述规则包含一个匹配键(KERNEL)以及一个赋值键(NAME). 这些键和它们的属性的语义将在稍后具体说明. 注意到匹配键通过连等号(==)与它的值联系起来, 赋值键通过等号(=)与它的值关联.

注意 udev 不支持任何形式的行连接符, 不要在你的规则种插入任何断行符,这将会导致 udev 把你的一条规则看做是多条规则但不会按预料工作.

基本规则

udev 提供一些用来书写精确匹配规则的匹配键, 其中一些常用键将在下面介绍, 其他将在文档的后面说明. 要得到完整列表可以查看 udev 的手册页.

KERNEL - 为设备匹配的内核名字

SUBSYSTEM - 匹配设备的子系统

DRIVER - 匹配支持设备的驱动名称

在你使用一系列匹配键来准确匹配设备后,udev 通过赋值键为接下来发生的事给你提供更好的控制. 你可以查看 udev 的手册页查看完整的赋值键列表. 最基本的赋值键在下面说明, 其他的将在文档结束时说明.

NAME - 设备节点应该使用的名字

SYMLINK - 一个设备节点可选名字的符号链接列表

正如之前所说,udev 会为设备创建一个真正的设备节点. 如果你希望为设备节点提供可选名字,你得通过 SYMLINK 使用符号链接功能, 实际上是维护一个符号链接列表,这些符号链接都会指向真实的设备节点. 为了维护这些链接我们介绍一个新的附加操作符: +=. 你可以在一个规则中附加多个符号链接到列表中,每个链接通过空格分开.

KERNEL=="hdb", NAME="my_spare_disk"

上面规则意思是:匹配一个设备命名为 hdb 的设备,把它重新命名为 my_spare_disk. 设备节点出现在/dev/my_spare_disk.

KERNEL=="hdb", DRIVER=="ide-disk", SYMLINK+="sparedisk"

上面规则意思是：匹配一个内核命名为 hdb 以及驱动为 ide-disk 的设备，命名设备节点为缺省名字并创建一个指向它的 sparedisk 符号链接。注意到我们没有指明设备节点名字，于是 udev 使用缺省名字。为了保留标准/dev 布

局,你自己的规则通常没有 NAME 但会创建一些 SYMLINK 并/或执行其他赋值操作.
KERNEL=="hdc", SYMLINK+="cdrom cdrom0"
上面规则很可能就是你要写的典型规则。它在/dev/cdrom 和/dev/cdrom0 创建了两个符号链接,都指向/dev/hdc. 再一次地,没有 NAME 赋值键,所以使用缺省的内核名字(hdc).

匹配 sysfs 属性

到目前为止介绍的匹配键仅仅提供了有限的匹配能力. 实际上我们需要更加优良的控制:我们想基于设备的高级属性来识别设备, 如供应商编码, 产品编号, 序列号, 存储能力, 分区数等等.
一些驱动导出这些信息到 sysfs, udev 允许我们使用 ATTR 键通过稍捎不同的语法来合并 sysfs 匹配到自己的规则中. 这里有一个匹配 sysfs 中单个属性例子. 更多细节将在稍后的帮助你基于 sysfs 属性书写规则的文档中提供.
SUBSYSTEM=="block", ATTR{size}=="234441648", SYMLINK+="my_disk"

设备级联

linux 内核实际上以树状结构展示设备, 这个信息通过 sysfs 显露出来,在书写规则时这非常有用. 例如我的硬盘设备的展示是一个 SCSI 磁盘设备的孩子, 这个 SCSI 磁盘设备又是一个 ATA 控制器设备的孩子, 该控制器又是 PCI 总线设备的孩子. 你很有可能发现你需要从一个讨论中的设备的双亲那里引用信息, 比如我的硬盘设备的序列号在设备级别并不暴露出来, 而是在 SCSI 磁盘级别通过它的直接双亲展现。
目前介绍的四个主要匹配键(KERNEL/SUBSYSTEM/DRIVER/ATTR)仅仅跟对应设备的值匹配, 并不跟双亲设备的值匹配. udev 提供了在树中向上查找的匹配键变量:
KERNELS - 为设备匹配的内核名字,或任何双亲设备中的内核名
SUBSYSTEMS - 匹配设备的子系统名,或任何双亲设备中的子系统名
DRIVERS - 匹配支持设备的驱动名,或任何支持双亲设备的驱动名
ATTRS - 匹配设备的 sysfs 属性,或任何双亲设备的 sysfs 属性
由于在心里要考虑到级联结构,你可能感觉到规则书写变的有点复杂了. 歇一会吧,有工具可以帮助我们的,稍微献上.

字符串替换

但写的规则潜在的要处理多个相似的设备时, udev 的 printf-like string substitution operators 就非常有用了. 你可以在你的规则里面的任何赋值里面包含这些操作符, udev 在它们执行时会计算.
最常用的操作符是%k 和%n. %k 计算设备的内核名, 例如设备的"sda3"将(缺省)出现在/dev/sda3. %n 计算设备(存储设备的分区号)的内核号码, 例如"3"将被换成"/dev/sda3".
udev 也提供了一些高级功能替换操作符. 在读完本文剩下内容后可以查询 udev 的手册页. 以上例子中的操作符也有一个可选的语法 - $kernel 和$number. 因此如果你希望在规则中匹配字符%, 你必需写成%%, 如果你希望匹配字符$, 你必须写成$$.
为阐述下字符串替换的概念,我们来展示几个例子:
KERNEL=="mice", NAME="input/%k"
KERNEL=="loop0", NAME="loop/%n", SYMLINK+="%k"
第一条规则确保鼠标设备节点一定出现在/dev/input 目录下(缺省是在/dev/mice 下面). 第二条规则确保名字为 loop0 的设备节点在/dev/loop/0 创建,也会照常创建一个符号链接/dev/loop0.
上面规则的使用都比较可疑,因为他们都可以通过不使用任何替换操作符来重写. 这些替换的真正威力将会在下一节显现.

字符串匹配

不仅有字符串精确匹配, udev 也允许你使用 shell 风格的模式匹配. 支持的 3 种模式为:
* - 匹配任何字符, 匹配 0 次或多次
? - 匹配任何字符,但只匹配一次.

[] - 匹配任何单个字符，这些字符在方括号里面指定，范围是受限的.
这里有一些例子，注意字符串替换符的使用:
KERNEL=="fd[0-9]*", NAME="floppy/%n", SYMLINK+="%k"
KERNEL=="hiddev*", NAME="usb/%k"
第一条规则匹配所有软盘驱动并确保设备节点放置在 /dev/floppy 目录下，也创建一个缺省名字的符号链接．第二条规则确保 hiddev 设备节点放在/dev/usb 目录下面.

从 sysfs 中查找信息

sysfs 树

从 sysfs 中获取有意思信息的概念在之前的例子中已经触摸到了．为了基于这些信息书写规则,你首先需要知道属性名和他们的当前值.
sysfs 实际上是一个非常简单的结构,逻辑上以目录形式区分．每一个目录包含一定量的文件(属性),这些文件往往都仅仅包含一个值．也会有一些符号链接,它们把设备链到双亲那里，级联结构已在上面说明了.
一些目录被引向顶层设备路径,这些目录展示了拥有对应设备节点的实际设备．顶层设备路径可以被分类为包含 dev 文件的 sysfs 目录，下列命令可以列举出这些文件:
#find /sys -name dev
例如，在我的系统中, /sys/block/sda 目录就是我的硬盘设备路径，它通过/sys/block/sda/device 符号链接链向它的双亲 SCSI 磁盘设备.
但你书写基于 sysfs 信息的规则时，只是简单的匹配链条上一部分文件的属性内容．例如，我可以这样读我的硬盘的大小:
#cat /sys/block/sda/size
234441648
在一个 udev 规则里面，我可以使用 ATTR{size}=="234441648"来识别这个磁盘．因为 udev 反复遍历设备链的入口,我可以通过 ATTRS 匹配另外链中的属性(如:/sys/class/block/sda/device 属性)，但是在处理链中不同部分时会有一些告诫，稍后描述.
虽然这对于关于 sysfs 的结构和 udev 如何匹配值的介绍很有用，但对 sysfs 的全面梳理是个既耗时也没必要的事.
udevinfo
敲入 udevinfo 大概就是你用来创建规则的最直接的工具了。你需要知道的全部就是设备的 sysfs 设备路径．下面是一个精简的例子:
udevinfo -a -p /sys/block/sda
looking at device '/block/sda':
KERNEL=="sda"
SUBSYSTEM=="block"
ATTR{stat}==" 128535 2246 2788977 766188 73998 317300 3132216 5735004 0 516516 6503316"
ATTR{size}=="234441648"
ATTR{removable}=="0"
ATTR{range}=="16"
ATTR{dev}=="8:0"
looking at parent device '/devices/pci0000:00/0000:00:07.0/host0/target0:0:0/0:0:0:0':
KERNELS=="0:0:0:0"
SUBSYSTEMS=="scsi"
DRIVERS=="sd"
ATTRS{ioerr_cnt}=="0×0"
ATTRS{iodone_cnt}=="0×31737"
ATTRS{iorequest_cnt}=="0×31737"

ATTRS{iocounterbits}=="32"
ATTRS{timeout}=="30"
ATTRS{state}=="running"
ATTRS{rev}=="3.42"
ATTRS{model}=="ST3120827AS "
ATTRS{vendor}=="ATA "
ATTRS{scsi_level}=="6"
ATTRS{type}=="0"
ATTRS{queue_type}=="none"
ATTRS{queue_depth}=="1"
ATTRS{device_blocked}=="0"
looking at parent device '/devices/pci0000:00/0000:00:07.0':
KERNELS=="0000:00:07.0"
SUBSYSTEMS=="pci"
DRIVERS=="sata_nv"
ATTRS{vendor}=="0×10de"
ATTRS{device}=="0×037f"
正如你看到的, udevinfo 简单的产生一个你可以在 udev 规则中作为匹配键的属性列表. 从上面例子中我可以为该设备产生下面两条规则:
SUBSYSTEM=="block", ATTR{size}=="234441648", NAME="my_hard_disk"
SUBSYSTEM=="block", SUBSYSTEMS=="scsi", ATTRS{model}=="ST3120827AS", NAME="my_hard_disk"
你可能发现到例子中使用了颜色, 这是为了阐述把设备属性和单个双亲设备属性并在一起是合法的, 你不能混合匹配多个双亲设备属性,这样你的规则不能工作. 例如,下列规则是不合理的,因为它试图匹配两个双亲设备属性:
SUBSYSTEM=="block", ATTRS{model}=="ST3120827AS", DRIVERS=="sata_nv", NAME="my_hard_disk"
通常有大量的属性提供给你,你必需挑选其中一些来创建你的规则. 一般来讲,你希望挑选那些能够固定标志你的设备并便于理解的属性. 上例中我选取的是我的磁盘大小和它的模型号,而没有使用没有意义的诸如ATTRS{iodone_cnt}=="0×31737"数字.
仔细观察下 udevinfo 的输出级联结构效果, 设备的绿色部分使用了标准匹配键如 KERNEL 和 ATTR, 蓝色部分和栗色部分使用了双亲遍历变量如 SUBSYSTEMS 和 ATTRS, 这就是为什么级联结构的复杂性实际上是很容易处理的原因,只要确保使用 udevinfo 建议的准确值就行了.
另外一点需要指出的是 udevinfo 输出的文本属性之间用空格填充(例如上面的 ST3120827AS)是很常见的. 在你的规则中你可以添加额外的空格,也可以像我那样去掉.
使用 udevinfo 的唯一复杂之处在于要求你知道顶级设备路径(例如上面例子中的/sys/block/sda), 这通常并不明显. 但是, 因为你通常是为已经存在的设备节点写规则, 你可以自己使用 udevinfo 查找设备路径:
#udevinfo -a -p $(udevinfo -q path -n /dev/sda)

可选方法

虽然 udevinfo 差不多是列举你要构建的规则的准确属性的最直接方式, 但一些用户仍然乐于使用其他工具, 诸如 usbview 的工具可以显示类似的信息集, 这些信息也可以用在规则中.

高级话题

控制权限和所有权

udev 允许你在规则中使用另外的赋值来控制每个设备的所有权和权限属性.
GROUP 赋值允许你定义哪个 Unix 组应该拥有设备节点. 这里有一个例子规则, 它定义 video 组拥有 framebuffer 设备:

KERNEL=="fb[0-9]*", NAME="fb/%n", SYMLINK+="%k", GROUP="video"
OWNER 键可能用处不大，它允许你定义哪个 Unix 用户应该具有设备节点的拥有权限．假设有个临时情况要你让 join 拥有软盘设备,你可以使用:
KERNEL="fd[0-9]*", OWNER="join"
udev 缺省用 Unix 的 0660 权限(拥有者和组员拥有读写功能)创建设备节点．需要的话你可以在特定设备的规则中使用包含 MODE 赋值键覆盖这些缺省值．作为例子,下面的规则定义了 inotify 节点可以被每个人读写:
KERNEL=="inotify", NAME="misc/%k", SYMLINK+="%k", MODE="0666"

使用外部程序来命名设备

某些情况下你可能要求比udev标准规则提供的更多弹性，这种情况下你可以请求udev运行一个程序并运用程序的标准输出来提供设备命名.
要使用这个功能,你只需简单的在 PROGRAM 赋值中指定要运行程序(以及任何阐述)的完整路径，然后在 NAME/SYMLINK 赋值中使用一些%c 替换.
下列例子引用一个位于/bin/device_namer 的虚构程序. device_namer 带一个表示内核名字的命令行参数，基于内核名 device_namer 做一些魔幻变换接着产生一些输出到普通 stdout 管道,这些输出被分割为很多小块，每一小块是一个单词,块之间用一个空格分开.
在我们的第一个例子中，我们假设 device_namer 输出块的数目，每一个形成当前设备的一个符号链接(名字可选).
KERNEL=="hda", PROGRAM="/bin/device_namer %k", SYMLINK+="%c"
下一个例子假设device_namer输出两块 ,第一块是设备名字，第二块是另外的符号链接名字．我们现在介绍%c[N]替换，它引向输出的第 N 块:
KERNEL=="hda", PROGRAM="/bin/device_namer %k", NAME="%c{1}", SYMLINK+="%c{2}"
再下个例子假设 device_namer 输出设备名的一部分，后面跟着的是快数，它将形成另外的符号链接．我们现在介绍%c{N+}替换，它将被计算为块 N, N+1, N+2...直到输出结束.
KERNEL=="hda", PROGRAM="/bin/device_namer %k", NAME="%c{1}", SYMLINK+="%c{2+}"
输出块也可在赋值键中使用,而不仅仅是 NAME 和 SYMLINK 中．下面例子使用一个虚构程序来决定哪个 Unix 组拥有设备:
KERNEL=="hda", PROGRAM="/bin/who_owns_device %k", GROUP="%c"

特定事件发生时运行外部程序

另外一个书写 udev 规则的原因是为了在设备连接或者断开时运行一个特定程序．例如，你可能想在你的数码相机连到系统时执行一个脚本来自动下载相机里面的所有照片.
不要把这个跟上述的 PROGRAM 功能弄混淆了, PROGRAM 是用来运行产生设备名字(除此之外不应该做其他事情)的程序．但这些程序执行时，设备节点设备节点还没有被创建，所以对设备的任何形式的操作都是不可能的.
这里介绍的功能允许你在设备节点到位后运行一个程序．该程序可以作用在设备上，但是它不准在时间周期外运行,因为当程序运行时 udev 会正常中止．这个限制的一个权宜之计是确保你的程序立即分离自身.
这里有个展示 RUN 赋值的例子:
KERNEL=="sdb", RUN+="/usr/bin/my_program"
当/usr/bin/my_program 执行时, udev 环境的各部分可作为环境变量可用,包括诸如 SUBSYSTEM 的键值．你也可以使用 ACTION 环境变量来检测设备是否连接或断开，它的值是"add"和"remove"其中的一个.
udev 并不在任何激活终端中运行这些程序,也不再 shell 上下文中执行．确信你的程序是被标记为可执行的，如果它是个 shell 脚本请确保它以适当的 shabang 开头(比如: #!/bin/sh)也不要期望任何标准输出出现在你的终端上.

环境交互

udev 为环境变量提供了一个 ENV 键，即可用来匹配也可用来赋值.
在赋值情况下,你可以设置稍后匹配的环境变量．你也可以设置环境变量,供通过上面提供的技巧触发的任何外部程

序使用. 一个虚构的设置环境变量的例子规则如下:
KERNEL=="fd0", SYMLINK+="floppy", ENV{some_var}="value"
在匹配情况下你可以确保规则仅仅依赖一个环境变量的值而运行. 注意 udev 看到的环境跟你在控制台上得到的用户环境不一样. 一个虚构的涉及环境匹配的规则如下:
KERNEL=="fd0", ENV{an_env_var}=="yes", SYMLINK+="floppy"
上面规则仅仅在 udev 环境中的$an_env_var 的值设为"yes"时才创建/dev/floppy 链接.

另外选项

另外一个有用的赋值是 OPTIONS 列表. 不多的可用的选项有:
all_partitions - 为一个块设备创建所有可能的分区, 而不是初始检测到的那些分区.
ignore_device - 完全忽略事件.
last_rule - 确保后面的所有规则不会有效.
例如, 下面规则设置我的硬盘节点的组所有权,并且后面的规则对它没有任何效果:
KERNEL=="sda", GROUP="disk", OPTIONS+="last_rule"

例子

USB 打印机

我启动我的打印机, 它就被赋予了一个设备节点/dev/lp0. 我对这样的单调的名字不满意并打算使用 udevinfo 帮我写一个规则来提供一个可选名字:
udevinfo -a -p $(udevinfo -q path -n /dev/lp0)
looking at device '/class/usb/lp0':
KERNEL=="lp0"
SUBSYSTEM=="usb"
DRIVER=""
ATTR{dev}=="180:0"
looking at parent device '/devices/pci0000:00/0000:00:1d.0/usb1/1-1':
SUBSYSTEMS=="usb"
ATTRS{manufacturer}=="EPSON"
ATTRS{product}=="USB Printer"
ATTRS{serial}=="L72010011070626380"
我的规则变成了这样:
SUBSYSTEM=="usb", ATTRS{serial}=="L72010011070626380", SYMLINK+="epson_680"

USB 相机

跟大多数相机一样, 我的相机标识自己为一个通过 USB 总线来连接通过 SCSI 传输的外部硬盘. 为了访问我的相片我先挂载驱动然后拷贝图片文件到我的硬盘中.
不是所有相机都这样工作, 其中一些使用非存储协议,如 gphoto2 支持的相机. 在 gphoto 情况下,你不用为你的设备写任何规则,因为纯粹由用户空间控制(而不是指定的内核驱动).
USB 相机设备的一个通常的复杂性在于它们通常标识自己是一个单分区磁盘,即带有/dev/sdb1 的/dev/sdb. sdb 节点对我毫无用处,但对 sdb1 有兴趣 - 这才是我要挂载的. 这里有个麻烦因为 sysfs 是链式的, udevinfo 为/dev/sdb1 产生的有用属性跟为/dev/sdb 产生的一样, 这导致你的规则潜在的匹配到原始磁盘和分区, 这不是你想要的, 你的规则应该明确下.
为解决这个问题,你需要简单的考虑下 sdb 和 sdb1 之间有什么区别. 令人惊奇的简单: 只是名字上的区别, 所以我们可在 NAME 域上使用一个简单的模式匹配.
udevinfo -a -p $(udevinfo -q path -n /dev/sdb1)

looking at device '/block/sdb/sdb1':
KERNEL=="sdb1"
SUBSYSTEM=="block"
looking at parent device '/devices/pci0000:00/0000:00:02.1/usb1/1-1/1-1:1.0/host6/target6:0:0/6:0:0:0':
KERNELS=="6:0:0:0"
SUBSYSTEMS=="scsi"
DRIVERS=="sd"
ATTRS{rev}=="1.00"
ATTRS{model}=="X250,D560Z,C350Z"
ATTRS{vendor}=="OLYMPUS "
ATTRS{scsi_level}=="3"
ATTRS{type}=="0"
我的规则:
KERNEL=="sd?1", SUBSYSTEMS=="scsi", ATTRS{model}=="X250,D560Z,C350Z", SYMLINK+="camera"

USB 硬盘

USB 硬盘跟 USB 相机差不多,但典型的使用方式不同. 在相机例子中, 我有讲到我对 sdb 节点没有兴趣, 它仅仅在分区(比如使用 fdisk)时才有用, 但我为什么要为我的相机分区呢?
当然如果你有一个 100GB 的 USB 硬盘, 你希望为它分区是很好理解的, 这种情况下我们可以使用 udev 的字符串替换长处:
KERNEL=="sd*", SUBSYSTEM=="scsi", ATTRS{model}=="USB 2.0 Storage Device", SYMLINK+="usbhd%n"
这个规则创建诸如下面的符号链接:
/dev/usbhd - 可被 fdisk 使用的 node
/dev/usbhd1 - 第一块分区(可挂载)
/dev/usbhd2 - 第二块分区(可挂载)

USB 读卡器

USB 读卡器(CompactFlash, SmartMedia 等)属于 USB 存储设备的另外一种范围, 它有不同的使用需求.
这些设备特别的在媒介改变时不用通知主机. 所以如果你插入没有媒介的设备,接着插入一张卡, 计算机不会意识到这点, 你也不会有媒介的可挂载的 sdb1 分区节点.
一个可能的解决办法是使用 all_partttions 选项优点, 它将为每个规则匹配的块设备创建 16 个分区节点:
KERNEL="sd*", SUBSYSTEM=="scsi", ATTRS{model}=="USB 2.0 CompactFlash Reader", SYMLINK+="cfrdr%n", OPTIONS+="all_partitions"
你将得到这些命名节点：cfrdr, cfrdr1, cfrdr2, cfrdr3, ..., cfrdr15.

USB Palm 导航仪

这些设备作为 USB 串行设备工作, 所以缺省的你仅仅得到 ttyUSB1 设备节点. palm 工具依赖于/dev/pilot, 一些用户希望使用一条规则提供这个.
Carsten Clasohm 的博客上有完整的源. Carsten 的规则如下:
SUBSYSTEM=="usb", ATTRS{product}=="Palm Handheld", KERNEL=="ttyUSB*", SYMLINK+="pilot"
注意到产品字符串因产品不同而不同, 所以确保你检查(使用 udevinfo)了哪一个可以应用到你自己的产品.

CD/VCD 驱动

我的电脑有两个光驱: 一个 DVD 只读驱动(hdc)和一个 DVD 刻录机(hdd). 我不希望这些设备节点改变除非我物理

性的重新接线，但是一些用户喜欢拥有诸如/dev/dvd 这么方便的设备节点.
我们都知道 KERNEL 是这些设备的名字，规则书写就很简单了．这是我的系统上一些规则:
SUBSYSTEM=="block", KERNEL=="hdc", SYMLINK+="dvd", GROUP="cdrom"
SUBSYSTEM=="block", KERNEL=="hdd", SYMLINK+="dvdrw", GROUP="cdrom"

网卡

尽管它们都是通过名字引用，网卡往往没有与之关联的设备节点．尽管这样,规则书写过程还是相同的.
在规则中简单的匹配网卡 MAC 地址是有意义的,因为它们是唯一的．但是，确信你使用的是 udevinfo 显示的准确 MAC 地址，因为如果你没有精确匹配,你的规则不会工作.
udevinfo -a -p /sys/class/net/eth0
looking at class device '/sys/class/net/eth0':
KERNEL=="eth0"
ATTR{address}=="00:52:8b:d5:04:48"
这是我的规则:
KERNEL=="eth*", ATTR{address}=="00:52:8b:d5:04:48", NAME="lan"
为了让这个规则生效你得重启网络驱动，你可以卸载并重新加载模块或简单的重启系统即可．你还得需要重新配置你的系统使用"lan"取代"eth0"．我以前遇到过这样的麻烦(网卡不能重命名)直到我去除了所有对 eth0 的引用．在此之后你应该可以在任何 ifconfig 或类似的工具的调用中使用"lan"而不是"eth0"了.

测试和调试

让你的规则跑起来

假定你在一个有 inotify 支持的最近内核上工作, udev 将自动监视你的规则目录并且自动挑取你对规则文件的任何修改.
尽管这样, udev 也不会自动重新处理所有设备并试图应用新规则．例如，如果你写了个规则来为你的已连接相机添加一个额外符号链接，你不能指望这个符号链接可以马上显现出来.
为使符号链接显示，你可以断开并重连你的相机或者在非可移除设备情况下运行 udevtrigger.
如果你的内核没有 inotify 支持，新规则不会自动被检测到．这种情况下你必需在做出更改后运行 udevcontrol reload_rules 使之生效.

udevtest

如果你知道 sysfs 中的顶级设备路径，你可以使用 udevtest 来显示 udev 将要执行的动作，这可能会帮你调试你的规则．例如，假设你想调试作用在/sys/class/sound/dsp 上的规则:
udevtest /class/sound/dsp
main: looking at device '/class/sound/dsp' from subsystem 'sound'
udev_rules_get_name: add symlink 'dsp'
udev_rules_get_name: rule applied, 'dsp' becomes 'sound/dsp'
udev_device_event: device '/class/sound/dsp' already known, remove possible symlinks
udev_node_add: creating device node '/dev/sound/dsp', major = '14', minor = '3', mode = '0660', uid = '0', gid = '18'
udev_node_add: creating symlink '/dev/dsp' to 'sound/dsp'
注意/sys 前缀在 udevtest 命令行中被删除了，这是因为 udevtest 在设备路径上操作．还要留意的是 udevtest 是一个纯粹的测试/调试工具，它不创建任何设备节点无论输出怎么建议.