精通 RPM 之认识篇

RPM 是 RedHat Package Manager 的缩写,意即 RedHat(红帽子)软件包管理器。(RedHat 是美国有名的 LINUX 公司,网址为 http://www.redhat.com)

对于一个操作系统来说,不能没有一个象样的软件包管理器。没有软件包管理器的帮助,操作系统发行版的制作者将面临这样或那样的难题,用户安装,升 级,卸载与发布软件包也将是非常麻烦的,系统管理也容易出现问题。相反,有了专门的软件包管理器,软件制作者易于制作和发行自己的软件了,而对于普通用户 来说,软件包的安装维护将变得非常方便了。这种情况,对于一个操作系统的推广也会起到良好的促进作用。RPM 就是随着 RedHat LINUX 发行版的流行而迅速推广开来的,二者的表现相得易彰。

RPM 先行者

最初的时候,LINUX 系统的发布并没有使用什么软件包管理器。

随着时间的推移,RedHatmLINUX 开发者意识到开发一个软件包管理器的重要性,于是开发出 RPP 这个管理器。RPP 相对于 RPM 虽然是简单 的,但已有了几项重要的功能,如打一个简单的命令就可以实现软件的安装与卸载,包中可含有安装前后与卸载前后执行的脚本程序,还可以随时校验已安装的软件 包,查询功能也很强大。RPP 的缺点在于,RPP 打包是基于特别修改过的(针对 RPP)源代码的,因而这些源代码并非是纯正的源代码。由于这个原因,当软件包开发者想建立大量不同的软 件包时,将面临众多技术面的问题。RPP 也不能保证当前的执行程序是基于打包过的源程序的,并且 RPP 不支持多处理器体系结构。

与 RPP 同时开发的,还有 PMS(即 package management system,软件包管理系统),这是另一群 LINUX 爱好者开发的。PMS 采用的是纯正的源代码,它允许软件包制作者很快释出一个软件的最新版本,并且 可以立刻看到该软件的变化。RPM 采用了这一明智的做法,这也是 PMS 对 RPM 的一项重大贡献。PMS 的缺点是查询功能不强,没有包校验功能,不支持多体 系结构,数据库设计也不好。

在 RPP 和 PMS 之后,Rik Faith 和 Doug Hoffman 开发了 PM 管理器。该管理器整合了 RPP 与 PMS 的许多功能,但是数据库设计还不强,依然不支持多体系结构。

RPM 开发

此后,Marc Ewing 和 Erik Troan 两人在吸取 RPP,PMS,PM 设计经验的基础上,用 PERL 语开发了 RPM 软件包管理器,即 RPM1.0 版。

其成功之处在于:

可自动处理配置文件;
可重建大量的软件包;

易于使用。
其不足之处在于:

程序大,运行速度慢,因为它是用 PERL 这种解释型的语言写的;
数据库功能太弱;
不支持多体系结构;
包裹文件格式不可扩展。
针对 RPM1.0 的弱点,RPM 的开发者再度努力,将 RPM 升级到 2.0,3.0 和现在的 4.0 版本。他们主要做了以下几点:

用 C 重写了程序,这极大地提高了 RPM 的运行速度。
RPM 数据库格式进行了再设计,重点从性能和可靠性两方面提高。
软件包格式也进行了再设计,方便以后的扩展和升级。
建立了 rpmlib(RPM 函数库),方便其它程序调用 RPM。
增加多体系支持,方便 RPM 跨平台使用(不论是 x86 体系,还是 sparc 等其它体系)。

网上的 RPM

RPM 官方网址为 http://www.rpm.org,该网站刊登有 RPM 的最新消息,有关于 RPM 的说明文档,还可下载最新的 RPM 软件源代码和执行代码,不妨常去看看。

小结

如今的 RPM 使得软件包安装与卸载更容易,校验已安装的软件包是否正常也容易,将程序(源程序或执行程序)打包也简单了,跨平台的支持,遵循 GPL 版权发布源代码,使得 RPM 得到更广泛的应用与推广。RPM 正在风靡 LINUX 与非 LINUX 世界。
如果你想了解甚至精通 RPM,那么请跟我来吧!

精通 RPM 之安装篇

从一般意义上说，软件包的安装其实是文件的拷贝，RPM 安装软件包，也无外乎此。但 RPM 要更进一步、更聪明一些就需要多做些工作了。

聪明的安装

从一般意义上说，软件包的安装其实是文件的拷贝，即把软件所用的各个文件拷贝到特定目录。RPM 安装软件包，无外乎此。但 RPM 要更进一步，更聪明一些。在安装前，它通常要执行以下操作：

1.检查软件包的依赖（Dependency）

RPM 格式的软件包中可包含有依赖关系的描述，如软件执行时需要什么动态链接库，需要什么程序存在及版本号要求等。当 RPM 检查时发现所依赖的链接库或程序等不存在或不符合要求时，默认的做法是中止软件包安装。

2.检查软件包的冲突（Conflicts）

有的软件与某些软件不能共存，软件包制作者会将这种冲突记录到 RPM 软件包中。安装时，若 RPM 发现有冲突存在，将会中止安装。

3.执行安装前脚本程序（Preinstall）

此类程序由软件包制作者设定，需要在安装前执行。通常是检测操作环境，建立有关目录，清理多余文件等等，为顺利安装作准备。

4.处理配置文件（Configfiles）

RPM 对配置文件（Configfiles）有着特别的处理。因为用户常常需要根据实际情况，对软件的配置文件做相应的修改。如果安装时简单地覆盖了此类文件， 则用户又得重新手工设置，很麻烦。这种情况下，RPM 做得比较明智：它将原配置文件换个名字保存了起来（原文件名后缀加上.rpmorig），用户可根据 需要再恢复，避免重新设置的尴尬。

5.解压软件包并存放到相应位置

这是最重要的部分，也是软件包安装的关键所在。在这一步，RPM 将软件包解压缩，将其中的文件一个个存放到正确的位置，同时，文件的操作权限等属性相应设置正确。

6.执行安装后脚本程序（Postinstall）

此类程序为软件的正确执行设定相关资源，如修改 inetd.conf、运行 ldconfig 程序以利新的动态链接库生效等等。

7.更新 RPM 数据库

安装后，RPM 将所安装的软件及相关信息记录到其数据库中，便于以后升级、查询、校验和卸载。

8.执行安装时触发脚本程序（Triggerin）

触发脚本程序是指软件包满足某种条件（如已安装软件包 sendmail，或 file 版本大于 3.0）时才触发执行的脚本程序，它用于软件包之间的交互控 制。触发脚本程序有三类：一是软件包安装时才触发的，称为安装时触发脚本程序（triggerin）；二是软件包卸载前触发的，叫作卸载前触发脚本程序（triggerun）；三是软件包卸载后才触发执行的，称作卸载后触发脚本程序（triggerpostun）。这些触发脚本程序，大大扩展了 RPM 软 件包管理的功能。

命令格式

安装 RPM 格式的软件包，可使用如下命令格式：

rpm -i [安装选项 1 安装选项 2...] 包裹文件 1 包裹文件 2...

注：可用--install 代替-i，效果相同。

选项列表（见选项详解）

包裹文件

对于要安装的 RPM 格式的包裹文件，RPM 对其名字不作强制要求。用户可以使用以下三种方式的命名样式：

1.典型的命名样式（常用）：

格式为：软件名-版本号-释出号.体系号.rpm

注：体系号指的是执行程序适用的处理器体系，如 i386 体系，sparc 体系等。体系号为 src 时表明为源代码包，否则为执行程序包。

如 abc-3.2-1.i386.rpm 为执行程序包，软件名为 abc，版本号为 3.2，释出号为 1，适用体系为 i386，而 abc-3.2-1.src.rpm 则为源代码包。

2.URL 形式的命名样式（较常用）：

* FTP 方式的命名格式：ftp://[用户名[:密码]@]主机[:端口]/包裹文件

注： []括住的内容表示可选。主机可以是主机名，也可是 IP 地址。包裹文件可含目录信息。如未指定用户名，则 RPM 采用匿名方式传输数据（用户名为 anonymous）。如未指定密码，则 RPM 会根据实际情况提示用户输入密码。如未指定端口，则 RPM 使用默认端口（一般为 21）。

如 ftp://ftp.xxx.com/yyy.rpm（使用匿名传输，主机 ftp.xxx.com，包裹文件 yyy.rpm）；
如 ftp://24.109.164.55:1024/pub/yyy.rpm（匿名 FTP 传输，主机 IP:24.109.164.55，使用 1024 端口，包裹文件在/pub 目录下）；
如 ftp://zhsoft@ftp.xxx.com/yyy.rpm(主机 ftp.xxx.com,FTP 用户名 zhsoft，如有密码，RPM 将会自动提示输入）；
如 ftp://zhsoft:password@ftp.xxx.com/yyy.rpm（主机 ftp.xxx.com，FTP 用户名 zhsoft，密码 password）。

* HTTP 方式的命名：http://主机[:端口]/包裹文件

注：[]括住的内容可选。主机可以是主机名，也可是 IP 地址。包裹文件可含目录信息。如未指定端口，则 RPM 默认使用 80 端口。

如 http://www.xxx.com/yyy.rpm（用 HTTP 获取 www.xxx.com 主机上的 yyy.rpm 文件）；
又如 http://www.xxx.com:8080/pub/yyy.rpm（用 HTTP 获取 www.xxx.com 主机上/pub 目录下的 yyy.rpm 文件，使用端口 8080）。

3.其它形式（很少使用）：

命名格式：任意

如将 abc-3.2-1.i386.rpm 改名为 abc.txt，用 RPM 安装也会安装成功，其根本原因是 RPM 判定一个文件是否 RPM 格式，不是看名字，而是看内容，看其是否符合特定的格式。

选项详解

一、安装指定用选项

1.hash（或-h）：以#显示安装进度

如果一个软件包很大、安装费时时，用户若想及时了解安装进度，必须用此选项。该选项以显示#号表示进度，每个#号表示 2%的进度，总共要显示 50 个#号。下面安装 MySQL 软件包，如下所示：

```
# rpm -i --hash MySQL-3.22.32-1.i386.rpm
########################################################
#
```

从上看出，软件包安装顺利完成。

2.percent：以%（百分比）显示安装进度

percent 的含义是百分比，而此选项的作用就是以百分比（%）来显示安装进度的。同样是安装 MySQL 软件包，下面的输出就不一样：

```
# rpm -i --percent MySQL-3.22.32-1.i386.rpm
%% 0.000000
%% 0.002600
%% 0.020586
......
%% 100.000000
```

#

上例中省略了很多输出（.....表示）。因为如果软件包很大，则用--percent 时输出的内容会很多，所以用户最好用--hash 选项，以#号来表示安装进度，这样简洁明了。

3.test：安装测试

所谓安装测试，意即并非真正的安装，它不拷贝和建立任何文件。使用本选项的目的在于：检测软件包的依赖关系是否满足，是否存在潜在的冲突等等。

```
# rpm -i --test autofs-3.1.3-2.i386.rpm
error: failed dependencies:
mktemp is needed by autofs-3.1.3-2
#
```

本例进行安装测试时发现了依赖方面的错误,autofs 所需要的 mktemp 包不存在，安装中断。

下面先安装 mktemp 软件包，再进行 autofs 安装测试，看情况怎么样：

```
# rpm -i mktemp-1.5-2.i386.rpm
# rpm -i --test autofs-3.1.3-2.i386.rpm
#
```

由上看出，这次测试没有出现什么错误。

4.replacepkgs：替换软件包

为什么要替换软件包？原因可能是系统中的软件包已经破坏了,其中一个或多个文件丢失或损毁。如果用户想修复这个软件包，用直接安装的方法，RPM 将报错退出：

```
# rpm -V file
missing/usr/man/man1/file.1
# rpm -i file-3.26-6.i386.rpm
package file-3.26-6 is already installed
#
```

注：本例先用 RPM 校验命令校验一下 file 软件包，发现/usr/man/man1/file.1 文件丢失。之后用安装命令安装，RPM 提示软件包 file-3.26-6 已经安装了。（有关 RPM 校验命令，<<精通 RPM 之校验篇>>中将会有详细的介绍）

如果采用--replacepkgs 选项，结果又怎样呢？

```
# rpm -i --replacepkgs file-3.26-6.i386.rpm
# rpm -V file
#
```

看来，RPM 成功地替换了原软件包，校验一下该包，发现没有错误输出，所以现在的软件包是完整的。

5.replacefiles：替换文件

RPM 是聪明的软件包管理器，它维护着每个已安装软件包的文件信息。如果在安装一个新的软件包时，RPM 发现其中某个文件和已安装的某个软件包中的文件名字相同但内容不同，那么 RPM 就会认为这是一个文件冲突，会报错退出：

```
# rpm -i ff-4.0-2.i386.rpm
file /root/my.a from install of ff-4.0-2 conflicts with file from package zoo-6.0-1
#
```

注：本例中 RPM 发现要安装的软件包 ff-4.0-2 与已安装的软件包 zoo-6.0-1 中，含有相同的一个文件/root/my.a，但其内容并不相同，所以提示了文件冲突的错误。

如果用户想忽略这个错误，可使用--replacefiles 选项，指示 RPM 发现文件冲突时，直接替换掉原文件即可。注意：除非用户对所冲突的文件有很深的了解，不要轻易替换文件，以免破坏已安装软件包的完整性，确保其能正常运行。

```
# rpm -i --replacefiles ff-4.0-2.i386.rpm
#
```

采用该选项后，软件包能顺利安装了。

另外，说到替换文件，若要安装的软件包中的文件已存在，但此文件并不属于任何软件包，RPM 的做法是将文件换名保存（文件名后缀加.rpmorig），并且以警告信息提醒用户。如下所示：

```
# rpm -i foo-6.0-1.i386.rpm
warning: /etc/foo.conf saved as /etc/foo.conf.rpmorig
#
```

6.allfiles：安装所有文件

读者看到此选项，也许要问：难道 RPM 安装软件包不是安装其中所有的文件吗？

我的回答是：如果是初次安装的话，RPM 确是将包中所有文件全部安装。但是，如果是修复软件包（用--replacepkgs 选项），那结果就不一定了。 个中原因是：RPM 包中有些配置文件可标识为 missingok 属性（missingok 指的是即使丢失，照样 OK），这样的包安装后，若这种类型的配置 文件被删除，则修复时 RPM 默认的做法是不再安装这种类型的文件，除非采用--allfiles 选项。下面看个实际的例子：

```
# rpm -i foo-6.0-1.i386.rpm
# ls -l /etc/foo.conf
-rw-r--r-- 1 root root9 Oct 11 09:50 /etc/foo.conf
# rm -f /etc/foo.conf
# rpm -i --replacepkgs foo-6.0-1.i386.rpm
# ls -l /etc/foo.conf
ls: /etc/foo.conf: 文件或目录不存在
# rpm -i --replacepkgs --allfiles foo-6.0-1.i386.rpm
# ls -l /etc/foo.conf
-rw-r--r-- 1 root root9 Oct 11 09:50 /etc/foo.conf
```

注:本例中已预先知道 foo 包中的配置文件/etc/foo.conf 带有 missingok 属性。

（1）用 rpm -i 命令安装 foo 包；

（2）用 ls 命令列一下属于该包的配置文件/etc/foo.conf（能列出来，表明文件存在）；

（3）用 rm 命令删除了这个文件；

（4）用 rpm -i --replacepkgs 命令修复 foo 软件包；

（5）因为修复时未用--allfiles 选项，所以用 ls 命令列文件/etc/foo.conf 时出错了：文件不存在，没有安装上；

（6）用 rpm -i --replacepkgs --allfiles 命令安装修复 foo 软件包；

（7）再度用 ls 命令列文件，列了出来，表明这次安装上了。

由本例看出，若用户确实想安全恢复（修复）某个软件包，最好使用--allfiles 选项。一般情况下则不必这么做，因为带有 missingok 属性的配置文件本来就是可以丢失的嘛，不必太在意了。

7.force：强制执行

force 的含义是强制。--force 选项的作用就是强制安装软件包，不考虑软件包是否已安装，也不考虑有没有文件冲突。其效果相当于同时选用--replacepkgs

与--replacefiles 选项进行安装。

8.excludedocs：不安装说明文档

RPM 有多个好功能，其中之一就是：它将文件分为配置文件，说明文档和其它文件三种，这样便于区别对待，灵活处理。

基于 RPM 安装的 LINUX 发行版中包括 5000 多个说明文档，有 50M 字节，占用的空间不小。如果想节省空间，可使用--excludedocs 选项以排除安装说明文档。

如软件包 file 中包含/usr/man/man1/file.1 和/usr/man/man4/magic.4 两个说明文档。安装时若使用--excludedocs 选项，这两个文档就不会被安装。

```
# rpm -i --excludedocs file-3.26-6.i386.rpm
# ls -l /usr/man/man1/file.1
ls: /usr/man/man1/file.1: 文件或目录不存在
# ls -l /usr/man/man4/magic.4
ls: /usr/man/man4/magic.4: 文件或目录不存在
#
```

如果用户想让 RPM 默认不安装说明文档，则可以这么做：编辑~/.rpmmacros 文件（用户主目录 HOME 下的 RPM 宏文件），加入下面一行：

```
%_excludedocs 1
```

其作用是定义 RPM 内部的_excludedocs 宏为 1，确认不安装说明文档。

这样的话，就不用在命令行使用--excludedocs 选项了。

```
# rpm -i file-3.26-6.i386.rpm
# ls -l /usr/man/man1/file.1
ls: /usr/man/man1/file.1: 文件或目录不存在
# ls -l /usr/man/man4/magic.4
ls: /usr/man/man4/magic.4: 文件或目录不存在
#
```

9.includedocs：安装说明文档

RPM 一般情况下是安装说明文档的，但是如果用户自行修改了设定（象上面的例子），则 RPM 就不再安装说明文档了，除非特别指定，这就用到--includedocs 选项。这种情况下，只有用此选项才能确保安装说明文档。

```
# rpm -i --includedocs file-3.26-6.i386.rpm
# ls -l /usr/man/man1/file.1
```

```
-rwxr-xr-x 1 root root12023 Mar 23 1999 /usr/man/man1/file.1
# ls -l /usr/man/man4/magic.4
-rwxr-xr-x 1 root root 6625 Mar 23 1999 /usr/man/man4/magic.4
#
```

10.noscripts：不执行脚本程序

一个 RPM 软件包中可包含五种脚本程序，即：安装前脚本程序，安装后脚本程序，卸载前脚本程序，卸载后脚本程序和校验脚本程序。安装时使用--noscripts 选项，可禁止安装前与安装后脚本程序的执行。

```
# rpm -i foo-6.0-1.i386.rpm
preinstall is running ... done
postinstall is running ... done
#
```

上面不禁止脚本程序的执行，下面不执行脚本程序，请看输出结果：

```
# rpm -i --noscripts foo-6.0-1.i386.rpm
#
```

看，因为没有安装前与安装后脚本程序的执行，所以没有什么输出了。

注意：一般用户不要使用此选项。此选项主要提供给软件包制作者使用的。通过禁止执行脚本程序，可以防止因安装带有 BUG 的软件包而宕掉整个系统。当软件包去掉了 BUG 后，此选项就不必使用了。

11.nodeps：不检查依赖

RPM 管理软件包，不仅管理包中的所有文件，还同时管理着软件包之间的依赖关系。如 A 依赖于 B 运行，若 B 不存在了，则 A 也就运行不了了。RPM 维护着这种关系，尽量避免破坏，以保证软件的正常运行。

```
# rpm -i autofs-3.1.3-2.i386.rpm
error: failed dependencies:
mktemp is needed by autofs-3.1.3-2
#
```

本例安装中出现了依赖方面的错误（autofs 依赖 mktemp，但 mktemp 不存在），安装过程中断了。如果要 RPM 不管依赖关系是否正常都安装，就要用--nodeps 选项，指示 RPM 不检查依赖，这样就能正常安装了。

```
# rpm -i --nodeps autofs-3.1.3-2.i386.rpm
#
```

注：除非用户对软件包有足够了解，或只想看看软件，否则不要使用本选项，以维护软件间正常的依赖关系。

12.notriggers：不执行触发程序

为了软件包间的交互控制，RPM 设计了三种触发程序：安装时触发程序，卸载前触发程序和卸载后触发程序。为了防止因某一软件包的安装而引发安装时触发程序的执行，可以使用--notriggers 选项。

13.ignorearch：忽略体系与 ignoreos ：忽略操作系统

何谓体系?体系就是 CPU 的类别，有 Intel 的 x86（如 i386，i486 系列）体系，有 Sun 的 sparc 体系等等。当一个软件包建立时，RPM 就 为其指定了所适用的 CPU 体系，也为其指定了所适用的操作系统。这样做的好处就是，RPM 容易知道为一台计算机所建立的软件包是否适用于兼容于另一台计算 机。RPM 的资源配置文件(默认为/usr/lib/rpm/rpmrc)中,就定义了体系的兼容关系(arch_compat 表达）和操作系统的兼容关 系（os_compat 表达）。RPM 安装一个包裹文件时，要做这样一个兼容方面的检查。如果用户想忽略体系，不管其是否兼容，请用--ignorearch 选项安装。如果也不管操作系统是否兼容，可用--ignoreos 选项。需要指出的是，若非知道自己这样的目的，否则不要试图这样 做。

14.ignoresize：不检查空间大小

RPM 安装软件包前，首先要检查当前系统是否有足够的剩余空间，如果空间不足的话，安装将无法完成。使用本选项的目的，在于指示 RPM 不做空间大小方面的检查，意即不管系统是否有容纳要安装的软件包的空间，照直安装就行了。

15.relocate：重定位

RPM 软件包在制作过程中，可以定义一个或多个重定位前缀，以此方便软件的重定位（即把软件包中的文件放到自定义的目录下面），从而增加软件包安装的灵活性。

本选项用于更换指定的重定位目录，如果一个包有多个重定位前缀，可以使用本选项多次。

```
# rpm -qpl file-3.26-6.i386.rpm
/usr/bin/file
/usr/man/man1/file.1
/usr/man/man4/magic.4
/usr/share/magic
# rpm -i --relocate /usr=/tmp file-3.26-6.i386.rpm
# rpm -ql file
```

```
/tmp/bin/file
/tmp/man/man1/file.1
/tmp/man/man4/magic.4
/tmp/share/magic
#
```

注：本例中先用 rpm -qpl 列出包裹文件 file-3.26-6.i386.rpm 当中的文件，可以看到其中的文件均是以/usr 开头的。之后进行重定位安装，将/usr 换 作/tmp。最后从列出已安装的 file 包的文件列表可以看出，原来的/usr 目录前缀换作指定的前缀/tmp 了，这样实现了软件的重定位。（关于 RPM 查询命令，详见<<精通 RPM 之五--查询篇>>）

16.badreloc：强制重定位

RPM 软件包的重定位，依赖于制作时重定位前缀的定义。如果没有定义重定位前缀，或者用户安装时所指定的重定位前缀不存在，则 RPM 会报错退出，中断安装。此 时，若用本选项，则 RPM 会不管这些错误，进行强制的重定位安装。注：本选项和--relocate 选项同时使用时才有意义。

如上例，采用并不存在的重定位前缀/usr/man 进行安装：

```
# rpm -i --relocate /usr/man=/tmp file-3.26-6.i386.rpm
path /usr/man is not relocateable for package file-3.26-6
#
```

RPM 提示了错误：/usr/man 目录在 file-3.26-6 包中是不可重定位的。

如加上--badreloc 选项，结果怎么样：

```
# rpm -i --relocate /usr/man=/tmp --badreloc file-3.26-6.i386.rpm
# rpm -ql file
/usr/bin/file
/tmp/man1/file.1
/tmp/man4/magic.4
/usr/share/magic
#
```

由上可以看到，这样的强制重定位获得通过，之后再用查询命令 rpm -ql 列一下 file 包的文件，就会发现原来的/usr/man/man1/file.1 定位成了 /tmp/man1/file.1，而 /usr/man/man4/magic.4 则成了/tmp/man4/magic.4。怎么样?神奇吧!

17.excludepath：不安装指定目录下的文件

利用此选项，可以禁止 RPM 安装某些指定目录下的文件。如不想安装 file 软件包中的说明文档，除了可使用--excludedocs 选项外，还可使用本选项。因为 file 的说明文档均在目录/usr/man 下面。

```
# rpm -i --excludepath /usr/man file-3.26-6.i386.rpm
# rpm -qls file
normal/usr/bin/file
not installed /usr/man/man1/file.1
not installed /usr/man/man4/magic.4
normal/usr/share/magic
#
```

注：例子中先安装 file 软件包，并用--excludepath 禁止安装/usr/man 下的文件，再用 RPM 查询命令 rpm -qls 列出 file 包的文件及状态，可以看到：/usr/man 下的两个文件均未安装（not installed）。

18.justdb：仅更新数据库

安装软件包使用此选项后，RPM 将只更新其数据库数据，文件系统不更新，意即并不拷贝和建立包中的文件。

```
# rpm -i --justdb file-3.26-6.i386.rpm
# rpm -qls file
normal/usr/bin/file
normal/usr/man/man1/file.1
normal/usr/man/man4/magic.4
normal/usr/share/magic
# ls -l /usr/bin/file
ls: /usr/bin/file: 文件或目录不存在
#
```

注：例子中安装 file 包但指明只更新数据库，结果是：虽然用查询命令查到该软件包中所有文件状态均正常（normal），但用 ls 命令列包中所含文件 /usr/bin/file，该文件却不存在。这表明包中文件并未建立。

19.prefix：指定重定位前缀

如前所述，可重定位的软件包可含一个或多个重定位前缀，可用--relocate 选项来改变某个重定位前缀之值。--prefix 选项仅能用来改变默认的重定位前缀（默认是第一个重定位前缀），它不需要说明原重定位前缀。下面的例子同样将 file 包安装到/tmp 目录：

```
# rpm -qpl file-3.26-6.i386.rpm
/usr/bin/file
```

```
/usr/man/man1/file.1
/usr/man/man4/magic.4
/usr/share/magic
# rpm -i --prefix /tmp file-3.26-6.i386.rpm
# rpm -ql file
/tmp/bin/file
/tmp/man/man1/file.1
/tmp/man/man4/magic.4
/tmp/share/magic
#
```

20.ftpproxy：指定 FTP 代理主机

本选项指定 FTP 代理主机，主机可以是主机名，也可以是 IP 地址。

21.ftpport：指定 FTP 端口

本选项指定 FTP 协议使用的 TCP 端口，系统默认是 21。

请看下面的例子：

```
# rpm -iv --ftpport 8888 ftp://root@xwboc/tmp/file-3.26-6.i386.rpm
Password for root@xwboc:
Password for root@xwboc:
Retrieving ftp://root@xwboc/tmp/file-3.26-6.i386.rpm
file-3.26-6
#
```

本例中用 FTP 方式安装 file 软件包，数据传输使用 8888 端口。执行中，RPM 先提示输入 root 用户的密码两次，之后下载（retrieve）软件，最后成功安装。

22.httpproxy：指定 HTTP 代理主机

本选项指定 HTTP 代理主机，主机可以是主机名，也可以是 IP 地址。

23.httpport：指定 HTTP 端口

本选项指定 HTTP 协议使用的 TCP 端口，系统默认是 80。

下面的例子就指定了 80 端口来安装 autofs 软件，当然不指定也行，因为系统默认的就是这个端口。

```
# rpm -iv --httpport 80
http://24.109.164.55/rpms/autofs-3.1.3-2.i386.rpm
```

```
Retrieving http://24.109.164.55/rpms/autofs-3.1.3-2.i386.rpm
autofs-3.1.3-2
#
```

注：同 FTP 方式一样，RPM 都是下载再安装。

二、通用选项

所谓通用选项，就是不管是安装，升级，卸载还是其它软件包操作，统统适用的选项。

这样的选项主要以下几个：

1.-v：显示附加信息

一般情况下，RPM 和不少 LINUX 命令（如 cp，rm，mv 等）一样，都本着尽可能少输出信息的原则（除非必要，否则不要输出），这样做是为了避免浪费 CPU 资源。所以，当用户使用某个命令后没有什么输出，这就意味着命令执行成功了。如果有什么错误，程序会提示的。RPM 也是这个样子。为了使 RPM 输出 多一点的信息，可以使用-v 选项。

如不带此选项安装 file 包，没有输出：

```
# rpm -i file-3.26-6.i386.rpm
#
```

而采用-v 选项后安装时输出了包名：

```
# rpm -i -v file-3.26-6.i386.rpm
file-3.26-6
#
```

2.-vv：显示调试信息

选用-vv 选项，可以使 RPM 输出更多的信息。这些信息，主要供 RPM 软件开发者研究使用的，一般用户也可以看看，从中可以知道 RPM 究竟干些什么。

下面采用此选项安装 file 包，同时使用管道线（|）将输出数据送 nl 命令，以带行号输出每行数据。

从执行结果看，输出的信息很多，其中以 D:开头的行均是调试信息。现在逐行解释：

第 1 行：统计要安装的软件包个数；

第 2 行：RPM 找到 1 个包；

第 3 行：查看哪些包需要下载（这些要下载的包，都是以 FTP 或 HTTP 形式书写的包名）；

第 4 行：下载了 0 个包；

第 5-10 行：取包裹文件的文件头信息，算出软件包占用的空间 68019（字节），以利检查

系统是否有足够空间。

第 11 行：打开 RPM 数据库；

第 12 行：找到 0 个源代码包，1 个执行代码包；

第 13-18 行：检查软件包的依赖关系。其依赖（requires）的东西均得到满足（satisfied）。

第 19 行：准备安装执行代码包；

第 20 行：取得已安装文件系统的列表；

第 21-26 行：再度取包裹文件头信息，算出软件包占用空间；

第 27-31 行：列出包中文件及要执行的操作，第 27 行的 test=0 表示不是进行安装测试，

file 包中的文件均要执行创建操作（create）。

第 32 行：如果有安装前执行脚本程序的话，执行它；

第 33 行：安装软件包 file-3.26-6；

第 34 行：如果有安装后执行脚本程序的话，执行它。

3.root：指定根目录

指定根目录 ROOT，其作用在于将系统所有操作限定在指定的目录下面，这样 RPM 操作的数据库位置变了，软件包的安装位置也变了。它是通过 chroot 系 统调用实现的。RPM 默认的数据库目录是/var/lib/rpm，如果指定根目录为/usr，则 RPM 的数据库目录将变为 /usr/var/lib/rpm。同样，如果包中有个文件为 /etc/zhsoft.txt，则安装后为/usr/etc/zhsoft.txt。

看下面的例子：

```
# rpm -i --root /usr file-3.26-6.i386.rpm
failed to open /usr/var/lib/rpm/packages.rpm

error: 不能打开 /usr/var/lib/rpm/packages.rpm
#
```

注：指定 RPM 根目录为/usr 安装 file 出现错误，RPM 打不开 /usr/var/lib/rpm/packages.rpm 文件，这是因为实际上在 /usr/var/lib/rpm 目录下没有任何 RPM 数据库。如果想成功安装，可用初始化 RPM 数据库命令来建立 RPM 数据库，命令是 rpm --initdb --root /usr（这些功能将在以后讲到）。

4.rcfile：指定 RPM 资源配置文件

RPM 的资源配置文件里，存放着 RPM 的默认设置，有编译选项 optflags，体系定义 arch_canon，兼容定义 arch_compat，宏文件定义 macrofiles 等。RPM 默认的资源配置文件（按读取顺序）为：/usr/lib/rpm/rpmrc，/etc/rpmrc，~/.rpmrc。（后两个文件现在意义不大了，如今一般换作 RPM 宏文件了。较之资源配置文件，宏文件有更大的灵活性。）

如果想让 RPM 使用用户指定的资源配置文件，就用--rcfile 来设定，可同时设定多个文件，文件间以冒号（:）分隔，如 /usr/lib/rpm/rpmrc:/etc/rpmrc:~/.rpmrc 这种形式。

5.dbpath：指定 RPM 数据库目录

本选项不同于--root 选项，它仅仅是指定 RPM 数据库的目录，并不改变安装文件的位置。

精通 RPM 之卸载篇

安全地卸载

RPM 卸载软件包,并不是简单地将原来安装的文件逐个删除,那样做的话,可能会出现这样或那样的问题。如,A 软件包依靠 B 软件包做某些工作,若 B 软件包卸载了,则 A 软件包就不能正常运行了。RPM 为用户安全考虑,会做某些检查,尽可能排除出错的情况。

RPM 在卸载软件包时,分步骤进行如下操作:
1. 检查依赖(Denpendency)
检查依赖的目的,在于确保当前没有任何软件包依赖于当前要卸载的软件包。
2. 执行卸载前触发脚本程序(Triggerun)

3. 执行卸载前脚本程序(Preuninstall)
本程序为软件包制作者设定，完成卸载前的一些工作，具体如,要卸载的软件包中某个
程序当前正在运行时,脚本程序必须杀掉它,否则无法正确卸载。
4. 检查配置文件(Configfiles)
RPM 卸载软件包前会检查包中的配置文件是否修改了,如果修改了,RPM 要换个名字保存
该文件,文件名一般是"原文件名+.rpmsave"。(这是 RPM 软件包管理的特色之一)
5. 真正卸载
即按照软件包中的文件列表,将文件逐个删除。要注意的是,如果其中有文件属于别的
软件包(即两个软件包拥有一个名字相同的文件),则 RPM 并不删除。
6. 执行卸载后脚本程序(Postuninstall)
本程序主要是完成卸载的善后工作,如将系统配置文件 inetd.conf 修改一下,去掉安装
时添加的东西等等。
7. 更新 RPM 数据库
卸载后,RPM 删除该软件包在数据库中的所有信息。
8. 执行卸载后触发脚本程序(Triggerpostun)

命令格式

卸载 RPM 格式的软件包,可使用如下命令格式:

```
rpm -e [卸载选项 1 卸载选项 2...] [软件包标识 1 软件包标识 2...]
```

注: 也可使用--erase 代替-e,效果相同。

软件包标识

对于已安装的软件包,RPM 使用如下格式来唯一标识它:
Name[-Subname]-Version-Release
其中: Name: 指软件名;
Subname : 本项可选,指软件包子包的名字;
Version : 指软件版本号。注意:其中不能含减号(-)字符;
Release : 指软件释出号。
例如: 包裹文件 file-3.26-6.i386.rpm 安装后,其软件包标识为 file-3.26-6。可以用此标识卸载,查询,校验该软件包。在命令行上输 入软件包标识时,可以写全,也可以采用简写的方式。如 file-3.26-6,可仅输入 file,或 file-3.26,或输全。又如: 带有子包的软件包标识 XFree86-devel-3.3.6-6,可仅输入 XFree86-devel,或 XFree86-devel-3.3.6,还 可完全输入。
注意: 软件包标识是大小写敏感的,即大写字母与小写字母表示不同的涵义。如 bash-2.03-10 标识不能输作 Bash-2.03-10,也不能输作 bAsh-2.03-10 等等。

选项列表

选项详解

通用选项的解释,请参见精通 RPM 之安装篇,本文不再赘述。
下面对指定用选项做些解释:

1. --test : 卸载测试
RPM 的卸载测试做些什么?主要是检查依赖关系,确保系统中没有软件包依赖于要卸载的软件包。如果还有软件包依赖要卸载的软件包,则 RPM 会提示依赖关系将中断的错误,例如:

```
# rpm -e --test mktemp
error: removing these packages would break dependencies:
mktemp is needed by autoconf-2.13-5
mktemp is needed by dev-2.7.7-4BP
mktemp is needed by groff-1.11a-10BP
mktemp is needed by gzip-1.2.4-15BP
mktemp is needed by initscripts-4.16-3BP
mktemp is needed by metamail-2.7-21BP
mktemp is needed by rpm-3.0.3-4BP
mktemp is needed by autofs-3.1.3-2
/bin/mktemp is needed by apache-1.3.12-5BP
/bin/mktemp is needed by linuxconf-1.16r1.3-4BP
#
```

当然,我们通过浏览卸载测试时输出的调试信息就知道得更清楚了。

```
# rpm -e --test -vv file | nl
1 D: opening database mode 0x0 in //var/lib/rpm/
2 D: getting list of mounted filesystems
3 + echo triggeruninstall
4 triggeruninstall
5 D: will remove files test = 1
6 D:file: /usr/share/magic action: remove
7 D:file: /usr/man/man4/magic.4 action: remove
8 D:file: /usr/man/man1/file.1 action: remove
9 D:file: /usr/bin/file action: remove
10 + echo triggerpostuninstall
11 triggerpostuninstall
12 D: removing database entry
```

注: 本例进行 file 包的卸载测试,输出信息送管道交 nl 列出行号,现逐行解释。(行号后以 D:开始的行输出的为调试信息)
第 1 行: 打开/var/lib/rpm 目录下的 RPM 数据库;
第 2 行: 取已安装文件系统列表;
第 3 行: 执行卸载前触发脚本程序;
第 4 行: 这一行是卸载前触发脚本程序的执行结果;
第 5 行: RPM 提示将要删除文件,其中 test=1 表明为卸载测试,并非真正删除;
第 6-9 行: 显示各文件的执行操作--删除;
第 10 行: 执行卸载后触发脚本程序;
第 11 行: 本行是卸载后触发脚本程序的执行结果;
第 12 行: 删除软件包在数据库中的信息。
2. --nodeps : 不检查依赖
RPM 管理软件包,不仅管理包中的所有文件,还同时管理着软件包之间的依赖关系。如 A 依赖于 B 运行,若 B 不存在了,则 A 也就运行不了了。RPM 维护着这种关系, 尽量避免破坏,以保证软件的正常运行。上例进行 mktemp 包的卸载测试时,RPM 提示了中断依赖关系的错误。如果使用本选项,不检查依赖,则卸载测试会 顺利完成。

```
# rpm -e --test --nodeps mktemp
#
```

需要说明的是,除非用户对软件包有足够了解,否则不要使用本选项,最好让 RPM 自动维护软件间的依赖关系,确保其能正常运行。
3. --noscripts : 不执行脚本程序
一个 RPM 软件包中可包含五种脚本程序,即:安装前脚本程序,安装后脚本程序,卸载前脚本程序,卸载后脚本程序和校验脚本程序。卸载时使用--noscripts 选项,可禁止卸载前与卸载后脚本程序的执行。
注意: 一般用户不要使用此选项。此选项主要提供给软件包制作者使用的。通过禁止执行脚本程序,可以防止因安装带有 BUG 的软件包而宕掉整个系统。当软件包去掉了 BUG 后,此选项就不必使用了。
4. --notriggers : 不执行触发程序
为了软件包间的交互控制,RPM 设计了三种触发程序:安装时触发程序,卸载前触发程序和卸载后触发程序。为了防止因某一软件包的卸载而引发卸载前与卸载后触发程序的执行,可以使用--notriggers 选项。通过比较下一例子与第 1 个选项介绍中的例子的输出,就可看到:选择--notriggers 选项 后,卸载前与卸载后触发程序都没有被触发执行。

```
# rpm -e --test -vv --notriggers file | nl
1 D: opening database mode 0x0 in //var/lib/rpm/
2 D: getting list of mounted filesystems
3 D: will remove files test = 1
4 D:file: /usr/share/magic action: remove
```

```
5 D:file: /usr/man/man4/magic.4 action: remove
6 D:file: /usr/man/man1/file.1 action: remove
7 D:file: /usr/bin/file action: remove
8 D: removing database entry
```

5. --allmatches : 卸载所有匹配的包

RPM 可将同一软件的不同版本安装到系统中,这样如果想全部卸载掉,这时仅简单输软件名是不行的,将出现错误,如:

```
# rpm -q foo
foo-6.0-1
foo-7.0-1
# rpm -e foo
error: "foo" specifies multiple packages
#
```

注: 本例中先查询 foo 包,发现有 6.0 和 7.0 版两个版本,接着卸载 foo 包,出现错误: 有多个软件包的名字均为 foo。
怎样才能全部卸载呢?
答案之一是一个一个卸载,卸载时输入软件包标识,因为软件名相同,所以最少再输个版本号,这种情况适用于软件版本少的时候。
答案之二是使用--allmatches 选项,仅输个软件名即可,指示 RPM 卸载所有匹配的包,比较方便。

```
# rpm -e --allmatches foo
# rpm -q foo
package foo is not installed
#
```

卸载过程没有出现错误,之后再查询 foo 包,RPM 则提示软件包未安装。

6. --justdb : 仅修改数据库

使用本选项后,RPM 将只更新其数据库,文件系统不更新,意即如果软件包安装有文件的话,卸载后这些文件照样存在。

```
# rpm -ql file
/usr/bin/file
/usr/man/man1/file.1
/usr/man/man4/magic.4
/usr/share/magic
# ls -l /usr/bin/file
-rwxr-xr-x 1 root root23948 Mar 23 1999 /usr/bin/file
```

```
# rpm -e --justdb file
# ls -l /usr/bin/file
-rwxr-xr-x 1 root root23948 Mar 23 1999 /usr/bin/file
#
```

注: 本例先用 rpm -ql 查询得到 file 包的文件列表,再用 ls 命令列包中的一个文件/usr/bin/file,能列出信息,故文件存在。然后用 rpm -e --justdb 卸载 file 包但只更新数据库,而后再用 ls 命令列/usr/bin/file,可以看到该文件依然存在。

精通 RPM 之升级篇

升级做什么

RPM 的升级功能是它受到用户好评的原因之一。因为用户自己将一个软件包从旧版本升级到新版本,特别是大型软件,需要有经验的支持和技术的积累,比较复杂,而用 RPM 升级软件,只需一个 rpm -U 命令就可以了,极大方便了用户。

软件升级基本做两项工作,一是安装新版本,二是卸载旧版本。RPM 还有一项重要的工作要做,这就是妥善处理配置文件(CONFIG FILE)。若直接采用安装方式,则用户已配置好的配置文件就会被覆盖,不符合用户要求。

配置文件处理

RPM 对某个配置文件,通过比较三种不同的 MD5 检查和(checksum)来决定如何处理它。这三种不同的 MD5 检查和是:

1. 原检查和。它是旧版本软件包安装时配置文件的 MD5 检查和。
2. 当前检查和。它是升级时旧版本配置文件的 MD5 检查和。
3. 新检查和。它是新版本软件包中配置文件的 MD5 检查和。

RPM 针对以下几种情况分别处理:

1. 当原检查和=X,当前检查和=X,新检查和=X 时:

这表明配置文件未曾修改过。此时,RPM 会将新的配置文件覆盖掉原文件,而不是对原文件不作处理,原因在于: 虽然文件名和文件内容都没有变化,但文件别的方面的属性(如文件的属主,属组,权限等)却可能改变,所以有必要覆盖一下。

2. 当原检查和=X,当前检查和=X,新检查和=Y 时:

这表明原配置文件没有改动过,但是它与新软件包中的配置文件却有所不同。这种情况下,RPM 将用新文件覆盖掉旧文件,并且旧文件不作保存(因为它不曾改动过,没有必要保存)。

3. 当原检查和=X,当前检查和=Y,新检查和=X 时:

这表明新文件与旧文件内容相同,但当前文件已经作过修改,这些修改对于新版本来说应该是合法的,可以使用的。因此,RPM 对当前文件予以保留。

4. 当原检查和=X,当前检查和=Y,新检查和=Y 时:
这表明原文件经过修改,现在已与新文件相同,这或许是用户用来修补安全上的漏洞,新版本也作了同样的修改。这种情况下,RPM 将新文件覆盖当前文件,避免文件属性方面的不同。
5. 当原检查和=X,当前检查和=Y,新检查和=Z 时:
这表明用户已修改了原文件,并且当前内容与新文件内容不同。这种情况下,RPM 无法保证新版本软件能正常使用当前的配置文件,所以采用了一个比较明智的办法,既能保护用户的配置数据,又能保证新版本软件正常。这种作法就是将当前文件换名保存(给原文件名加个.rpmsave 的后缀,如原文件名为 ABC,则 换名后为 ABC.rpmsave),同时安装新文件,并给出警告信息,如:

```
warning: /etc/.funkey saved as /etc/.funkey.rpmsave
```

6. 当没有原检查和时:
此种情况下,当前检查和与新检查和已无关紧要,这表明没有安装过此配置文件。因为没有安装过此配置文件,所以 RPM 无法判断当前文件是否被用户修改过。这种情况下,RPM 会将当前文件换名保存(原文件名后缀不是加个.rpmsave,而是.rpmorig),同时安装新文件,并给出警告信息,如:

```
warning: /etc/.inputdef saved as /etc/.inputdef.rpmsave
```

升级命令格式

升级 RPM 包时,请用以下命令格式:
rpm -U [升级选项 1 升级选项 2...] [软件包标识 1 软件包标识 2...]
其中: 也可使用--upgrade 代替-U,效果相同。
软件包标识
有关软件包标识的定义,请参见精通 RPM 之卸载篇。

选项列表

选项说明

因为升级也是一种安装,所以升级的选项列表与安装选项列表基本相同,只是升级的选项列表增加了一项--oldpackage。现着重说明一下这个选项,其它选项说明见精通 RPM 之安装篇,在此恕不赘述。

--oldpackage 选项: 从名字上就可以看出来是老版本软件包的意思。为什么要将软件"升级"到老版本?(这里的升级其实是降级)这里面有个原因。用户一直好好地用着老版本的软 件,当有一天发现有新版本发布时,马上用 rpm -U 命令升级到系统中,但因为新版本有"臭虫",所以这个软件暂时不能正常工作。而这时,直接用 rpm -U 命令是升级不到老版本的,因为一般情况的升级是将老版本升级到新版本,RPM 默认这一点。若想升级到老版本,则必须用这个特殊的选项。下面举个例子:

```
# rpm -U -v lze-6.0-1.i386.rpm
package lze-7.0-1 (which is newer then lze-6.0-1) is already installed
#
```

注: 本例在升级过程中出现错误,RPM 提示 lze 软件包已安装,并且现存版本号 7.0,高于准备升级的版本号 6.0,升级无法继续。
若在命令行使用--oldpackage,结果会怎么样呢?

```
# rpm -U -v --oldpackage lze-6.0-1.i386.rpm
lze-6.0-1
#
```

注: 命令执行后输出了软件包标识 lze-6.0-1,表明升级到老版本成功了。
下面通过输出调试信息来观察一下升级软件包时 RPM 做的主要工作:

```
# rpm -U -vv --oldpackage foo-3.0-2.i386.rpm 2>&1 | nl
1 D: counting packages to install
2 D: found 1 packages
3 D: looking for packages to download
4 D: retrieved 0 packages
5 D: New Header signature
6 D: Signature size: 68
7 D: Signature pad : 4
8 D: sigsize : 72
9 D: Header + Archive: 1577
10 D: expected size : 1577
11 D: opening database mode 0x42 in //var/lib/rpm/
12 D: found 0 source and 1 binary packages
13 D: requires: /bin/sh satisfied by db file lists.
14 D: installing binary packages
15 D: getting list of mounted filesystems
16 D: New Header signature
17 D: Signature size: 68
18 D: Signature pad : 4
19 D: sigsize : 72
20 D: Header + Archive: 1577
21 D: expected size : 1577
22 D: package: foo-3.0-2 files test = 0
23 D:file: /etc/foo.conf action: create
24 D:file: /usr/bin/foo action: create
25 D: running preinstall script (if any)
26 + echo preinstall
27 preinstall
28 foo-3.0-2
29 D: running postinstall scripts (if any)
30 + echo postinstall
31 postinstall
32 + echo triggerinstall
33 triggerinstall
34 + echo triggeruninstall
35 triggeruninstall
36 + echo preuninstall
```

```
37 preuninstall
38 D: will remove files test = 0
39 D:file: /usr/bin/foo action: skip
40 D:file: /etc/foo.conf action: skip
41 D: running postuninstall script (if any)
42 + echo postuninstall
43 postuninstall
44 D: removing database entry
45 D: removing name index
46 D: removing group index
47 D: removing requiredby index for /bin/sh
48 D: removing trigger index for file
49 D: removing trigger index for file
50 D: removing trigger index for file
51 D: removing file index for foo.conf
52 D: removing file index for foo
```

注：第 1-4 行：计算命令行上要升级的包数,并且下载那些需要下载的包裹文件;
第 5-10 行：根据包裹文件头部信息,确定软件占用空间;
第 11,12 行：打开 RPM 数据库及包裹文件;
第 13 行：检查依赖是否满足,本例满足;
第 14 行：安装执行程序包;
第 15 行：取当前已安装文件系统列表;
第 16-21 行：再度检查包裹头信息,确定占用系统空间;
第 22-24 行：确定包中各个文件的执行操作(action),均为建立(create);
第 25 行：执行安装前脚本程序(如果有的话);
第 26-27 行：以+开头的为脚本程序执行的命令,其后为其输出结果;
第 28 行：安装 foo-3.0-2 包;
第 29 行：执行安装后脚本程序(如果有的话);
第 30-31 行：以+开头的为脚本程序执行的命令,其它为执行结果;
第 32-33 行：执行安装时触发脚本程序;
第 34-35 行：执行卸载前触发脚本程序,自此开始卸载原软件包;
第 36-37 行：执行卸载前脚本程序;
第 38-40 行：确定原包中各文件的执行操作,本例均为跳过(skip),即不作处理;
第 41-43 行：执行卸载后脚本程序;
第 44-52 行：删除原包在 RPM 数据库中的所有信息(数据及索引)。

精通 RPM 之查询篇

RPM 不仅在安装,升级,卸载方面工作出色,而且在查询方面比其它软件包管理工具更胜一筹。这从以下几种情况可以看出：
* 当你在浏览系统文件时,发现一个文件,想知道它来自哪个软件包时,可以用 RPM 来查询得知;

* 当你的朋友给你发送来一个软件包,但你不知道这是个什么样的软件包,不知道它做些什么,安装些什么,来源是哪里。这时,你可以用 RPM 查询搞定;
* 几个月前你安装了 XFree86 窗口软件,但现在你忘了它的版本号,也不知它的说明文档在哪里。这时,你可以用 RPM 查询一下这个软件包,得到这方面的信息。

RPM 的查询还有一个高级功能,即定制输出功能。你可用--queryformat(或-qf)来定制一下输出格式,这样,RPM 查询得到的信息将以你定制的格式输出,这样很是方便,尤适合于程序的自动处理。

命令格式

查询 RPM 格式的软件包，可使用如下命令格式：

```
rpm -q [查询选项 1 查询选项 2...]
```

注：也可使用--query 代替-q，效果相同。

选项列表

选项详解

指定用选项中 ftp 与 http 相关的四个选项(--ftpproxy,--ftpport,--httpproxy,--httpport)和通用选项的解释,请参见精通 RPM 之安装篇,本文不再赘述。下面对指定用选项做些解释:

指定用选项可分为如下几类:

一、软件包选择类

此类选项在一次只能选择一个,选择多个时 RPM 将提示错误:
rpm: one type of query/verify may be performed at a time

从查询方面看,一个是查询那些已安装的软件包,一个是查询未安装的软件包。

1. 查询已安装的软件包,使用下列选项:

(1) -a (或--all) : 查询所有已安装的软件包

```
# rpm -q -a
setup-2.0.2-1
filesystem-1.3.4-5
basesystem-6.0-5
```

```
agrep-2.04-5
aktion-0.3.6-2
amor-0.5-1
dhcpcd-1.3.17pl2-1
ldconfig-1.9.5-15
glibc-2.1.2-12
chkconfig-1.0.6-2
......
#
```

注: 本例查找当前系统中安装的所有软件包,输出很多,仅列出几个,剩下的省略掉了(以.....表示)。

(2) -g (或--group) : 查询有哪些软件包属于指定类别

RPM 根据软件功用的不同,将软件分为以下若干类:(括号内为注释)

```
Amusements/Games (娱乐/游戏)
Amusements/Graphics(娱乐/图形)
Applications/Archiving (应用/档案)
Applications/Communications (应用/通讯)
Applications/Databases (应用/数据库)
Applications/Editors (应用/编辑器)
Applications/Emulators (应用/仿真器)
Applications/Engineering (应用/工程)
Applications/File (应用/文件)
Applications/Internet (应用/因特网)
Applications/Multimedia (应用/多媒体)
Applications/Productivity (应用/产品)
Applications/Publishing (应用/印刷)
Applications/System (应用/系统)
Applications/Text (应用/文本)
Development/Debuggers (开发/调试器)
Development/Languages (开发/语言)
Development/Libraries (开发/函数库)
Development/System (开发/系统)
Development/Tools (开发/工具)
Documentation (说明文档)
System Environment/Base (系统环境/基础)
System Environment/Daemons (系统环境/守护)
System Environment/Kernel (系统环境/内核)
System Environment/Libraries (系统环境/函数库)
System Environment/Shells (系统环境/接口)
User Interface/Desktops (用户界面/桌面)
```

```
User Interface/X (用户界面/X 窗口)
User Interface/X Hardware Support (用户界面/X 硬件支持)
Other (其它)
```

注意: 类别是大小写敏感的,这一点输入时要小心。如果用户想要查询当前系统安装了哪些游戏类的软件包,可这样做:

```
# rpm -q -g Amusements/Games
clanbomber-1.01-1
kdegames-1.1.2-1
xbill-2.0-6
xboard-4.0.0-3
xboing-2.4-7
xgammon-0.98-14
xjewel-1.6-11
xpat2-1.04-10
xpilot-3.6.2-6
xpuzzles-5.4.1-7
xtrojka-1.2.3-6
#
```

如果用户想查询若干类别的软件包时,可以把类别都列出来,一起查询,如:

```
# rpm -q -g Applications/Editors Applications/File
emacs-20.3-15
emacs-nox-20.3-15
emacs-X11-20.3-15
vim-common-5.6-12
vim-minimal-5.6-12
vim-X11-5.6-12
fileutils-4.0-3
bzip2-0.9.5d-1
findutils-4.1-32
git-4.3.17-6
gzip-1.2.4-15
slocate-1.4-7
stat-1.5-12
file-3.26-6
#
```

注: 本例查询结果中前 6 个为"应用/编辑器"类别,后 8 个为"应用/文件"类别。

(3) -f (或--file) : 查询有哪些软件包拥有指定文件这个选项非常有用。当用户不清楚某个文件属于哪个软件包时,可试试这个选项。假如我记不清/usr/sbin/ftpd 这个文件在哪个包中,现在用 RPM 查询:

```
# rpm -qf /usr/sbin/ftpd
inet6-apps-0.36-3
#
```

现在知道了,它在 inet6-apps-0.36-3 包中。命令中-qf 是-q -f 的缩写,当选项仅带一个减号时,多个选项可以合并在一起,如-a -b -c 三个选项可写作-abc,执行效果相同。

(4) --whatrequires : 查询有哪些包需要指定功能

本选项指示 RPM 从所有已安装的软件包中,查询有哪些软件包提供了用户指定的功能,命令行上可输入一个或多个功能。

```
# rpm -q --whatrequires /bin/ps libc.so.6 | head
autofs-3.1.3-2
agrep-2.04-5
aktion-0.3.6-2
amor-0.5-1
chkconfig-1.0.6-2
libtermcap-2.0.8-16
bash-2.03-10
ncurses-4.2-18
info-3.12f-4
fileutils-4.0-3
#
```

注: 本例从系统中查询哪些软件包需要/bin/ps 和 libc.so.6 功能,通过管道输出前 10 行内容。

(5) --whatprovides : 查询有哪些包提供指定功能

本选项指示 RPM 从所有已安装的软件包中,查询有哪些软件包提供了用户指定的功能。命令行上可输入一个或多个功能。

```
# rpm -q --whatprovides libc.so.6 /bin/ps
glibc-2.1.2-12
no package provides /bin/ps
#
```

注: 本例从系统中查询有哪些软件包提供 libc.so.6 与/bin/ps 功能,结果是 glibc-2.1.2-12 包提供了 libc.so.6,而没有包提供/bin/ps 功能。

(6) --triggeredby : 查询有哪些包被指定的包触发本选项指示 RPM 从所有已安装的软件包中,查询有哪些包可被用户指定的包触发。指定的包可输入一个或多个。

```
# rpm -q --triggeredby file
foo-8.0-1
#
```

注: 本例查询后得知,安装或卸载 foo-8.0-1 包,将会触发与 file 包相关的脚本程序执行。

2. 查询未安装的软件包: (仅有一个选项可用)

(1) -p : 查询指定包裹文件的信息

使用本选项,通过指定一个或多个包裹文件名,RPM 就可以获取相应包裹的有关信息。包裹文件可以是一般形式,也可是 ftp/http 形式。

```
# rpm -qp -l --ftpport 2121
ftp://zzz:pass@linux/zhsoft/file-3.26-6.i386.rpm
/usr/bin/file
/usr/man/man1/file.1
/usr/man/man4/magic.4
/usr/share/magic
#
```

注: 本例查询 ftp 形式的包裹,显示其文件列表(-l 选项使然,下面要讲到)。ftp 用户名 zzz,密码为 pass,远程机为 linux,文件名为 /zhsoft/file-3.26-6.i386.rpm,ftp 使用 2121 端口。

二、信息显示类

本类选项可同时选择多个,以显示多种信息。

1. -i : 显示软件包信息

当用户需要了解软件包的头部信息时,要使用该选项。

```
# rpm -qi file
Name: file Relocations: /usr
Version : 3.26 Vendor: Red Hat Software
Release : 6 Build Date: 1999 年 03 月 23 日 星期二 05 时 32 分 29 秒
Install date: 2001 年 10 月 14 日 星期日 21 时 18 分 25 秒 Build Host:
porky.devel.redhat.com
Group : Applications/File Source RPM: file-3.26-6.src.rpm
```

```
Size: 211946 License: distributable
Packager: Red Hat Software <http://developer.redhat.com/bugzilla/>
Summary : A utility for determining file types.
Description :
The file command is used to identify a particular file according to the
type of data contained by the file. File can identify many different
file types, including ELF binaries, system libraries, RPM packages, and
different graphics formats.

You should install the file package, since the file command is such a
useful utility.
#
```

上述输出中,包含多个域,各个域的含义为:

```
Name : 软件名;Relocations : 重定位目录前缀(一个或多个);
Version : 版本号;Vendor : 开发商;
Release : 释出号;Build Date : 包建立时间;
Install date : 安装时间; Build host : 包建立主机;
Group: 类别; Source RPM : 源代码包;
Size : 大小; License : 许可证;
Packager : 打包者;
Summary : 软件简介;
Description : 详细描述。
```

2. -l (或--list) : 显示软件包文件列表

当用户想知道软件包包含哪些文件时,要使用该选项。输出时,一行一个文件名,形成文
件列表。下例列出 file 包中文件:

```
# rpm -qlv file
-rwxr-xr-x root root23948 3 月 23 1999 /usr/bin/file
-rwxr-xr-x root root12023 3 月 23 1999 /usr/man/man1/file.1
-rwxr-xr-x root root 6625 3 月 23 1999 /usr/man/man4/magic.4
-rwxr-xr-x root root 169350 3 月 23 1999 /usr/share/magic
#
```

注: 本例列文件时选用了通用选项-v,这样列出的格式同 ls 命令列出的格式,从中可以看到各个文件的权限,属主,属组,大小,最后修改时间及文件名等信息。

3. -d (或--docfiles) : 显示软件包说明文档

RPM 将软件包中的文件分为三类,一是配置文件,一是说明文档,再一是其它文件(包括可执行程序及数据文件等)。通过-d 选项,可列出包中所有说明文档。下例列出 file 包中说明文档:

```
# rpm -qd file
/usr/man/man1/file.1
/usr/man/man4/magic.4
#
```

4. -c (或--configfiles) : 显示软件包配置文件

使用本选项可列出包中所有配置文件。下例列出 lze 包中的配置文件:

```
# rpm -qc lze
/etc/funkey.def
/etc/inputme.def
#
```

5. -s (或--state) : 显示软件包文件状态

RPM 已安装的软件包中各个文件可拥有如下 4 种状态:
* normal(正常): 这表明文件未被其它软件包修改过。
* replaced(已替换): 这表明文件已被其它软件包修改替换过了,不再是原先的文件了。* not installed(未安装): 这表明文件未安装。有一种原因可导致这种情况发生,这就是当安装软件包时使用了--excludedocs 选项(或是通过设置%_excludedocs 宏为 1),说明文档未被安装到系统中。当用 RPM 查询此软件包状态时,所有说明文档均显示此状态。
* net shared(网络共享): 这表明文件处于网络共享状态。这一状态是 RPM 用来支持 NFS(网络文件系统)的,用于避免一个 NFS 客户端系统删除一个正在共享的文件时,另一 NFS 客 户端系统无法正常使用含此文件的软件包。有两种情况可使文件在此状态,一是文件安装在真实网络共享的目录里,二是文件安装在 RPM 资源配置文件 中%_netsharedpath 宏所确定的目录里。

```
# rpm -i --excludedocs file-3.26-6.i386.rpm
# rpm -qs file
normal/usr/bin/file
not installed /usr/man/man1/file.1
not installed /usr/man/man4/magic.4
normal/usr/share/magic
#
```

注: 本例先安装 file 包裹文件,选用--excludedocs 表明不安装说明文档。而后再查询 file 包文件的状态,可以看到其说明文档均为 not installed(未安装),

其它文件为 normal(正常)状态。

6. -R (或--requires) : 显示软件包所需的功能

所谓功能,可以是软件包标识,可以是文件(如共享库 libc.so.6 等),也可以是虚拟的名字。软件包的依赖关系,就是依靠功能来实现的。RPM 安装软件包时,如果所需功能不存在,则其依赖关系不满足,RPM 将中断安装过程。

```
# rpm -q -R lze
/bin/sh
ld-linux.so.2
libc.so.6
libc.so.6(GLIBC_2.0)
libc.so.6(GLIBC_2.1)
#
```

注: 本例查询 lze 包所需的功能。

7. --provides : 显示软件包提供的功能

一个软件包,可以提供若干功能,这些功能可以是动态链接库等实际的文件,也可以是虚拟的名字(只要其它软件包可以用就行了)。如果想查询某个包提供了哪些功能,要用--provides
选项。

下面的例子查询一个包裹文件所提供的功能:

```
# rpm -qp --provides zlib-1.1.3-6.i386.rpm
libz.so.1
libz.so.1(GCC.INTERNAL)
#
```

注: 本包裹文件提供的功能是 libz.so.1 动态链接库。

8. --conflicts : 显示软件包冲突的功能

何谓冲突?冲突就是不同软件包之间不能共存的现象。RPM 制作软件包时,可以记录下与本软件包不能共存的功能,这样安装该包时,若冲突的功能已然存在,则 RPM 将中止安装。
下面的例子查询 at-3.1.7-8 包是否有与其冲突的功能:

```
# rpm -q --conflicts at-3.1.7-8
crontabs <= 1.5
#
```

注: 本例显示 at 包与版本号小于 1.5 的 crontabs 包有冲突存在。

9. --scripts : 显示软件包内置的脚本程序

scripts 即脚本,指的是用 SHELL 语言编写的程序。选用本选项时,RPM 将输出软件包所含的各类脚本程序的内容。总共有 5 类这样的脚本程 序,即安装前脚本程序(preinstall),安装后脚本程序(postinstall),卸载前脚本程序(preuninstall),卸载后脚本程 序 (postuninstall)和校验脚本程序(verify)。下面的例子列出 zsh 包的脚本程序:

```
# rpm -q --scripts zsh-3.0.7-4 | nl
1 postinstall script (through /bin/sh):
2 if [ ! -f /etc/shells ] ; then
3 echo "/bin/zsh" > /etc/shells
4 else
5 echo "/bin/zsh" >> /etc/shells
6 fi
7 /sbin/install-info /usr/info/zsh.info.gz /usr/info/dir \
8 --entry="* zsh: (zsh). An enhanced bourne shell."
9 preuninstall script (through /bin/sh):
10 if [ "$1" = 0 ] ; then
11 /sbin/install-info --delete /usr/info/zsh.info.gz /usr/info/dir \
12 --entry="* zsh: (zsh). An enhanced bourne shell."
13 fi
14 postuninstall script (through /bin/sh):
15 if [ "$1" = 0 ] ; then
16 if [ -f /etc/shells ] ; then
17 TmpFile=`/bin/mktemp /tmp/.zshrpmXXXXXX`
18 grep -v '^/bin/zsh$' /etc/shells > $TmpFile
19 cp -f $TmpFile /etc/shells
20 rm -f $TmpFile
21 chmod 644 /etc/shells
22 fi
23 fi
#
```

注: 本例利用管道技术给查询的每行输出加上了行号,便于观察解释。
第 2-8 行: 为安装后脚本程序(postinstall)的源代码;
第 10-13 行: 为卸载前脚本程序(preuninstall)的源代码;
第 15-23 行: 为卸载后脚本程序(postuninstall)的源代码。
例中所有脚本程序均通过/bin/sh 解释执行,本例没有安装前脚本程序(preinstall)和
校验脚本程序(verify)。

10. --triggers ： 显示软件包内置的触发脚本程序
触发脚本程序是 scripts 类脚本程序的扩展,它用于软件包之间的交互控制。触发脚本
程序有安装时触发脚本程序(triggerin),卸载前触发脚本程序(triggerun)和卸载后触发脚
本程序(triggerpostun)三种。
下面的例子列出 zsh 包中的触发脚本程序:

```
# rpm -q --triggers zsh-3.0.7-4
triggerpostun script (through /bin/sh) -- zsh <= 3.0.7-2
if [ ! -f /etc/shells ] ; then
echo "/bin/zsh" > /etc/shells
else
echo "/bin/zsh" >> /etc/shells
fi
#
```

注: 输出的第 1 行说明脚本程序的类别(为卸载后触发脚本程序 triggerpostun),用什么程序解释(一般为/bin/sh),和触发的条件(在--之 后描述)。本例的触发条件为 zsh 的版本小于或等于 3.0.7-2。如果条件成立,则此脚本程序将会执行。输出的第 2-6 行为卸载后触发脚本程序的源代码。

11. --changelog ： 显示软件维护记录
changelog 即软件维护记录,用它来记录什么人,什么时间,改动了软件的什么地方。通
过查看维护记录,开发者或用户可以了解软件的开发进展情况。
下面的例子查询 lze 包的维护情况:

```
# rpm -q --changelog lze-6.0-2
* 五 5 月 01 1998 雨亦奇 <zhsoft@371.net>
- 增加多窗口操作
* 一 3 月 24 1997 雨亦奇 <zhsoft@371.net>
- 增加块操作命令
#
```

注: 从本例中就可以看出来,维护记录的书写有一定的规范:以星号(*)开头的行记录维护时间,维护人及其电子邮箱,而以减号(-)开头的行则记录维护的具体内容。

12. --dump ： 显示软件包中所有文件的属性信息
如果用户想查看某个软件包内文件的属性信息,请用--dump 选项。

```
# rpm -q --dump file
/usr/bin/file 23948 922138347 abaf6cfd51588ac7c484526fbdb84e5b 0100755
```

```
root root 0 0 0 X
/usr/man/man1/file.1 12023 922138346 76d9ff6567ab64a53eab50911272f5c1
0100755 root root 0 1 0 X
/usr/man/man4/magic.4 6625 922138346 b8d126803709f0da7f39f5125a132cd3
0100755root root 0 1 0 X
/usr/share/magic 169350 922138346 3bd2eaf3c5e0e84153ba7df38b7561fc 0100755
root root 0 0 0 X
#
```

现在根据例子的输出,解释一下 RPM 包中各个文件所拥有的属性信息:(以第 1 行为例)
/usr/bin/file : 为文件名;
23948 : 指文件大小(字节数);
922138347 : 指文件最后修改时间(秒数);
abaf6cfd51588ac7c484526fbdb84e5b : 指根据文件内容计算出的 MD5 检查和;
0100755 : 指文件存取权限;
root : 指文件属主;
root : 指文件属组;
0 : 配置文件标志,为 0 表示该文件非配置文件,为 1 表明该文件是配置文件;
0 : 说明文档标志,为 0 表示该文件非说明文档,为 1 表明该文件是说明文档;
0 : 指文件类型,为 0 表示普通文件,非 0 表示设备文件,包含主从设备号;
X : 符号连接内容,如果文件不是符号连接,则此值为 X。

三、定制输出类

定制输出是 RPM 一项强有力的功能,当用户想要 RPM 按自己的格式输出软件包信息时,可以通过使用--qf(或--queryformat)选项来实现定制。

定制输出要指定格式化的字符串(类似 printf 的格式串),该字符串以单引号'或双引号"引住。格式串中可包括普通文本,含转义符的文本,功能标签和数组循环三种不同的组件。

* 普通文本
格式串中的普通文本将按原样输出。

* 含转义符的文本
RPM 定义的转义符为\,称作反斜杠。当 RPM 遇到此字符时,会根据预先定义的转义序列,把后面的字符解释后输出。

转义序列如下:

```
\a : 输出响铃字符(值为 7),效果是喇叭鸣叫一声。
\b : 输出退格符(值为 8),效果是删除光标前面一个字符,且光标后退一个字符位置。
```

```
\f : 输出换页符(值为 12),根据终端解释的不同,效果也不相同,有的是实现清屏操
作,有的则是换一行。
\n : 输出换行符(值为 10),效果是光标移到下一行。
\r : 输出回车符(值为 13),效果是光标回到行首。
\t : 输出跳格(值为 9),根据终端解释的不同,效果也不同,有的是输出一个空格,有
的 则输出最多 8 个空格。
\v : 输出垂直方向的跳格(值为 11),用处不大。
\\ : 输出反斜杠(\)这个字符。
```

* 功能标签
RPM 内置了很多功能标签,如 NAME 表示软件名,VERSION 表示版本号,RELEASE 表示释出号等。输出功能标签所表示的内容时,需用这样一种格式:

%[输出宽度]{功能标签}
注:输出宽度可选,正值表示右对齐输出,负值表示左对齐输出。
如格式串中使用%{NAME}时将输出软件名,使用%20{NAME}时则右对齐输出软件名,宽度为 20,而使用%-20{NAME}时则按宽度为 20 左对齐输出软件名。看看下面的实际例子就会明白:

```
# rpm -q --qf "%{NAME}:%20{NAME}:%-20{NAME}:\n" file
file:file:file:
#
```

功能标签书写不区分大小写,即 NAME 也可输作 name,Name 等等。功能标签还可以带有选项,选项用":选项名"表示,选项名有大小写之分。 如% {FILEMODES}默认以数字形式输出,如采用 perms 选项,即%{FILEMODES:perms},输出结果将采用 rwx 的形式。请看下面的例 子:

```
# rpm -q --qf "%{filenames} %{FILEMODES:perms}\n" file
/usr/bin/file -rwxr-xr-x-
#
```

注: %{filenames}表示包中的文件名。

RPM 常用功能标签表

注: 可用 rpm --querytags 命令查到 RPM 内置的所有功能标签。

* 数组循环
对于功能标签是数组的,可以用数组循环(用[与]括住的部分)来输出数组的全部内容。数组循环内部可包含功能标签,普通文本和含转义符的文本。RPM 将根 据循环内部一个或多个数组类型的功能标签所拥有的单元个数,取其最小值,循环同样次数,输出解释或转义后的数据。对于功能标签仅含一个单元的,若想输出多

次,则需要在功能标签前面加个等号(=)。

下面的例子以数组循环的方式,输出 file 包所含的各个文件的文件名及权限,同时输出每一个文件时都要先输出软件包标识。

```
# rpm -q --qf
"[%{=NAME}-%{=VERSION}-%{=RELEASE} : %-20{FILENAMES} : %{FILEMODES:perms}\n]"
file
file-3.26-6 : /usr/bin/file : -rwxr-xr-x-
file-3.26-6 : /usr/man/man1/file.1 : -rwxr-xr-x-
file-3.26-6 : /usr/man/man4/magic.4 : -rwxr-xr-x-
file-3.26-6 : /usr/share/magic : -rwxr-xr-x-
#
```

通过使用定制输出,我们可以很方便地查找出系统中占用空间最大的软件包。

```
# rpm -q -a --qf "%{NAME} %{SIZE}\n" | sort -nr +1 | head -1
kernel-source 65824967
#
```

注:-a 项选择所有已安装的软件包,--qf 项定制查询格式, 格式串中用功能标签 NAME 输出软件名,用 SIZE 输出软件大小,查询得到的数据通过管道送 sort 命令,对第 2 列降序排序后由 head 命令取第 1 行内容,这即是占用空间最大的软件包。

精通 RPM 之校验篇

当用 RPM 安装,升级或卸载软件包时,RPM 将所有的信息记录到数据库中。RPM 的校验,就在于检查这些数据的完整性。一旦发现某个软件包被破坏(有文件丢失等情况),RPM 就会报告错误。RPM 通过这样一种机制,保证系统的正常运行。

RPM 除了校验软件包的依赖关系,还校验每个文件,检查其属性是否正确。文件属性包括属主,属组,权限,MD5 检查和,大小,主设备号,从设备号,符号连 接及最后修改时间 9 项内容,其中每一项改变 RPM 都会发现。这 9 项属性 RPM 并非都校验,因为文件类型不同,其中某些属性会没有意义,RPM 因而也不会去 检查。这里有一个表格:

注: -表示不校验,校表示校验。

校验时若发现文件丢失,RPM 将输出"missing 文件名"。若有属性方面错误,RPM 将用如下格式输出:

```
SM5DLUGT c 文件名

其中: S : 表示文件大小;
M : 表示权限;
5 : 表示 MD5 检查和;
D : 表示主从设备号;
L : 表示符号连接;
U : 表示属主;
G : 表示属组;
T : 表示最后修改时间。
```

如果文件某属性正常,则会显示点(.)字符,否则的话将显示其代表字符。看下例:

```
# rpm -V bash
S.5....T /etc/bashrc
#
```

本例校验 bash 包(校验命令 rpm -V 下面将详细解释),结果发现/etc/bashrc 文件有问题,输出的属性中:出现 S 字符,这表明文件大小改变了;出现 5,这表明文件内容与原内容不 同了;最后的 T 字符,则表明文件的最后修改时间改变了。通过这些属性字符,我们可以清楚地知道一个文件变动了什么,这是 RPM 的聪明之处。

格式中的 c 字符仅当校验文件为配置文件时才输出。如 lze 包中有/etc/inputme.def 配置文件,如果我们人为改变其最后修改时间,RPM 会发现什么:

```
# touch /etc/inputme.def
# rpm -V lze
.......T c /etc/inputme.def
#
```

注: 本例中用 touch 命令改变文件最后修改时间,然后校验,结果 RPM 发现了这个情况(以 T 表示出来),我们还看到了 c 字符,说明/etc/inputme.def 是个配置文件。

命令格式

校验 RPM 格式的软件包,可使用如下命令格式:

```
rpm -V [校验选项 1 校验选项 2...] [软件包标识或包裹文件...]
```

注：-V 可用--verify 或-y 代替，效果相同。

选项列表

选项说明

通用选项的解释，请参见精通 RPM 之安装篇，本文不再赘述。下面对指定用选项做些解释：
选项分为以下几类：

一、包选择类
1. -a（或-all）：校验所有已安装的软件包
本选项指示 RPM 校验系统中所有已安装的软件包：

```
# rpm -V -a
S.5....T c /etc/profile
.M....G. /usr/man/man1
missing/etc/rc.d/rc3.d/S85httpd
#
```

注：本例校验后，RPM 发现：
(1) 文件/etc/profile 的大小(S)，内容(5)及最后修改时间(T)已改变；
(2) 目录/usr/man/man1 的权限(M)，属组(G)已改变；
(3) 文件/etc/rc.d/rc3.d/S85httpd 丢失(missing)。

2. -f（或--file）：校验含指定文件的软件包
如果用户仅知道某个文件，但忘了它所在的软件包，又想校验这个包时，可用此选项。

```
# rpm -V -f /bin/cp /bin/bash
SM5....T /bin/ls
S.5....T /etc/bashrc
#
```

注：本例校验含/bin/cp 和/bin/bash 文件的软件包，发现有两个文件(/bin/ls 与/etc/bashrc)改变了。

3. -g（或--group）：校验指定类别的软件包
本选项用于校验已安装的指定类别的软件包。关于软件的类别，请参见精通 RPM 之查询篇。
下例校验 Applications/Editors 与 Applications/Text 类别的软件包：

```
# rpm -V -g Applications/Editors Applications/Text
.......T c /etc/inputme.def
#
```

4. -p : 校验指定包裹文件
如果用户想校验某个 RPM 包所含文件在系统中的属性是否正常,则可用-p 项指定该包裹文件进行校验, 如下例;

```
# rpm -V -p fileutils-4.0-3.i386.rpm
SM5....T /bin/ls
#
```

注: 本例中 RPM 发现 fileutils 包裹中的/bin/ls 文件与当前系统中的/bin/ls 文件有所不同(S 大小,M 权限,5 内容,T 时间)。

二、特殊要求类
1. --noscripts : 不执行校验脚本程序
有些 RPM 软件包中内置有校验脚本程序(verifyscript),它执行一些特殊的检查。当用户不想执行这类脚本程序时,需用--noscripts 选项。

```
# rpm -V -v XFree86-libs
Looking for /usr/X11R6/lib in /etc/ld.so.conf... found
# rpm -V -v --noscripts XFree86-libs
#
```

注: 本例中第一个校验所输出的内容为 XFree86-libs 包内的校验脚本程序的输出,第二个命令中用了--noscripts 选项,这使得校验脚本程序没有执行,因而也没有什么输出了。

2. --nodeps : 不检查依赖
RPM 校验软件包时,默认情况是检查依赖,因为依赖是否正常关系到一个软件能否可靠运行。校验时选用--nodeps 选项,用于指示 RPM 不检查依赖。注 意,本选项主要针对软件包制作者,除非必要,不要使用此选项。我们可以通过校验时用 -vv 选项输出调试信息,从而可以看到是否检查依赖的区别:

```
# rpm -V -vv lze-6.02-1
D: opening database mode 0x0 in //var/lib/rpm/
D: record number 7253720
D:requires: /bin/sh satisfied by db file lists.
D:requires: ld-linux.so.2satisfied by db provides.
D:requires: libc.so.6satisfied by db provides.
D:requires: libc.so.6(GLIBC_2.0)satisfied by db provides.
D:requires: libc.so.6(GLIBC_2.1)satisfied by db provides.
.......T c /etc/inputme.def
```

```
# rpm -V -vv --nodeps lze-6.02-1
D: opening database mode 0x0 in //var/lib/rpm/
D: record number 7253720
.......T c /etc/inputme.def
#
```

注: 例子中,有 requires 的行为检查依赖,以 D:开头的行为调试输出信息。

3. --nofiles : 忽略丢失文件的错误
默认情况下,RPM 在校验时若发现文件丢失,会报出"missing 文件名"的错误。如果用户想忽略这方面的错误,请用--nofiles 选项。请比较下例输出:

```
# rpm -V file
missing/usr/man/man1/file.1
# rpm -V --nofiles file
#
```

注: 使用此选项后 RPM 没有再报出文件丢失的错误。

4. --nomd5 : 忽略 MD5 检查和的错误
本选项用于指示 RPM 忽略文件内容的改变,即忽略 MD5 检查和错误。此选项常用于校验日志类型的文件(此类文件内容并不重要)。
请看下面的例子:

```
# rpm -V bash
S.5....T /etc/bashrc
# rpm -V --nomd5 bash
S......T /etc/bashrc
#
```

例子中校验 bash 包,不用--nomd5 选项时,输出有 5(MD5 检查和),表示 /etc/bashrc 内容变化了,而采用--nomd5 选项时,相应输出则为点(.),表示文件内容正常了。

精通 RPM 之制作篇（上）

要想制作一个 RPM 格式的软件包,需要编写软件包描述文件。其标准命名格式为:软件名-版本号-释出号.spec,这个文件,详细描述了有关该软件 包的诸多信息,如软件名,版本,类别,说明摘要,创建时要执行什么指令,安装时要执行什么操作,以及软件包所要包含的文件等等。有了这个文件,RPM 就可 以制作出相应的包裹文件来。

下面以我制作小赵编辑器 LZE 的软件包(lze-6.0-2.i386.rpm)为例,详细说明一下软件包描述文件的书写。其描述文件为 lze-6.0-2.spec,该文件内容如下:(用 nl -ba 命令列出,每行开头的数字为所在行在文件中的行号)

```
1 # 文件名称: lze-6.0-2.spec
2 # 文件功能: lze 软件包描述信息
3 # 文件作者: 纵横软件制作中心雨亦奇 国防大学研究生二队赵建利
4 # 修改时间: 2001.10.19
5
6 Name: lze
7 Version: 6.0
8 Release: 2
9 Summary: 小赵全屏幕中英文多窗口多功能编辑器(LINUX/UNIX 系统适用)
10 Group: Applications/Editors
11 License: Share
12 Vendor: 纵横软件制作中心
13 Packager: 雨亦奇(zhsoft@371.net)
14 Source: http://zhsoft.myetang.com/lze-6.0-2.src.tgz
15 Prefix: /usr
16 Requires: /bin/sh
17 Provides: lze-edit
18
19 %description
20 小赵编辑器，是为使用 SCO UNIX,LINUX 多用户系统的广大用户专门设计的全屏幕多窗
21 口中英文多功能编辑器。
22 它主要有以下十大特点：1.全屏幕菜单操作。2.显示方式多样。3.块操作丰富。4.十
23 字制表功能强大。5.多窗口操作灵活自如。6.文件操作功能齐全。7.解释输出功能独具特
24 色。8.自带中文输入法（增强五笔和增强拼音），实用方便。9.十六进制编辑功能，如虎
25 添翼。10.即时翻译,按到即译。
26 总之，小赵编辑器会成为您在 UNIX,LINUX 系统上编制程序和书写一般性文稿的好帮手。
27 它将在工作中助您一臂之力，轻松上阵，游刃有余!
28
29 %prep
30 echo "预处理脚本程序(prep)开始执行"
31 %setup
32
33 %build
34 echo "编译连接脚本程序(build)开始执行"
```

```
35 make
36
37 %install
38 echo "安装脚本程序(install)开始执行"
39 make install
40
41 %clean
42 echo "建包结束后清理脚本程序(clean)开始执行"
43
44 %pre
45 echo "安装前执行脚本程序(pre)开始执行"
46
47 %post
48 echo "安装后执行脚本程序(post)开始执行"
49
50 %preun
51 echo "卸载前执行脚本程序(preun)开始执行"
52
53 %postun
54 echo "卸载后执行脚本程序(postun)开始执行"
55
56 %veryfiscript
57 echo "软件包校验脚本程序(verifyscript)开始执行"
58
59 %triggerin -- xiuwu
60 echo "软件包安装时触发脚本程序(triggerin)开始执行"
61
62 %triggerun -- yuntaishan < 2.0
63 echo "软件包卸载前触发脚本程序(triggerun)开始执行"
64
65 %triggerpostun -- dapubu
66 echo "软件包卸载后触发脚本程序(triggerpostun)开始执行"
67
68 %files
69 %defattr (-,root,root)
70 %config /etc/funkey.def
71 %config /etc/inputme.def
72 %doc /usr/doc/lze-6.0/README
73 %doc /usr/doc/lze-6.0/LICENSE
74 /usr/bin/lze
75 /usr/bin/lzeime.py
76 /usr/bin/lzeime.wb
77 /etc/wbzc.dat
78
```

```
79 %changelog
80 * Tue Aug 18 1998 雨亦奇 <zhsoft@371.net>
81 - 内置拼音,五笔输入法
82 * Fri May 01 1998 雨亦奇 <zhsoft@371.net>
83 - 增加多窗口操作
84 * Mon Mar 24 1997 雨亦奇 <zhsoft@371.net>
85 - 增加块操作命令
86
```

该描述文件包括以下几方面的内容:

一、注释行

见第 1-4 行。

它以#号开头,起注解作用,可帮助用户理解所写的内容,但对软件包的生成不起任何作用。此文件中,注释行集中在文件首部。实际上,它可位于描述文件的任何位置。

二、文件头

见第 6-17 行。

文件头描述软件包的基本信息,它包含若干个域,其中有必选的域,也有可选的域。一个域占用一行,其描述格式为:

域名 : 域值

注意: 域名不分大小写,并且域值不能为空。

文件头必选域有以下六个:

1. Name :

此域定义软件名。

2. Version :

此域定义版本号。仅当软件较以前有较大改变时才增加版本号。注: 版本号中不能含减号(-)字符。

3. Release :

此域定义释出号。若软件较以前改变较小,则仅增加释出号,不改变版本号。注: 释出号中亦不能含减号(-)字符。

RPM 利用上述的 Name(软件名),Version(版本号),Release(释出号)及体系号来命名软件包,如本例输出的包裹文件名为 lze-6.0-2.i386.rpm。

4. Summary :

此域定义软件包简介,为一句话说明。

5. Group :

此域定义软件所属类别,详见精通 RPM 之查询篇,本例的 Applications/Editors

表示本软件属"应用/编辑器"类。

6. License :
此域定义软件适用的许可证或版权规则。该域也可用 Copyright(版权)来定义,二者同意。许可证具体有: GPL(通用公共许可证,自由软件适用),BSD,MIT,Public Domain(公共域),Distributable(贡献),Commercial(商业),Share(共享)等。

文件头可选的域包括如下几类:
1. 基本信息

1.1 Vendor :
此域定义软件的供应商(销售商)。

1.2 Distribution :
此域定义软件所属的发行版,这是软件包制作者自己的分类。通常,一个发行版由若干个软件包构成。如我想做一个名为"熊猫'95"的发行版,则其中每个软件包(如竹叶 95)的描述文件都应有这么一行:
Distribution : 熊猫'95

1.3 Icon :
此域指定软件包所用的图标文件名。此文件为 GIF 或 XPM 格式,必须存放在 RPM 的%_sourcedir(源码目录)宏所指示目录下,默认为/usr /src/dist/SOURCES。RPM 本身并不使用图标,但它将图标文件内容存贮到包裹文件中,安装时亦存贮到 RPM 数据库中。此图标可被图形界面 的 RPM 包管理工具使用,用以改善界面效果,增加可视性。如下例指示软件包使用 panda.xpm 作为图标:
Icon : panda.xpm

1.4 Packager :
此域定义打包者,亦即建立此软件包的人或公司。书写格式是:
打包者的名字〈电子信箱或相关网页〉
请参考描述文件第 13 行。

1.5 Serial :
此域定义软件序列号,也可使用域名 Epoch。软件序列号为一整数,由打包者指定,它应随着版本号的增加而不断增加,并且始终保持数值的唯一。软件序列号可被用来说明软件包之间的依赖关系。下例指定软件包序列号为 4:
Serial : 4
或用:
Epoch : 4

1.6 URL :
此域定义包含打包软件有关信息的网页地址。如:
URL : http://devplanet.fastethernet.net/gxedit.html

2. 依赖相关
依赖是 RPM 用来描述软件包之间关系的。一个软件包依赖的东西 RPM 称作功能,它可以是真实存在的软件包,也可以是虚拟的软件包(虚包)。虚包没有版本号。依赖相关的域有:

2.1 Provides :
此域定义软件包提供的功能,可重复多行。其描述格式为:

```
Provides : 功能 1 [,功能 2] ...
```

注: []所括为可选项,多个功能之间以逗号或空格分隔。
软件包所提供的功能一般是以虚包形式存在的共享库。当有多个软件包均提供相同的服务时,常用虚包来表示其服务。如,一个邮件客户端软件允许用户使用不同的 看信方式(文本形式,HTML 形式等),可以要求任何一个看信程序必须提供 mail-reader 虚包。这样,看信程序的描述文件应有这么一行:
Provides : mail-reader
如此它才能被邮件客户端使用。

2.2 Requires :
此域定义软件包所需的功能,可重复多行。其描述格式为:

```
Requires : 功能 1 [比较符 1 [序列号 1:]版本号 1[-释出号 1]] [,功能 2 [比较符 2 [序列号 2:]版本号 2[-释出号 2]]] ...
```

其中: * []所括为可选项;
* 比较符可使用<(小于),>(大于),=(等于),>=(大于等于)或<=(小于等于);
* 序列号不选时,RPM 默认为 0;
* 功能之间的逗号可选,也可使用空格进行分隔。
例子:Requires: aaa, bbb >= 3.0, ccc < 2:5.0-1
注: 本例定义生成的包在安装时需要系统有如下功能:
(1) aaa(系统中已安装 aaa 包,或者已安装软件包中有软件包提供 aaa 虚包);
(2) bbb 包已安装且版本要求大于等于 3.0;
(3) ccc 包已安装且版本要求小于序列号为 2,版本号为 5.0 且释出号为 1。
RPM 在进行版本比较时,执行比较的顺序是; 先版本号,再释出号,最后比较序列号。通过比较,确定哪个版本较新,哪个版本较老。

2.3 Conflicts :
此域定义有哪些功能与本软件包相冲突(不能共存)。此域亦可在描述文件中书写多次。其描述格式形同 Requires 域,为:

```
Conflicts : 功能 1 [比较符 1 [序列号 1:]版本号 1[-释出号 1]] [,功能 2 [比较
```

```
符 2 [序列号 2:]版本号 2[-释出号 2]]] ...
```

其中: * []所括为可选项;
* 比较符可使用<(小于),>(大于),=(等于),>=(大于等于)或<=(小于等于);
* 序列号不选时,RPM 默认为 0;
* 功能之间的逗号可选,也可使用空格进行分隔。
举个例子:
Conflicts : xxx=1:2.0 yyy>=3.0
注: 本例阐明生成的包冲突的功能有:
(1) 当系统中 xxx 包版本等于序列号为 1 且版本号为 2.0 时;(2) 当系统中 yyy 包版本大于等于 3.0 时。
*** 依赖关系的自动实现 ***
一般情况下,当 RPM 建立一个软件包时,它要执行/usr/lib/rpm 目录下的两个小程序。一个是 find-requires,用于查找软件包所需的 共享库,这些库将以虚包的形式加入到该软件包所需的功能(Requires)之中。另一个是 find-provides,它用于查找软件包所提供的共享 库,这些库将以虚包的形式加入到该软件包所提供的功能(Provides)之中。这两个程序都是 SHELL 程序,代码量虽小,但确实帮了软件包制作者一个 大忙--不必劳心费神地自己写这样的依赖关系了,因为程序均自动完成了。
下面三个域用于指示 RPM 是否执行这两个程序。

2.4 Autoreq :
此域用于指示 RPM 是否自动查找软件所需的共享库。仅当域值为 no 或 0 时,RPM 不执行 find-requires 程序,否则均执行该程序。

2.5 Autoprov :
此域用于指示 RPM 是否自动查找软件提供的共享库。仅当域值为 no 或 0 时,RPM 不执行 find-provides 程序,否则均执行该程序。

2.6 Autoreqprov :
此域用于指示 RPM 是否自动查找软件所需的共享库与其提供的共享库。仅当域值为 no
或 0 时,RPM 不执行 find-requires 与 find-provides 两个程序。此域相当于同时设定 Autoreq
与 Autoprov 域值为指定之值。
注: 上述三个域在描述文件中,它们之间因为顺序的不同而结果会有所不同,一般以最后一个为准。如:

```
Autoreq : yes
Autoreqprov : no
Autoprov : yes
```

注: 本例虽然第一行允许执行 find-requires,但第二行又不允许 find-requires 与 find-provides 两个程序运行,而第三行允 许 find-provides 运行,所以依照执行顺序,结果为不允许执行 find-requires,而允许执行 find-provides。
又如:

```
Autoreq : no
Autoreqprov : yes
Autoprov : no
```

注: 本例的结果为允许执行 find-requires,而不允许执行 find-provides。

3. 系统相关
RPM 制作软件包时,可以为其指定适用的 CPU 体系或操作系统,也可为其指定不适用的 CPU 体系或操作系统,这样,当 RPM 发现当前的 CPU 体系或操作系 统与软件包要求的不兼容时,将中止软件包的制作。RPM 默认的当前 CPU 体系由宏%_arch 定义,一般为 i386。RPM 默认的当前操作系统由 宏%_os 定义,一般为 linux。读者可以通过查看/usr/lib/rpm/macros 宏定义文件得到。
下面四个域说明软件包的适用范围:

3.1 Excludearch :
此域定义软件包不适用的体系。RPM 可选的体系名请参见/usr/lib/rpm/rpmrc 文件中的 arch_canon 项目。
软件包不适用于某个体系,可能有两方面的原因。一是该软件还没有移植到所定义的体系上;二是该软件含有特定的机器码(汇编语言),它与别的体系不兼容。
此域描述格式为:

```
Excludearch : 体系 1 [体系 2] ...
```

注: []所括为可选项,各体系之间以空格分隔。
如果当前体系在此域值之中,则 RPM 制作软件包时将报错退出,请看下面的例子。
在 lze-6.0-2.spec 文件头部分加入一行:
Excludearch : i386
再运行建包命令 rpm -bb(精通 RPM 之制作篇(下)将讲到):

```
# rpm -bb lze-6.0-2.spec
Architecture is excluded: i386
#
```

由上看出,RPM 提示了"体系不适用: i386"的错误。

3.2 Exclusivearch :
此域定义软件包适用的体系。其描述格式与 Excludearch 类似:

```
Exclusivearch : 体系 1 [体系 2] ...
```

注: []所括为可选项,各体系之间以空格分隔。
假如在 lze-6.0-2.spec 文件头加入一行:
Exclusivearch : i386 sparc
再运行建包命令将会怎么样:
rpm -bb lze-6.0-2.spec
Executing: %prep
预处理脚本程序(prep)开始执行
Executing: %build
编译连接脚本程序(build)开始执行
Executing: %install
安装脚本程序(install)开始执行

```
Processing files: lze
Finding Provides: (using /usr/lib/rpm/find-provides)...
Finding Requires: (using /usr/lib/rpm/find-requires)...
Provides: lze-edit
PreReq: /bin/sh
Requires: /bin/sh ld-linux.so.2 libc.so.6 libc.so.6(GLIBC_2.0)
libc.so.6(GLIBC_2.1)
Wrote: /usr/src/dist/RPMS/i386/lze-6.0-2.i386.rpm
#
```

看,此次建包(lze-6.0-2.i386.rpm)成功了,因为当前的体系(i386)正好适用。

3.3 Excludeos :
此域定义软件包不适用的操作系统。RPM 可选的操作系统请参考文件
/usr/lib/rpm/rpmrc 中的 os_canon 项目。
其描述格式为:

```
Excludeos : 操作系统 1 [操作系统 2] ...
```

注: []为可选项,操作系统之间以空格分隔。例如:
Excludeos : irix aix solaris
注: 如将此行加入到 lze 的描述文件中,则它会指示 RPM 不在 irix,aix,solaris
这三个操作系统上建立 lze 软件包。如果当前操作系统是三者之一,则 RPM 会报

错并中止软件包的制作。
如:

```
# rpm -bb lze-6.0-2.spec
OS is excluded: Solaris
#
```

3.4 Exclusiveos :
此域定义软件包适用的操作系统。其描述格式为:
Exclusiveos : 操作系统 1 [操作系统 2] ...
注: []为可选项,操作系统之间以空格分隔。例如:
Exclusiveos : linux solaris

4. 目录相关
4.1 Prefix :
此域定义可重定位的目录前缀,可在描述文件中书写多次。其描述格式为:

```
Prefix : 目录前缀 1 [目录前缀 2] ...
```

注: []为可选项,各目录前缀之间均以空格分隔。例如:
Prefix : /usr /etc
它也可写作:
Prefix : /usr
Prefix : /etc
RPM 利用可重定位的目录前缀,实现了软件包的重定位安装,使软件中的文件不必固定在某个绝对位置,这种做法很好。LZE 软件包描术文件 lze- 6.0-2.spec 中就定义了一个可重定位的前缀/usr(见第 15 行),这样,安装时就可将该包中在/usr 目录下的文件重定位到用户指定的目录, 如:
rpm -i --prefix /tmp lze-6.0-2.i386.rpm
#
或者:
rpm -i --relocate /usr=/tmp lze-6.0-2.i386.rpm
#
注: 此命令安装 lze 包,将其中含/usr 重定位目录前缀的文件定位到/tmp 目录。如包中
的/usr/bin/lze 文件安装后,因重定位而成了/tmp/bin/lze。(RPM 安装命令使用方法请参考精通 RPM 之安装篇)

4.2 Buildroot :
此域定义的是软件包所包含的文件共有的根目录,此根目录仅供 RPM 建立软件包时使用。即当 RPM 建立软件包时,将设定此目录为根(调用 chroot 函数),提取所需文件,生成软件包。

例如: 当 Buildroot 设定为/tmp 时,对于 LZE 包描述文件中所包含的/usr/bin/lze 文件,RPM 实际打包的则是/tmp/usr/bin/lze,但对生成的包查询后可以发现:原文件名并未改变,还是/usr/bin/lze。
如此说来,这就很有意思了。一般用户通过设定 Buildroot,也可以象超级用户(root)那样自由地建立各种各样的软件包了,即使包中有那些唯有超级用户才可以操作的目录或文件。安装这样的包与安装由超级用户建立的包,是没有什么分别的。
此域的描述格式很简单:
Buildroot : 目录
如,上例可定义为:
Buildroot : /tmp

5. 源码相关
下列四个域均是为制作源码包而设计的。源码包里有什么?用户可以通过查询包的文件列表得到,命令是"rpm -qpl 源码包文件"(请参阅精通 RPM 之查询篇有 关内容)。一般情况下,源码包里有这么四类文件: 一是程序源码(SOURCE),二是源码补丁(PATCH),三是软件包描述文件,四是图标文件(ICON)。通过安装源码包,用户可以轻松地实现现场编 译、连接和应用,同时更方便了软件开发者与软件包制作者:他们维护程序容易了,并且维护过后可以很快地生成执行代码包与源码包。这,也是所有人钟爱 RPM 的重要原因之一。

5.1 Source :
此域定义 RPM 打包时要包含的程序源码文件。这些文件一般先用 tar 命令打包,然后再用 gzip 压缩。一个描述文件中可包含多个 Source 域,当有多个 这样的域时,需要进行编号:第 1 个编为 Source0(也可直接用 Source),第 2 个编为 Source1,第 3 个编为 Source2 等等。此域的描述 格式为:

```
Source[编号] : 源码文件
```

注: []所括为可选项。具体用法如:
Source0 : lze-6.0-2.tar.gz
Source1 : lzeime-wb-6.0-2.tar.gz
Source2 : lzeime-py-6.0-2.tar.gz
Source3 : lze-lib-6.0-2.tar.gz
注: 此域域值可以采用 URL(统一资源定位)的形式,如 LZE 描述文件第 14 行。采用这种形式,主要是给其它用户提供该源码的位置信息。在 RPM 制作源包时, 它提取的是最后的文件名 lze-6.0-2.tar.gz,而不是
http://zhsoft.myetang.com/lze- 6.0-2.tar.gz(URL 前面的内容被 RPM 忽略了)。

5.2 NoSource :
在上例中,假如在打包时不想包含 Source1 与 Source2 定义的文件,那该怎么办?办法之一是将其所在行删除掉;

办法之二是将其所在行注释掉(即所在行前面加#号);
办法之三就是定义 Nosource 域,此域可重复。其描述格式为:

```
NoSource : 源码域编号
```

本例可写作:
NoSource : 1
NoSource : 2
注: 其中的 1 与 2 为编号,表示 Source1 和 Source2。
注意: 如果软件包描述文件中没有NoSource域,则RPM生成的源码包名字格式为"软件名-版本号-释出号.src.rpm"。如果使用了 NoSource 域,则 RPM 生成的源码包名字格式为"软件名-版本号-释出号.nosrc.rpm"(单从名字就可看出源码包包含的文件不完整)。

5.3 Patch :
Patch 的本义是补丁,用在这里指的是源程序的补丁,它是用 diff 命令比较新老源程序所产生的输出(命令为"diff -Nur 旧文件 新文件 >补丁文件"),而系统中的 patch 命令又可利用此输出将老版本的源程序升级为新版本。
此域定义 RPM 制作源码包时所要包含的补丁文件,该文件的命名建议用"软件名-版本号.补丁功能.patch"的格式。一个软件包描述文件中可有多个 Patch 域,当有多个这样的域时,也需要象 Source 域那样进行编号(注:第 1 个域编为 Patch0,也可省略 0,用 Patch)。
此域的描述格式为:

```
Patch[编号] : 源码补丁文件
```

注: []所括为可选项。具体用法如:
Patch0 : blather-4.5-bugfix.patch
Patch1 : blather-4.5-config.patch
Patch2 : blather-4.5-somethingelse.patch
注: 此域的域值也可以象 Source 域一样,采用 URL 的形式,RPM 仅提取其中的文件名供其使用。

5.4 NoPatch :
此域的功能类似 NoSource,其定义的编号对应的补丁文件 RPM 不作打包处理。此域在描述文件中可重复出现。如上例,若不想让源码包包含 Patch0 与 Patch2 域所指示的补丁文件,则可在描述文件写上这么两行:
NoPatch : 0
NoPatch : 2
注意: 如果软件包描述文件中没有NoPatch域,则RPM生成的源码包名字格式为"软件名-版本号-释出号.src.rpm"。如果使用了 NoPatch 域, 则 RPM 生成的源码包名字格式为"软件名-版本号-释出号.nosrc.rpm"(单从名字就可看出源码包包

含的文件不完整)。

三、功能段

见第 19-86(即文件头以下的部分)。
何谓功能段?可以这么说,功能段是描述软件包的重要数据和操作指令的段落,它包括段名与段内容两部分。没有功能段,RPM 便制作不出任何包裹文件。功能段的段名都是以百分号(%)开始的,占用一行。功能段的段内容范围是这样界定的:它从该功能段段名下一行开始到下一个功能段段名的前一行或到描述文件结束。如 LZE 描述文件,%description 段是从第 19 行到第 28 行(%prep 段从第 29 行开始),第 19 行为段名,第 20-28 行为段内容。而%prep 段是从第 29 行到第 32 行(第 33 行%build 段开始),其段名在第 29 行,段内容在第 30-32 行。另外要注意的是,各个功能段的位置是 自由的,可放在文件头以下的任何位置,不必拘泥某一固定位置。
必选的功能段
描述文件中,必选的功能段有:

1. %description

本段是描述段,段的内容是对软件包进行较为详细的介绍,不象文件头的 Summary 域仅用一句话说明。介绍的文本形式自由,可任意换行,不受限制。具体请参见 LZE 描述文件第 20-27 行。
本段段名描述格式是:

```
%description [子包选项]
```

其中,子包选项的格式为:[-n] 子包名
注: []所括为可选项。

三种形式的描述段段名:
(1) 段名格式为"%description"时:
本功能段描述的内容是关于父包的。父包也可叫作主软件包,它用软件名来命令,其名字格式是:软件名-版本号-释出号.体系.rpm。如:lze-6.0-2.i386.rpm。
(2) 段名格式为"%description 子包名"时:
本功能段描述的内容是关于子包的。子包选项中没有-n 选项时,子包是用软件名加子包名的形式命名,格式为: 软件名-子包名-版本号-释出号.体系.rpm。如分成两个子包的 LZE 软件:lze-bin-6.0-2.i386.rpm(执行程序包),lze-config-6.0-2.i386.rpm(配置文件包)。
(3) 段名格式为"%description -n 子包名"时:
本功能段描述的内容也是关于子包的。当子包选项中有-n 选项时,子包直接采用子包名的形式命名。它不包含软件名,命名格式为: 子包名-版本号-释出号.体系.rpm。如分成两个子包的 LZE 软件: bin-6.0-2.i386.rpm(执行程序包),config-6.0-2.i386.rpm(配置文件包)。注意:这种类型的子包内容通常是可被 其它软件包共用的函数库,如果专用,则尽量不要采用这样形式来定义子包。

2. %files
本段是文件段,它定义的是软件包需要包含哪些文件。本段通常放在描述文件尾部,以便于添加文件名,便于编辑。
本段段名描述格式为:

```
%files [子包选项] [-f 文件名]
```

注: []所括为可选项。
当没有任何选项时,本段内容定义的是父包要打包的文件列表;
当有子包选项时,本段内容定义的则是子包要打包的文件列表;
当选择-f 选项时,RPM 除了从文件段读取打包文件列表外,还将从指定的文件中读取要打包的文件列表。指定的文件中,一个文件名占用一行。此选项方便了软件包制作者,他们可以通过程序自动产生有关软件的文件列表,并将其写入到一个特定的文件中,这样制作软件包时,只需引用一下这个文件,RPM 就会自动从这个文件中读取文件名并将其加入包中。如果没有此选项,软件包制作者只能在文件段里,将要打包的文件名一个一个写进去,有点麻烦。

文件段的内容格式为:
[修饰符 1 [修饰符 2] ...] 文件名
其中:修饰符是可选的,一个文件可以有多个修饰符,文件名必须以/开头(绝对路径形式)。
修饰符有以下几类:

(1) 文件相关
* %doc :
此修饰符设定文件类型为说明文档(参见 LZE 描述文件第 72,73 行);
* %config :
此修饰符设定文件类型为配置文件(参见 LZE 描述文件第 70,71 行);
* %config(missingok) :
此修饰符设定文件类型为配置文件,且此文件可丢失。即使丢失了,RPM 在卸载软件包时并不认为这是个错误,并不报错。
此修饰符通常用于那些软件包安装后建立的符号连接文件,如
/etc/rc.d/rc2.d/S55named 等。此类文件在软件包卸载后可能需要删除,所以丢失了也不要紧。
* %config(noreplace) :
此修饰符设定文件类型为配置文件,且如果安装时系统中有同名的文件,则软件包中的这个文件将换个名字安装,其文件名后缀加个.rpmnew。(如果不用此 修饰符,则安装时 RPM 若发现有同名文件,则 RPM 会将系统中的这个文件换个名字,其后缀加上.rpmorig,而软件包中的文件则还用原来的名字。)在 软件包卸载时,系统中的同名文件被 RPM 换个名字保存起来,其后缀加上了.rpmsave。
如描述文件的文件段中定义了这么一行:
%config(noreplace) /etc/hello
则制成的包在安装时,若系统中已有此文件/etc/hello,则 RPM 会提示:

warning: /etc/hello created as /etc/hello.rpmnew
这表明包中的/etc/hello 文件被创建为/etc/hello.rpmnew 文件了。
如果卸载这个软件包,则系统中的/etc/hello 将会改名为/etc/hello.rpmsave。
* %ghost :
此修饰符所修饰的文件,其内容不被包含到软件包中。这样的文件一般是日志文件(log file)一类的文件,其文件属性(文件名,属主,属组等)很重要,但是文件内容并不重要。用此修饰符后,RPM 仅将其文件属性加入包中。
* %attr :
此修饰符设定文件的属性信息,使用格式为:
%attr(权限,属主,属组)
注: 权限常用数字形式(八进制),属主和属组可以是数字,也可以是字符串。如果文件的权限,属主和属组想使用系统默认值,则可用减号(-)表示它。
如下例采用两个修饰符,定义/etc/funkey.def 文件的权限为 755,属主默认,属组为 root,类型为配置文件:
%attr(755,-,root) %config /etc/funkey.def
* %verify :
此修饰符设定文件需要校验的那些属性。这些属性有:owner(属主),group(属组),mode(权限),md5(MD5 检查和),size(大小),maj(主设备号),min(从设备号),symlink(符号连接),mtime(最后修改时间)。
此修饰符使用格式为:
%verify([not] owner group mode md5 size maj min symlink mtime)
注: not 可选。当选用 not 时,表明需要校验除选定属性以外的那些属性。
如下例指示 RPM 校验/dev/ttyS0 文件时,要校验其权限,MD5 检查和,大小,主设备号,从设备号,符号连接和最后修改时间共七项属性信息:
%verify(mode md5 size maj min symlink mtime) /dev/ttyS0
这也可以采用 not 选项来实现,因为除去属主 owner 和属组 group 两项属性,剩下的就是需要校验的属性了:
%verify(not owner group) /dev/ttyS0

(2) 目录相关

```
* %docdir :
```

此修饰符定义说明文档前缀,这样,后面所有含指定文件名作为前缀的文件,RPM 打包时会将其类型统一设定为说明文档。
例如某描述文件的文件段中有这么三行:

```
/root/readme
%docdir /root
/root/mydoc.txt
```

此例指明/root 为说明文档的前缀,因为/root/mydoc.txt 在%docdir 的下一行,

所以 RPM 打包时会设定此文件的类型为说明文档。而/root/readme 文件则不做此设定,因为它在%docdir 定义之前。
通过此修饰符,用户可以很方便地设定说明文档一类的文件,因为它们通常固定在某个目录下面,有着共同的前缀。
* %dir :
RPM 在制作软件包时,如果要打包的文件是个目录,那么 RPM 会将该目录下面的所有文件包含到软件包中。(注意:如果要打包的文件是个符号连接,此符号连 接又指向一个目录,则 RPM 并不会将其视作目录,只会把它当为普通文件处理。)如果仅想将这个目录名包含到软件包中,制作者用此修饰符修饰一下这个目录名 就行了。
如: /etc 是个系统目录,其下有多个文件,如果想将其均加入包中,描述文件的文件段里可写上这么一行:
/etc
如果仅想包含此目录,则可用:

```
%dir /etc
```

(3) 另类修饰符
此类只有一个%defattr。说它是另类修饰符,是由于它设定的是默认的文件属性,而非特定的某个文件。它一般放在文件段内容的第一行。
其使用格式为:

```
%defattr(权限,属主,属组)
```

其中: 权限,属主和属组都可以使用减号(-)。使用减号的属性将由系统设定。
例如: %defattr(022,zzz,zhsoft) 设定其后的所有文件权限为 022,属主为 zzz,属组为 zhsoft;又如: %defattr(-,zzz,-) 则是设定其后的所有文件属主为 zzz,权限与属组由系统设置。
可选的功能段
描述文件中,可选功能段的内容都是些脚本程序。(LZE 描述文件中多个脚本程序中仅含一个 echo 命令)
可选的功能段的描述格式为:

```
功能段名 [子包选项]
```

注: 子包选项为"[-n] 子包名"。当无子包选项时,段内容描述的是父包的脚本程序。当有子包选项时,段内容则是描述子包的脚本程序。

可选的功能段可分为如下三类:
1. 建包用功能段:

RPM 通过源程序来建立一个软件包时,要执行预处理,编译,安装和清理四项操作,分别对应于%prep,%build,%install 和%clean 四个段。
下面按其执行顺序逐段进行说明:
1.1 %prep :
此为预处理段,其内容为预处理脚本程序。该程序完成以下任务:
* 建立软件编译用目录;
* 将源程序解压缩;
* 通过打补丁,升级源程序;
* 执行其它一些操作,使源程序随时可进行编译。
在此脚本程序中,可使用如下两个宏命令:
1.1.1 %setup
这个宏利用系统中的 gzip 与 tar 等命令,来解压源程序包。RPM 会自动探测源程序包是否压缩,如果压缩,它会用 gzip 将其解压缩,否则直接用 tar 命令展开包中文件。其使用格式为:

```
%setup [-n name] [-c] [-D] [-T] [-b N] [-a N]
```

注: []所括为可选项。
(1) 当没有任何选项时:
这个宏用来解压默认的源程序包(由文件头 Source 或 Source0 域指定)。注意:源程序包中的文件应用"软件名-版本号"作为其上层目录,这 样%setup 宏就可以正常工作。如果不以"软件名-版本号"作为其上层目录,则%setup 宏工作时有一个指令"cd 软件名-版本号"(转目录)会因为系统中没有此目录而出错退出(除非在此宏上面加上建立此目录的命令)。如 LZE 软件源程序所在的目录为 lze-6.0,我需要用命令"tar cvzf lze-6.0-2.src.tgz lze-6.0"将源程序打包并压缩,这样的包就可以被%setup 宏正确使用了。
下面是%setup 宏命令所执行的一系列命令: (指令前面为行号)

```
1 cd /usr/src/dist/BUILD
2 echo "预处理脚本程序(prep)开始执行"
3 cd /usr/src/dist/BUILD
4 rm -rf lze-6.0
5 /bin/gzip -dc /usr/src/dist/SOURCES/lze-6.0-2.src.tgz | tar -xvvf -
6 STATUS=$?
7 if [ $STATUS -ne 0 ]; then
8exit $STATUS
9 fi
10 cd lze-6.0
11 [ `/usr/bin/id -u` = '0' ] && /bin/chown -Rhf root .
12 [ `/usr/bin/id -u` = '0' ] && /bin/chgrp -Rhf root .
13 /bin/chmod -Rf a+rX,g-w,o-w .
14 exit 1
```

看,第10行就有一个转到lze-6.0目录的命令,如果没有这个目录,程序就会出错退出了。也许你要问:这些指令你是怎么知道的?其实这很简单,只要 在%setup宏下面加上一句"exit 1"命令,让预处理脚本程序非正常退出即可。这样RPM所执行的预处理脚本程序作为临时文件在其退出时并未删除,只要看一下这个文件(在/var/tmp 目录下以rpm-tmp开头)就知道%setup宏命令做什么了。

(2) -n name :
上面已经谈到,源程序包中的文件应采用"软件名-版本号"作为上层目录。如果用了别的什么目录(如name),%setup宏无法正常工作,那该怎么办? 没关系,可以用-n选项,引用一下这个目录(name)就行了。假如我的LZE源程序包中的文件是以lze为上层目录,那么我就可以用"%setup -n lze"宏命令来解压缩该包。

(3) -c :
此选项的作用是创建上层目录("软件名-版本号"目录)并转到这个目录。对于LZE软件,其效果相当于在上例的第4行与第5行之间加上这么两行命令:
mkdir -p lze-6.0
cd lze-6.0
它适用的情况是:有的源程序包是在源程序所在目录下打的包,所以其中的文件都没有上层目录。这样的话,要想正确解压,必须创建上层目录。

(4) -D :
本选项的作用是在解压源程序包之前不要删除软件的上层目录(软件名-版本号)。在上例中,其效果是不执行第4行的命令(rm -rf lze-6.0)。

(5) -T :
本选项的作用是不解压默认的源程序包(由文件头的Source或Source0域所定义)。在上例中,其效果是不执行第5-9行的命令:第5行是解压源程 序包(用gzip -dc将包的内容解压缩到管道中,再由tar -xvvf -从管道中读取数据并展开),第6-9行是检查解压命令的返回值,非0时执行非正常退出。

(6) -b N :
本选项指示RPM在转到上层目录前解压第N个源程序包(由文件头SourceN域定义)。这适用于含上层目录的源程序包。注意:如果使用此选项时不同时使 用-T选项,则RPM解压的是两个源程序包,一个是默认的包(由Source或Source0域定义),一个是-b选项指定的包(由SourceN域定 义)。这样,当N等于0时,默认的源程序包将被解压两次。所以,如果想仅解压指定源程序包,请同时使用-T选项,以禁止解压默认的源程序包。
下面的宏命令仅解压第1个源程序包,然后转到上层目录:
%setup -b 1 -T

(7) -a N :本选项指示RPM在转到上层目录后再解压第N个源程序包(由文件头SourceN域定义)。这适用于不含上层目录的源程序包。使用本选项时,一般加上-c选项,以创建上层目录并转到此目录。注意:如果使用此选项时不同时使用-T

选项,则 RPM 解压的是两个源程序包,一个是默认的包(由 Source 或 Source0 域定义),一个是-a 选项指定的包(由 SourceN 域定义)。这样,当 N 等于 0 时,默认的源程序包将被解压两次。所以,如果想仅解压指定 源程序包,请同时使用-T 选项,以禁止解压默认的源程序包。

下面的宏命令让 RPM 先转到上层目录,再仅解压第 2 个源程序包:
%setup -T -a 2
1.1.2 %patch
此宏利用系统中的 patch 命令,来给指定的源程序包打补丁,从而将程序升级。其使用格式为:
%patch [-P N] [-p N] [-b name] [-E]
注: []所括为可选项。
为了说明下列选项的作用,我们为 LZE 软件包描述文件中定义三个补丁文件:
Patch0 : lze-patch.zero
Patch1 : lze-patch.one
Patch2 : lze-patch.three

(1) 当没有任何选项时:
没有任何选项时,该宏使用的是默认的补丁文件(第 0 个补丁文件),即由文件头 Patch 或 Patch0 域所定义的文件(LZE 包使用 lze-patch.zero)。
该宏在执行时,扩展为以下指令:
echo "Patch #0:"
patch -p0 -s < /usr/src/dist/SOURCES/lze-patch.zero
注: 第一行指令是利用 echo 命令向屏幕输出字符串"Patch #0:"。第二行指令则是利用 patch 命令读取补丁文件 lze-patch.zero 升级源程序。
patch 命令用了两个选项:(有关 patch 命令用法,详见其用户手册)
* -p : 这个选项用于确定 patch 所要操作的文件。它针对补丁文件头部的文件名,删除名字中指定数目个斜杠(/)前面的所有字符,从而得到要操作的文件名。如补 丁文件里有个文件名/usr/zzz/src/lze.c,则用-p0 时 patch 操作的文件名不变,用-p1 时则变为 usr/zzz/src /lze.c,用-p2 时则变为 zzz/src/lze.c,如用-p4 则操作的文件名变为 lze.c。
* -s : 这个选项指示 patch 在打补丁过程中不输出任何信息,即使有错误发生。

(2) -P N :
使用此选项以指示 RPM 使用第 N 个补丁文件(由文件头 PatchN 域定义)。如想让 RPM 使用 LZE 的第 2 个补丁文件 Patch2(lze-patch.three)时,可使用"-P 2"来指定。

(3) -p N :
此选项与其参数是由%patch 宏直接传给 patch 命令的。请参见上面 patch 命令所用的-p 选项的介绍。

(4) -b name :
当有多个 patch 命令操作同一个文件时,patch 会将原文件换名保存(其后缀变

作.orig),如 lze.c 会变作 lze.orig。如果想用别的名 字作后缀,则可用-b 设置一下,这样原文件会换名为"原文件名+后缀",如用-b ZZZ 时,lze.c 会换名保存为 lze.cZZZ。
此选项在执行时,实际上是给 patch 命令传递了一个选项及参数,即--suffix name。

(5) -E :
此选项直接传给 patch 命令,其作用是:如果一个文件打完补丁后内容为空(字节数为 0),则删除这个文件。

1.2 %build :
此为编译段,其内容为编译脚本程序。该程序完成源程序的编译和连接。一个最简单的例子就是程序中仅有一个 make 命令。这适用于大部分情况,因为多数软件均有自己的 makefile,这样通过 make 命令就可实现编译与连接。如果没有 makefile 的话,需要软件包制作者自己在编译段书写上一系列的编译 连接命令。

1.3 %install :
此为安装段,其内容是安装脚本程序。该程序将已编译连接好的执行程序或其它文件存放到指定目录下,这些程序或文件供 RPM 打包时使用。一个最简单的例子就 是程序中仅用一个 make install 命令,从而完成安装。这也需要相应的软件有 makefile 维护文件。没有的话,软件包制作者也得自己写指令。

1.4 %clean :
此为清理段,其内容是清理脚本程序。此程序在 RPM 制作好软件包后才执行,它通常是删除那些编译连接时产生的临时文件或目录,完成缮后工作。

2. 管理用功能段:
此类段用于软件包自身的管理(安装,卸载和校验),包括%pre,%post,%preun,%postun,和%verifyscript 五个功能段。
2.1 %pre :

该段内容为安装前脚本程序。它在软件包安装之前执行,通常是检测操作环境,建立有关目录,清理多余文件等等,为软件包的顺利安装做准备。本段很少使用。
其段名格式为: %pre [子包选项]

2.2 %post :
该段内容为安装后脚本程序。它在软件包安装完成之后执行,常用来建立符号连接,修改系统配置文件,运行 ldconfig 程序等,以利软件的正常运行。
其段名格式为: %post [子包选项]

2.3 %preun :
该段内容为卸载前脚本程序。它在软件包卸载之前执行,主要为卸载做准备。具体如,要卸载的软件包中某个程序当前正在运行时,此脚本程序必须杀掉它,否则无法正确卸载。

其段名格式为: %preun［子包选项］

2.4 %postun :
该段内容为卸载后脚本程序。它在软件包卸载后执行,完成卸载的缮后工作,如将系统配置文件 inetd.conf 改回原来的样子,重新运行一下 ldconfig 命令,将已卸载的共享库从缓冲文件 ld.so.cache 中删除等等。
其段名格式为: %postun［子包选项］

2.5 %verifyscript :
该段内容为校验脚本程序。RPM 校验软件包时,除了执行标准的校验外,如果软件包制作者设定有此校验脚本程序,还将执行之。

其段名格式为: %verifyscript［子包选项］
下面是 XFree86-libs-3.3.6-6.i386.rpm 软件包中的校验脚本程序,它校验的是动态链接库目录/usr/X11R6/lib。校 验时,在/etc/ld.so.cache 文件中查找 /usr/X11R6/lib,如果找不到,则显示"missing",找到则显示"found"。

```
# verifyscript
echo -n "Looking for /usr/X11R6/lib in /etc/ld.so.conf... "
if ! grep "^/usr/X11R6/lib$" /etc/ld.so.conf > /dev/null
then
echo "missing"
echo "/usr/X11R6/lib missing from /etc/ld.so.conf" >&2
else
echo "found"
fi
```

3. 交互用功能段:
这类功能段有%triggerin,%triggerun,%triggerpostun,它们的内容都是 RPM 用于软件包之间交互控制的脚本程序。这些脚本程序都是在系统满足指定的条件下才触发执行的:

1) %triggerin : 段内为安装时触发脚本程序,当其所在软件包与指定软件包仅有一方已安装时,安装另一方将触发此程序执行;

2) %triggerun : 段内为卸载时触发脚本程序,当其所在软件包与指定软件包都已安装时,卸载二者中的任一个将触发此程序执行;

3) %triggerpostun : 段内为卸载后触发脚本程序,只有指定软件包卸载后才触发此程序执行。

3.1 段名格式
它们的段名描述格式均为:

```
交互段名 [子包选项] [-p 解释程序] -- 触发条件 1 [,触发条件 2] ...
```

注: []所括为可选项。子包选项见前面介绍,不赘述。

3.1.1 -p 选项:
此选项用于指定一个解释程序,来解释执行交互功能段的脚本程序。默认情况下,RPM 使
用/bin/sh 来执行脚本(此类脚本用 SHELL 语言编写,也叫 SHELL 程序)。有的 RPM 包则是使用/usr/bin/perl
来执行脚本(此类脚本是用 PERL 这种解释性语言写的),这就需要用-p 选项来指定解释程序为
/usr/bin/perl,如:

```
%triggerin -- sendmail
ln -sf /usr/bin/sendmail /etc/mymailer/mailer
%triggerin -- vmail
ln -sf /usr/bin/vmail /etc/mymailer/mailer
```

注: 此例中定义 package 子软件包安装时触发脚本程序:当触发条件(fileutils>3.0,perl<1.2)满足时,用/usr/bin/perl 执行脚本,即用 print 命令输出字符串"I'm in my trigger!"。

3.1.2 触发条件:
交互功能段的触发条件格式是:

```
功能名 [比较符 版本号]
```

其中:比较符与版本号可选。仅有一个功能名时,表明该功能存在时触发程序执行。比较符可用大于(>),等于(=),小于(<),大于等于(>=)和小于等于(<=)。
如触发条件 bash,又如触发条件 fileutils>3.0,这种使用均合法。
交互功能段最少有一个触发条件。当有多个触发条件时,这些条件间均以逗号(,)分隔,它们之间是"或"的关系,即只要其中有一个条件系统满足,RPM 就将 执行触发脚本程序。如上面介绍-p 选项时举的例子:例子中有两个触发条件 fileutils>3.0 和 perl<1.2,在安装软件包时,只 要有一个条件满足,RPM 就会执行触发脚本,即输出"I'm in my trigger!"。

3.2 交互用功能段的使用
为什么要使用交互用功能段?下面的例子很能说明问题。
假定 mymailer 软件包需要/etc/mymailer/mailer 这个符号连接文件指向当前使用的邮件发送代理程序。如果 sendmail 包安装 了,那么这个符号连接文件应指向/usr/bin/sendmail 程序。如果 vmail 包安装了,那么它应当指向

/usr/bin/vmail 程序。如 果这两个软件包都安装了(实际上,sendmail 与 vmail 彼此是冲突的),那么我们也无需考虑符号连接指向哪个文件了。当然,如果这两个包都未安 装,那么/etc/mymailer/mailer 符号连接文件也没有理由存在了。

上述要求,我们通过为 mymailer 软件包编写触发脚本程序来实现,这些脚本程序在下列事件发生时,将改变/etc/mymailer/mailer 符号连接的内容:
1) sendmail 已安装;
2) vmail 已安装;
3) sendmail 卸载时;
4) vmail 卸载时。

前两个事件触发的脚本程序可以这样写:

```
%triggerin -- sendmail
ln -sf /usr/bin/sendmail /etc/mymailer/mailer
%triggerin -- vmail
ln -sf /usr/bin/vmail /etc/mymailer/mailer
```

这是两个安装时被 sendmail 或 vmail 所触发脚本程序。它们将在下列情况下执行:
1) 在 mymailer 包已安装的情况下,安装或升级 sendmail 包;
2) 在 mymailer 包已安装的情况下,安装或升级 vmail 包;
3) 在 sendmail 包已安装的情况下,安装或升级 mymailer 包;
4) 在 vmail 包已安装的情况下,安装或升级 mymailer 包。
后两个事件触发的脚本程序可以这么写:

```
%triggerun -- sendmail
[ $2 = 0 ] || exit 0
if [ -f /usr/bin/vmail ]
then
ln -sf /usr/bin/vmail /etc/mymailer/mailer
else
rm -f /etc/mymailer/mailer
fi
%triggerun -- vmail
[ $2 = 0 ] || exit 0
if [ -f /usr/bin/sendmail ]
then
ln -sf /usr/bin/sendmail /etc/mymailer/mailer
else
rm -f /etc/mymailer/mailer
fi
```

这两个脚本程序在下列情况下触发执行:
1) 在 sendmail 包已安装的情况下,卸载 mymailer 包;
2) 在 vmail 包已安装的情况下,卸载 mymailer 包;
3) 在 mymailer 包已安装的情况下,卸载 sendmail 包;
4) 在 mymailer 包已安装的情况下,卸载 vmail 包。
为了确保在 mymailer 包卸载后符号连接文件/etc/mymailer/mailer 也被删除,可以在
mymailer 软件包描述文件的%postun 功能段内,加上删除该文件的命令:

```
%postun
[ $1 = 0 ] && rm -f /etc/mymailer/mailer
```

注: %postun 段内为卸载后执行脚本程序,在 mymailer 包卸载后执行。
由上看出,当一个软件包与另一个软件包存在密切关系时,我们可以通过交互用功能段实现某些文件的管理,这不仅扩展了 RPM 软件包管理的功能,又有助于软件包的正常运行。

4. 其它功能段
其它功能段只有一个,即%changelog。这个段的内容是软件维护记录,它记录每次软件维护的时间,维护人及其 EMAIL,维护的项目等。
%changelog 段内容格式为:
* 星期 月份 日子 年份 维护内容
注: 每个维护记录均以*开头,星期,月份均须为英文缩写。维护内容多时可分行编写,
每行开头最好以减号(-)开头。可以采用类似 LZE 方式的维护记录写作格式:(见 LZE 描述文件第 80-85 行)
* 星期 月份 日子 年份 维护人 EMAIL
- 维护内容 1
- 维护内容 2
- ..... (其它维护内容)

精通 RPM 之制作篇（中）

一个 RPM 的软件包描述文件,可以仅生成一个父包或一个子包,也可以生成一个父包和多个子包。通过设定子包选项,可以使生成的子包采用"软件名-子 包名"的标准命名,也可使生成的子包采用自己的名字。一个子包,通常是按照其包含的文件的用途或类型来归并文件进而打成包裹的。象前面的 LZE 描述文件很 简单,它将所有文件都包含进了父包中。我们也可以将文件分类作成子软件包,如可分成执行程序子包(lze- bin),配置文件子包(lze-config)和说明文档包(lze-doc)。我们还可以只分出一个配置文件子包(lze-config),其余文件 均打入父包中(lze)。通过这样详细地分类,有助于用户管理软件包,避免安装多余的

东西,同时也有助于升级。

要想创建子软件包,必须描述以下内容:

1. %package :
用此段创建一个子包。其名字由子包选项控制。子包选项为"[-n] 子包名",不选-n 时,生成的子包文件为"软件名-子包名-版本号-释出号.体系.rpm";选-n 时,生成的子包文件为"子包名-版本号-释出号.体系.rpm"。其应用格式为:

```
%package 子包选项
```

2. Summary

此域必须在%package 下面,它定义子包功能简介(一句话说明)。格式为:

```
Summary : 子包简介
```

3. Group

此域必须在%package 下面,其定义子包所属软件类别(软件类别请参见精通 RPM 之查询篇。格式为:

```
Group : 软件类别
```

4. %description :

此描述段的内容是较为详细的子包功能介绍,介绍为文本形式,格式不作要求,可任意换行或分段。格式为:

```
%description 子包选项
...介绍子包功能的内容...
```

5. %files :

此文件段的内容是子包所要包含的文件列表。文件列表中,一个文件占用一行,还可使用多种文件修饰符。(详见精通 RPM 之制作篇(上))

段名应用格式为:

```
%files 子包选项 [-f 文件名]
```

注意: 上述%description 与%files 段所用的子包选项形式,必须与%package 所用的子包选项形式一致,否则的话,它们定义的不是同一个子 包,RPM 检查时将报错退出。如定义过%package name 后,描述段名须用%description name,文件

段名须用%files name 方可。而用%description -n name 则不行,%files -n name 也不行。

子软件包也可使用%pre,%post,%preun,%postun,%triggerin,%triggerun 和%triggerpostun 等七 个可选的功能段,因为它们都可使用子包选项。当使用子包选项时,它们的段内容就是用来管理子软件包的脚本程序。要注意的是,这些段使用的子包选项形式也必 须与%package 段使用的子包选项形式一致。

条件语句的使用

在软件包描述文件中,可以灵活地使用条件语句,位置不限制。这些语句,用于当前体系与操作系统的判断,当条件为真或为假时,RPM 均会引用其相应的描述内容。

条件语句有两种格式:

```
1. {%ifarch,%ifnarch,%ifos,%ifnos} 值 1 [值 2] ...
描述内容
%endif
```

注: {}所括内容必选其中之一,[]所括为可选项,各个值之间以空格分隔,%endif 表示条件语句结束。

此语句的含义是:
1) 使用%ifarch 时,表示如果当前体系为值 1 或值 2...,则引用描述内容。
2) 使用%ifnarch 时,表示如果当前体系不为值 1 或值 2...,则引用描述内容。
3) 使用%ifos 时,表示如果当前操作系统为值 1 或值 2...,则引用描述内容。
4) 使用%ifnos 时,表示如果当前操作系统不为值 1 或值 2...,则引用描述内容。

如果想在 LZE 包描述文件的文件段增加只适用于 sparc 体系的文件 /etc/sparc.lze 和 /etc/sparc.ime,则可在文件段内加入如下语句:

```
%ifarch sparc
/etc/sparc.lze
/etc/sparc.ime
%endif
```

这样做以后,如果当前体系为 sparc,则 RPM 在打包时会加入这两个文件。

```
2. {%ifarch,%ifnarch,%ifos,%ifnos} 值 1 [值 2] ...
描述内容 1
%else
描述内容 2
%endif
```

注: {}所括内容必选其中之一,[]所括为可选项,各个值之间以空格分隔,%else 表示另外一种情况,%endif 表示条件语句结束。

此语句的含义是:
1) 使用%ifarch 时,表示如果当前体系为值 1 或值 2...,则引用描述内容 1,否则引用描述 内容 2。
2) 使用%ifnarch 时,表示如果当前体系不为值 1 或值 2...,则引用描述内容 1,否则引用描述内容 2。
3) 使用%ifos 时,表示如果当前操作系统为值 1 或值 2...,则引用描述内容 1,否则引用描述内容 2。
4) 使用%ifnos 时,表示如果当前操作系统不为值 1 或值 2...,则引用描述内容 1,否则引用描述内容 2。

如果想根据当前操作系统来确定 LZE 包的名字,则可在描述文件头使用如下语句定义 Name 域:

```
%ifos linux
Name : lzeforlinux
%else

%ifos aix
Name : lzeforaix
%else
Name : lzeforothersys
%endif

%endif
```

本例中使用了嵌套的条件语句,它说明的情况是:如果操作系统为 linux,则软件名定为 lzeforlinux,如果操作系统为 aix,则软件名定为 lzeforaix,如果不是上述两个操作系统,则将软件名定为 lzeforothersys。

如何在描述文件中使用宏(macros)

1. 宏是什么?
学过 C 语言的人都知道,宏是用来实现文本替换的,即定义了宏名与宏体后,文件中所有有宏名的地方在预处理时将被宏体替换掉。使用宏可以减少文字的录入量,方便了编程人员。在软件包描述文件中使用宏,也是基于这个目的,只不过这个宏与 C 语言的宏定义格式不同而已。

2. 宏的定义

描述文件中宏的定义格式为:

```
%define <name>[(opts)] <body>
```

注:[]所括为可选项。<name>为宏名,宏名可用字母,数字和下划线(_),并且其长度最小为 3。opts 为一个或多个选项,各选项之间无分 隔,选项采用 getopt 函数要求的形式,即选项为单个字符,如果某个选项需要参数,则需要在这个选项后加个冒号(:)。<body>为宏 体,它周围的空字符将被删掉。宏体的内容须在一行上。

如没有选项的宏定义:

```
%define aaa "This is my software"
```

如仅有一个选项的宏定义:

```
%define xxx(p:Z) echo %{-p:%{-p*}} %{-Z}
```

3. 宏的使用

宏的使用格式为:

```
%<name> [opt1] [opt2]... [arg1] [arg2]...
```

或

```
%{<name>} [opt1] [opt2]... [arg1] [arg2]...
```

注: []所括为可选项;<name>为所应用的宏名,宏名可以用{}括住;opt1,opt2...为选项,均须以减号(-)开头,并且如果选项需要参数,则必须提供一个选项参数;arg1,arg2...则为宏的参数。

如上面定义的 xxx 宏,可这样使用:

```
%xxx -p zhsoft hello world<
```

例子中,xxx 宏使用一个选项-p,zhsoft 为-p 选项的参数,还有两个宏的参数 hello 和 world。

注意: 宏使用时最好多换一行(即宏下面空一行),因为宏在扩展后并不换行,这样如果不多换行,则下面一行若有内容的话,宏扩展后的内容将和下面一行的内容合并在 一起,极容易出现错误。这也是笔者发现 RPM 宏的问题之一。还有一个问题,如果注释行上存在宏,则这个宏也将扩展,错矣!因为注释本来就是要忽略掉的,有 宏也不必再扩展了。这两个问题都需要引起 RPM 开发者的注意,并切实加以解决。

4. 宏体专用的宏

宏体中可使用如下专用的宏:(类似 SHELL 形式的宏)

1) %0 : 表示所在宏的宏名;
2) %* : 表示宏的所有参数;
3) %# : 表示宏的参数个数;
4) %{-f} : 表示如果宏使用了-f 选项,则它表示-f 及其选项参数;
5) %{-f*} : 表示如果宏使用了-f 选项,则它表示-f 所带的参数;
6) %{-f:X} : 表示如果宏使用了-f 选项,则它表示 X;
7) %{!-f:Y} : 表示如果宏没有使用-f 选项,则它表示 Y;

%1,%2,... : 表示宏的参数 1,参数 2...

如,上例中 xxx 宏执行时,若宏体中有上述专用的宏,则专用宏将会扩展为:

1) %0 扩展为 xxx;
2) %*扩展为 hello world;
3) %#扩展为 2;
4) %{-p}扩展为-p zhsoft;
5) %{-p*}扩展为 zhsoft;
6) %{-p:good}扩展为 good;
7) %{!-Z:bad}扩展为 bad; (因为 xxx 宏未使用-Z 选项)

%1 为 hello,%2 为 world,没有其它参数。

5. 系统内置的宏

系统内置的宏可分如下三类:

5.1 定义类
1) %define ... : 定义一个宏;(原来,%define 也是一个宏啊)
2) %undefine ... : 取消一个宏;(宏取消后,此语句下面的描述文件就不能再使用这个宏了,即使使用,该宏也不会被扩展了)

5.2 调试类
1) %trace : 打印宏扩展前后的调试信息;
2) %dump : 打印活动的宏(宏名及宏体);
3) %{echo:...} : 打印...到标准错误设备;
4) %{warn:...} : 打印...到标准错误设备;
5) %{error:...} : 打印...到标准错误设备,并且返回 BADSPEC 值;

5.3 特殊类
这类宏的默认值通常放在/usr/lib/rpm/macros 文件中,用户通过编辑自己主目录(HOME)下的.rpmmacros 文件(~/.rpmmacros),可重定义这类宏,改变其默认值,以供 RPM 在软件包制作,安装及查询时使用自己的定义。

这类宏的定义格式为:

```
%<name> <body>
```

注: <name>为宏名,<body>为宏体。

1) %packager,%vendor,%distribution :

这三个宏用于定义描述文件中 Packager,Vendor,Distribution 三个可选域的默认的域值,即如果这三个域中有哪个未在描述文件中定义,且其相对应的宏有定义,则 RPM 会采用其对应的宏的宏体。

如我的~/.rpmmacros 文件中有这样三行:

```
%vendor 纵横软件制作中心
%packager 雨亦奇
%distribution 小赵'2001
```

这样,软件包描述文件中再也不用定义那三个域了,由此制作出来的软件包在查询时,其打包者(Packager),销售商(Vendor)及发行版(Distribution)均自动搞定了,一劳永逸。

2) %buildroot,%_provides :

这两个宏定义软件包建包时用的根目录及软件包所提供的功能。它们在打包时不会象 上面那三个宏一样主动被 RPM 采用,而是必须在描述文件中写那么几行。即:

```
%vendor 纵横软件制作中心
%packager 雨亦奇
%distribution 小赵'2001
Buildroot : %buildroot
Provides : %_provides
```

3) %_topdir,%_builddir,%_rpmdir,%_sourcedir,%_specdir,%_srcrpmdir :

这六个宏都是 RPM 制作软件包时要用的,它们在/usr/lib/rpm/macros 文件中的默认值为:

```
%_topdir %{_usrsrc}/dist
%_builddir %{_topdir}/BUILD
%_rpmdir %{_topdir}/RPMS
%_sourcedir %{_topdir}/SOURCES
```

```
%_specdir %{_topdir}/SPECS
%_srcrpmdir %{_topdir}/SRPMS
```

%_topdir 宏定义的是 RPM 制作软件包时所用目录的顶层目录,一般为 /usr/src/dist(%{_usrsrc} 宏的值为/usr/src)。在顶层目录下面,又有五个子目录:

- 编译连接源程序时用的目录,由%_builddir 宏定义,常用 BUILD;
- 生成的 RPM 执行程序包存放的目录,由%_rpmdir 宏定义,常用 RPMS;
- 软件源程序存放的目录,由%_sourcedir 宏定义,常用 SOURCES;
- 软件包描述文件存放的目录,由%_specdir 宏定义,常用 SPECS;
- 生成的 RPM 源程序包存放的目录,由%_srcrpmdir 宏定义,常用 SRPMS。

由于宏的递归特性,我们可以通过只定义%_topdir 宏来达到改变%_builddir 等五个宏的目的。(注 意:%_builddir 等五个宏的宏体如无特殊要求,尽量不要改变,它们是标准的定义,应该采用)这对于普通用户来说,意义非常重大。因为 RPM 默认的顶层目录/usr/src/dist 并不是每个用户都可以随便使用的,普通用户更想在自己所有的目录下用 RPM 来制作些软件包。我也有这种想 法,所以在 ~/.rpmmacros 文件里加上这么一行:

```
%_topdir /usr/zzz/rpm
```

同时,在此宏定义的目录下面建立了 RPM 所需的子目录,使用命令为:

```
$ cd /usr/zzz
$ mkdir -p rpm/{BUILD,RPMS/i386,SOURCES,SPECS,SRPMS}
$
```

命令中的 i386 是 RPM 默认的体系名,RPM 生成的执行程序包是存放在"RPMS/体系名"目录下面的。这么做以后,我就可以在自己的目录下制作 RPM 软件包了,象超级用户一样自由。

4)%_excludedocs,%_ftpport,%_ftpproxy,%_httpport,%_httpproxy,%_netsharepath :

这六个宏对 RPM 软件包的安装和查询起作用。

- %_excludedocs : 如果其值定义为 1,则 RPM 安装软件包时,对说明文档的默认作法是不安装;
- %_ftpport : 此宏用于定义 RPM 默认的 FTP 端口;
- %_ftpproxy : 此宏用于定义 RPM 默认的 FTP 代理服务器;
- %_httpport : 此宏用于定义 RPM 默认的 HTTP 端口;
- %_httpproxy : 此宏用于定义 RPM 默认的 HTTP 代理服务器;
- %_netsharepath : 此宏用于定义 RPM 默认的网络共享目录,适用于网络文件系统(NFS)。

6. 一种特殊的宏

这种宏的用法是:

```
%(SHELL 命令及其参数)
```

它的结果是取指定的 SHELL 命令的标准输出的结果作为描述文件内容的一部分。如软件包描述文件的某个部分需要加上当前日期,则可以用:

```
%(date +%Y-%m-%d)
```

执行后,该宏将扩展为类似 2001-10-31 的日期数据。用户不妨在自己的描述文件的预处理段(%prep)内加上这么两行试试:

```
%(date +%Y-%m-%d)
exit 1
```

注: exit 1 用于中止 RPM 的执行。

描述文件模板
以下所有描述文件模板均以 LZE 软件包制作为例,以源程序现场编译后产生的文件为准生成软件包。描述文件中一般只描述必要的部分。另外,如果文件段的所有 文件已存在于系统中,并且想直接利用打包,则可以去掉 Source 域,去掉 RPM 建包用功能段 (%prep,%build,%install,%clean)。

1. 只有父包,没有任何子包:
此描述文件见精通 RPM 之制作篇(上)。此文件中还可以去掉几个可选的功能段,如%pre,%post,%preun,%postun,%triggerin,%triggerun,%triggerpostun。这几个段在此文件 中无实质用途,执行时仅显示 RPM 开始执行某个脚本程序的信息。此描述文件仅生成软件包 lze-6.0-2.i386.rpm(父包)。

2. 有父包,也有子包:

描述文件如下:

```
1 # 文件名称: lze-6.0-2.spec1
2 # 文件功能: lze 软件包描述信息
3 # 文件作者: 纵横软件制作中心雨亦奇 国防大学研究生二队赵建利
4 # 修改时间: 2001.10.31
5
6 Name: lze
7 Version: 6.0
8 Release: 2
9 Summary: 小赵全屏幕中英文多窗口多功能编辑器(LINUX/UNIX 系统适用)
10 Group: Applications/Editors
```

```
11 License: Share
12 Source: http://zhsoft.myetang.com/lze-6.0-2.src.tgz
13
14 %description
15 小赵编辑器，是为使用SCO UNIX,LINUX多用户系统的广大用户专门设计的全屏幕多窗
16 口中英文多功能编辑器。
17 它主要有以下十大特点：1.全屏幕菜单操作。2.显示方式多样。3.块操作丰富。4.十
18 字制表功能强大。5.多窗口操作灵活自如。6.文件操作功能齐全。7.解释输出功能独具特
19 色。8.自带中文输入法（增强五笔和增强拼音），实用方便。9.十六进制编辑功能，如虎
20 添翼。10.即时翻译,按到即译。
21 总之,小赵编辑器会成为您在UNIX,LINUX系统上编制程序和书写一般性文稿的好帮手。
22 它将在工作中助您一臂之力，轻松上阵，游刃有余！
23
24 %prep
25 echo "预处理脚本程序(prep)开始执行"
26 %setup
27
28 %build
29 echo "编译连接脚本程序(build)开始执行"
30 make
31
32 %install
33 echo "安装脚本程序(install)开始执行"
34 make install
35
36 # 配置文件子包
37 %package config
38 summary : 小赵编辑器LZE的配置文件
39 group : Applications/Editors
40
41 %description config
42 小赵编辑器用配置文件包括功能键定义文件与
43 输入法控制文件,用户可根据实际情况加以修改。
44
45 %files config
46 %config /etc/funkey.def
47 %config /etc/inputme.def
48
49 # 说明文档子包
```

```
50 %package doc
51 summary : 小赵编辑器 LZE 的说明文档
52 group : Applications/Editors
53
54 %description doc
55 小赵编辑器说明文档,详细介绍了该编辑器的
56 命令行用法及内置的各项菜单的功能与操作,对用
57 户熟悉小赵编辑器有很大作用。
58
59 %files doc
60 %doc /usr/doc/lze-6.0/README
61 %doc /usr/doc/lze-6.0/LICENSE
62
63 # 父包文件段
64 %files
65 %defattr (-,root,root)
66 /usr/bin/lze
67 /usr/bin/lzeime.py
68 /usr/bin/lzeime.wb
69 /etc/wbzc.dat
70
```

此描述文件生成软件包有:lze-6.0-2.i386.rpm(父包),lze-config-6.0-2.i386.rpm(配置文件子包)和lze-doc-6.0-2.i386.rpm(说明文档子包)。

3. 没有父包,只有子包:

没有父包,意味着描述文件中可以没有父包的文件段(%files),请看下面的描述文件:

```
1 # 文件名称: lze-6.0-2.spec2
2 # 文件功能: lze 软件包描述信息
3 # 文件作者: 纵横软件制作中心雨亦奇 国防大学研究生二队赵建利
4 # 修改时间: 2001.10.31
5
6 Name: lze
7 Version: 6.0
8 Release: 2
9 Summary: 小赵全屏幕中英文多窗口多功能编辑器(LINUX/UNIX 系统适用)
10 Group: Applications/Editors
11 License: Share
12 Source: http://zhsoft.myetang.com/lze-6.0-2.src.tgz
13
14 %description
15 小赵编辑器，是为使用 SCO UNIX,LINUX 多用户系统的广大用户专门设计的全
```

```
屏幕多窗
16 口中英文多功能编辑器。
17 它主要有以下十大特点：1.全屏幕菜单操作。2.显示方式多样。3.块操作丰
富。4.十
18 字制表功能强大。5.多窗口操作灵活自如。6.文件操作功能齐全。7.解释输
出功能独具特
19 色。8.自带中文输入法（增强五笔和增强拼音），实用方便。9.十六进制编
辑功能，如虎
20 添翼。10.即时翻译,按到即译。
21 总之,小赵编辑器会成为您在 UNIX,LINUX 系统上编制程序和书写一般性文稿
的好帮手。
22 它将在工作中助您一臂之力，轻松上阵，游刃有余!
23
24 %prep
25 echo "预处理脚本程序(prep)开始执行"
26 %setup
27
28 %build
29 echo "编译连接脚本程序(build)开始执行"
30 make
31
32 %install
33 echo "安装脚本程序(install)开始执行"
34 make install
35
36 # 配置文件子包
37 %package config
38 summary : 小赵编辑器 LZE 的配置文件
39 group : Applications/Editors
40
41 %description config
42 小赵编辑器用配置文件包括功能键定义文件与
43 输入法控制文件,用户可根据实际情况加以修改。
44
45 %files config
46 %config /etc/funkey.def
47 %config /etc/inputme.def
48
49 # 说明文档子包
50 %package doc
51 summary : 小赵编辑器 LZE 的说明文档
52 group : Applications/Editors
53
54 %description doc
```

```
55 小赵编辑器说明文档,详细介绍了该编辑器的
56 命令行用法及内置的各项菜单的功能与操作,对用
57 户熟悉小赵编辑器有很大作用。
58
59 %files doc
60 %doc /usr/doc/lze-6.0/README
61 %doc /usr/doc/lze-6.0/LICENSE
62
63 # 执行程序子包
64 %package bin
65 summary : 小赵编辑器 LZE 的执行程序
66 group : Applications/Editors
67
68 %description bin
69 小赵编辑器执行程序为 lze,五笔输入法服务器执行程序
70 为 lzeime.wb,拼音输入法服务器执行程序为 lzeime.py。
71
72 %files bin
73 %defattr (-,root,root)
74 /usr/bin/lze
75 /usr/bin/lzeime.py
76 /usr/bin/lzeime.wb
77 /etc/wbzc.dat
78
```

此描述文件生成三个软件包:lze-config-6.0-2.i386.rpm(配置文件子包),lze-doc-6.0-2.i386.rpm(说明文档子包),lze-bin-6.0-2.i386.rpm(执行程序子包)。

精通 RPM 之制作篇（下）

要想制作 RPM 格式的软件包,需使用如下命令格式:

```
rpm-bX[制作选项 1 制作选项 2...]描述文件 1 描述文件 2...
```

注:-bX 可用-tX 替换,效果有所不同:使用-b 时,需要在命令行上指定软件包描述文件;而使用-t 时,在命令行上需要指定的不是软件包描述文件,而是 含有此描述文件的 TAR 格式的包裹文件(必须用 gzip 压缩),RPM 在准备制作软件包时,会自动从此包裹文件中提取描述文件,根据其中的建包指令,生成 RPM 格式的软件包。注意:此描述文件必须以.spec 为后缀,其中必须有 Source 域,并且此域定义的源程序包必须存放在当前目录下,否则的 话,RPM 将报错退出。
注意,-bX 与-tX 当中的 X,取不同的值时,RPM 将执行不同的操作:(下面各个例子

均对LZE的描述文件lze-6.0-2.spec进行操作,有的则通过管道技术,用nl命令给输出加上行号,以便解释)

1)X=p时,指示RPM执行描述文件中的预处理段(%prep)的脚本程序。该脚本程序一般用来解压源程序包,并且为源程序打补丁,升级源程序。

```
#rpm-bplze-6.0-2.spec2>&1|nl
1Executing:%prep
2+umask022
3+cd/usr/src/dist/BUILD
4+echo'预处理脚本程序(prep)开始执行'
5 预处理脚本程序(prep)开始执行
6+cd/usr/src/dist/BUILD
7+rm-rflze-6.0
8+/bin/gzip-dc/usr/src/dist/SOURCES/lze-6.0-2.src.tgz
9+tar-xvvf-
10drwxr-xr-xroot/root02001-11-0216:02lze-6.0/
11-rw-------root/root2462262001-11-0216:00lze-6.0/lze.c
12-rw-------root/root982492001-11-0216:00lze-6.0/lzeime.wb.c
13-rw-------root/root3399022001-11-0216:00lze-6.0/lzeime.py.c
14-rw-r--r--root/root12832001-11-0216:00lze-6.0/funkey.def
15-rwxr--r--root/root2502001-11-0216:00lze-6.0/inputme.def
16-rw-r--r--root/root8132742001-11-0216:00lze-6.0/wbzc.dat
17-rw-------root/root4742001-11-0216:02lze-6.0/makefile
18+STATUS=0
19+'['0-ne0']'
20+cdlze-6.0
21++/usr/bin/id-u
22+'['0=0']'
23+/bin/chown-Rhfroot.
24++/usr/bin/id-u
25+'['0=0']'
26+/bin/chgrp-Rhfroot.
27+/bin/chmod-Rfa+rX,g-w,o-w.
28+exit0
#
```

注:本例中第1行显示的"Executing:%prep"表明RPM开始执行预处理段的脚本程序。例中行号后那些以加号(+)开始的行均为预处理段脚本程序的指令,其它内容则为指令执行所输出的结果。从预处理段脚本程序所执行的指令,我们就可以看出RPM正在做什么:
第2行:设置文件创建掩码;
第3行:转到RPM默认的编译目录;

第 4 行:回显字符串,这才是用户所写的预处理段脚本程序的开始,第 2,3 行均是 RPM 增加的指令;
第 6-27 行:为%setup 所扩展出来的指令及其执行结果,其中可以看到 RPM 用 gzip 命令解压 LZE 的源程序包(第 8 行),而后用 tar 命令展开源程序包(第 9 行)。
第 28 行:正常退出,返回值为 0。

2)X=l 时,指示 RPM 检查文件段(%files),看其中的文件是否存在。如果不存在,则 RPM 会报错退出。

```
#rpm-bl-vvlze-6.0-2.spec|nl
1Processingfiles:lze
2D:File0:0100644root.root/etc/funkey.def
3D:File1:0100644root.root/etc/inputme.def
4D:File2:0100644root.root/usr/doc/lze-6.0-2/README
5Filenotfound:/usr/doc/lze-6.0-2/LICENSE
6D:File3:0100555root.root/usr/bin/lze
7D:File4:0100511root.root/usr/bin/lzeime.py
8D:File5:0100511root.root/usr/bin/lzeime.wb
9D:File6:0100644root.root/etc/wbzc.dat
10Provides:lze-edit
11PreReq:/bin/sh
12Requires:/bin/sh
#
```

注:本例命令中使用了通用选项-vv,以输出 RPM 检查时的调试信息(以行号和 D:开始的行)。
由上看出,RPM 对文件段的所有文件逐一进行检查,找到的文件显示其权限,属主和属组信息,结果发现/usr/doc/lze- 6.0-2.LICENSE 文件不存在,于是报出错误(File not found)(见第 5 行)。RPM 在检查过文件后,还显示出描述文件要求 LZE 提供的功能(Provides)和所需的功能(Requires)。

3)当 X=c 时,指示 RPM 依次执行预处理段(%prep),编译段(%build)的脚本程序。编译段的脚本程序用来编译连接软件的源程序,生成可执行 程序,通常一个 make 命令足够。因为一个有良好习惯的程序员会把 Makefile(程序维护文件)写好,以便其它程序员编译该软件。如果某个软件没有维 护文件的话,用户要么自己写个 Makefile,要么在编译段里写上软件编译与连接的各项命令。

4)当 X=i 时,指示 RPM 依次执行预处理段(%prep),编译段(%build)和安装段(%install)的脚本程序。安装段的脚本程序的任务是 将编译连接好的执行程序拷贝到适当的目录(如/bin,/usr/bin 等公共执行目录),以便打包或执行。它通常执行的指令是 make install。

5)当 X=b 时,指示 RPM 依次执行预处理段(%prep),编译段(%build),安装段

(%install)的脚本程序,之后根据文件段(%files)的文件列表,将文件打包,生成RPM 执行程序包,最后执行清理段(%clean)。

```
#rpm-bblze-6.0-2.spec2>&1|nl
1Executing:%prep
2+umask022
3+cd/usr/src/dist/BUILD
4+echo'预处理脚本程序(prep)开始执行'
5 预处理脚本程序(prep)开始执行
6+cd/usr/src/dist/BUILD
7+rm-rflze-6.0
8+tar-xvvf-
9+/bin/gzip-dc/usr/src/dist/SOURCES/lze-6.0-2.src.tgz
10drwxr-xr-xroot/root02001-11-0217:04lze-6.0/
11-rw-------root/root2462262001-11-0216:00lze-6.0/lze.c
12-rw-------root/root982492001-11-0216:00lze-6.0/lzeime.wb.c
13-rw-------root/root3399022001-11-0216:00lze-6.0/lzeime.py.c
14-rw-r--r--root/root12832001-11-0216:00lze-6.0/funkey.def
15-rwxr--r--root/root2502001-11-0216:00lze-6.0/inputme.def
16-rw-r--r--root/root8132742001-11-0216:00lze-6.0/wbzc.dat
17-rw-------root/root4742001-11-0216:02lze-6.0/makefile
18-rw-r--r--root/root12552001-11-0217:04lze-6.0/getinputme.c
19+STATUS=0
20+'['0-ne0']'
21+cdlze-6.0
22++/usr/bin/id-u
23+'['0=0']'
24+/bin/chown-Rhfroot.
25++/usr/bin/id-u
26+'['0=0']'
27+/bin/chgrp-Rhfroot.
28+/bin/chmod-Rfa+rX,g-w,o-w.
29+exit0
30Executing:%build
31+umask022
32+cd/usr/src/dist/BUILD
33+cdlze-6.0
34+echo'编译连接脚本程序(build)开始执行'
35 编译连接脚本程序(build)开始执行
36+make
37cc-fwritable-strings-DUSE_AS_LZE-DFOR_LINUX-s-I/usr/zzz/src/include-DFOR_LZE_IN
38bsrc/mycurses.clze.cgetinputme.c/usr/zzz/src/my.a
39cc-DFOR_LINUX-s-I/usr/zzz/src/include-olzeime.wbgetinputme.clzeime.wb.c/usr/zzz/
40lzeime.wb.c:Infunction`do_service':
```

```
41lzeime.wb.c:1409:warning:passingarg5of`bsearch'fromincompatiblepointertype
42cc-DFOR_LINUX-s-I/usr/zzz/src/include-olzeime.pygetinputme.clzeime.py.c/usr/zzz/
43+exit0
44Executing:%install
45+umask022
46+cd/usr/src/dist/BUILD
47+cdlze-6.0
48+echo'安装脚本程序(install)开始执行'
49 安装脚本程序(install)开始执行
50+makeinstall
51installing...
52done
53+exit0
54Processingfiles:lze
55FindingProvides:(using/usr/lib/rpm/find-provides)...
56FindingRequires:(using/usr/lib/rpm/find-requires)...
57Provides:lze-edit
58PreReq:/bin/sh
59Requires:/bin/shld-linux.so.2libc.so.6libc.so.6(GLIBC_2.0)libc.so.6(GLIBC_2.1)
60Wrote:/usr/src/dist/RPMS/i386/lze-6.0-2.i386.rpm
61Executing:%clean
62+umask022
63+cd/usr/src/dist/BUILD
64+cdlze-6.0
65+echo'建包结束后清理脚本程序(clean)开始执行'
66 建包结束后清理脚本程序(clean)开始执行
67+exit0
#
```

注:本例中,各行解释如下:
第 1 行:显示 RPM 开始执行预处理段(%prep);
第 2-29 行:为预处理段(%prep)脚本程序执行的命令与结果,命令前有加号(+),结果前面则没有。预处理完成了 LZE 源程序包的解压缩(gzip-dc)与包的展开(tar-xvvf),从而为源程序的编译打下了基础;
第 30 行:RPM 表示开始执行编译段(%build);
第 31-43 行:为编译段(%build)脚本程序执行的命令与结果。脚本程序执行的 make 维护命令,该命令执行了 cc 编译程序,以编译与连接 LZE 软件;
第 44 行:RPM 表示开始执行安装段(%install);
第 45-53 行:为安装段(%install)脚本程序执行的命令与结果。本段程序执行了 makeinstall 命令,将 LZE 的执行程序等文件拷贝到系统目录下;
第 54 行:RPM 显示:开始处理 LZE 的文件(为%files 文件段的内容);
第 55-59 行:RPM 利用/usr/lib/rpm/find-provides 程序查找 LZE 提供的功能,又利用/usr/lib/rpm/find-requires 查找 LZE 所依赖的功能,以设定 LZE 的依赖

关系;
第 60 行:制作完成 LZE 执行程序包,文件名为 lze-6.0-2.i386.rpm,在 /usr/src/dist/RPMS/i386 目录下;
第 61 行:RPM 显示:开始执行清理段(%clean)脚本程序,本段程序用于清理临时文件;
第 62-67 行:为清理段(%clean)脚本程序执行的命令与结果。

从上我们可以清楚看出 RPM 制作软件包的工作流程:预处理段,编译段,安装段,软件包制作,清理段。

6)当 X=s 时,指示 RPM 建立源码包。RPM 源码包的内容包括软件包描述文件(SPEC),软件源程序,软件补丁程序,图标文件等几项。建立源码包不需要执行软件包描述文件中的各个功能段,仅需将所需文件包含到包中即可。

```
#rpm-bslze-6.0-2.spec
Wrote:/usr/src/dist/SRPMS/lze-6.0-2.src.rpm
#rpm-qplv/usr/src/dist/SRPMS/lze-6.0-2.src.rpm
-rw-------rootroot206711 月 215:44lze-6.0-2.spec
-rw-r--r--rootroot53870611 月 217:05lze-6.0-2.src.tgz
#
```

注:本例中,使用 rpm-bs 命令生成了 LZE 源码包 lze-6.0-2.src.rpm(在 RPM 标准源码目录/usr/src/dist/SRPMS 下),然后用 rpm -qplv 命令查询源码包中所含的文件信息,从中可以看到 LZE 源码包中有两个文件:一个是软件包描述文件 lze-6.0-2.spec,一个是 LZE 源 代码包 lze-6.0-2.src.tgz(TAR 打包再用 gzip 压缩),此文件由描述文件中的 Source 域确定。

7)当 X=a 时,指示 RPM 依次执行预处理段(%prep),编译段(%build),安装段(%install)脚本程序,之后先生成 RPM 源码包,再 根据文件段(%files)的文件列表,将文件打包,生成 RPM 执行程序包,最后执行清理段(%clean)脚本程序,清除中间文件。此命令执行的结果相 当于先执行 rpm-bs 命令生成源码包,再执行 rpm-bb 命令生成执行码包。

选项列表

选项详解

通用选项的解释见精通 RPM 之安装篇,本文不再赘述。

1.--short-circuit:单步执行
此选项的目的在于单步执行功能段,仅适用命令为 RPM-bc(或-tc)和-bi(或-ti)

时。当用 rpm-bc--short-circuit 命令 时,RPM 将不再执行预处理段(%prep)的脚本程序,直接执行编译段(%build)的脚本程序。当用 rpm-bi--short-circuit 命 令时,RPM 将不再执行预处理段(%prep)和编译段(%build)的脚本程序,仅执行安装段(%install)的脚本程序。

为什么要使用这个选项?其原因可能是源程序包中的文件有问题,导致 RPM 在编译或安装过程中出现了这样或那样的错误,如不能单步执行,则无法排除这些错误。单步执行后,用户可以进入源程序所在的目录,查看源程序,修改错误,然后再用 tar 命令生成一个正确的源程序包,覆盖掉原包即可。这样,RPM 再制作 时就不会有问题了。

2.--timecheck:设置时间检查值
该选项的用法是:--timecheck<secs>
注:<secs>为检查的时间段,单位是秒,如--timecheck600 设置检查的时间段为 600 秒,即 10 分钟。

设置时间检查值的目的,在于检验打包文件是否是指定时间段内产生的新文件,如果不是,则 RPM 会产生警告信息,提醒用户某个文件不是新文件,而是老文件,可能是某种错误导致的。这种错误,可能是由于 makefile(维护文件)或描述文件中的安装段脚本程序书写不正确,从而使某个程序不能正确安装到指定目 录,因而 RPM 打包时总是引用老的程序文件。如下面的例子:

```
#rpm-bl--timecheck600lze-6.0-2.spec
Processingfiles:lze
warning:TIMECHECKfailure:/usr/bin/lze
FindingProvides:(using/usr/lib/rpm/find-provides)...
FindingRequires:(using/usr/lib/rpm/find-requires)...
Provides:lze-edit
PreReq:/bin/sh
Requires:/bin/shld-linux.so.2libc.so.6libc.so.6(GLIBC_2.0)libc.so.6(GLIBC_2.1)
#
```

注:本例中利用 RPM 检查文件列表,看哪个文件是在 10 分钟以前产生的老文件,结果 RPM 发出警告信 息:"warning:TIMECHECKfailure:/usr/bin/lze",这说明文件/usr/bin/lze 时间检查出现错误,这是个老 文件。经查,是 makefile 的问题,里面少了一句"cplze/usr/bin"。修正后,再次编译连接程序,再进行时间检查就没有这个问题了。

3.--buildroot:设定建包用根目录
该选项的用法是:

```
--buildroot<dir>
```

注:<dir>为用户指定的建包用的根目录。
此选项相当于在软件包描述文件的文件头加上一行:

```
Buildroot:<dir>
```

只不过在命令行上设定比较自由罢了。

通过设定建包用根目录,一般用户也可以建立那些只有超级用户(root)才能建立的 RPM 包了。下面以普通用户 zzz 建立 LZE 软件包为例,说明一下这种形式的包的建立过程。

1)在用户 zzz 的 HOME 目录(/usr/zzz)下,建立 RPM 建包所用的目录:

```
$cd/usr/zzz
$mkdir-prpm/{BUILD,RPMS/i386,SOURCES,SPECS,SRPMS}
$
```

2)拷贝描述文件 lze-6.0-2.spec 到 rpm/SPECS 目录,拷贝源程序包 lze-6.0-2.src.tgz 到 rpm/SOURCES 目录。

```
$cd/usr/zzz
$cp/root/lze-6.0-2.specrpm/SPECS
$cp/root/lze-6.0-2.src.tgzrpm/SOURCES
$
```

3)建立.rpmmacros 文件,编辑它:

```
$cd/usr/zzz
$vi.rpmmacros
```

在此文件中增加一行:

```
%_topdir/usr/zzz/rpm
```

用于确定 RPM 建包用的顶层目录为/usr/zzz/rpm。

4)转到描述文件目录,编辑修改 lze-6.0-2.spec 安装段的脚本程序:

```
$cd/usr/zzz/rpm/SPECS
```

```
$vilze-6.0-2.spec
$
```

用于确定 RPM 建包用的顶层目录为/usr/zzz/rpm。

安装段脚本程序修改为:(原 make 命令被注释掉)

```
%install
echo"安装脚本程序(install)开始执行"
#makeinstall
mkdir-p$RPM_BUILD_ROOT/usr/bin
mkdir-p$RPM_BUILD_ROOT/etc
mkdir-p$RPM_BUILD_ROOT/usr/doc/lze-6.0
cplzelzeime.wblzeime.py$RPM_BUILD_ROOT/usr/bin
cpinputme.deffunkey.defwbzc.dat$RPM_BUILD_ROOT/etc
cp/usr/doc/lze-6.0/*$RPM_BUILD_ROOT/usr/doc/lze-6.0
```

注:脚本程序中,使用建包用根目录的环境变量 RPM_BUILD_ROOT,相继在建包用根目录下创建若干子目录(usr/bin,etc,usr /doc/lze-6.0),然后拷贝程序或文件到相应的子目录中,从而完成完装。 5)选用--buildroot 选项,执行 RPM 建包命令:

```
$cd/usr/zzz/rpm/SPECS
$rpm-bb--buildroot/usr/zzz/tmplze-6.0-2.spec
Executing:%prep
+umask022
+cd/usr/zzz/rpm/BUILD
+echo'预处理脚本程序(prep)开始执行'
预处理脚本程序(prep)开始执行
+cd/usr/zzz/rpm/BUILD
+rm-rflze-6.0
+/bin/gzip-dc/usr/zzz/rpm/SOURCES/lze-6.0-2.src.tgz
+tar-xvvf-
drwxr-xr-xroot/root02001-11-0217:04lze-6.0/
-rw-------root/root2462262001-11-0216:00lze-6.0/lze.c
-rw-------root/root982492001-11-0216:00lze-6.0/lzeime.wb.c
-rw-------root/root3399022001-11-0216:00lze-6.0/lzeime.py.c
-rw-r--r--root/root12832001-11-0216:00lze-6.0/funkey.def
-rwxr--r--root/root2502001-11-0216:00lze-6.0/inputme.def
-rw-r--r--root/root8132742001-11-0216:00lze-6.0/wbzc.dat
-rw-------root/root4742001-11-0216:02lze-6.0/makefile
```

```
-rw-r--r--root/root12552001-11-0217:04lze-6.0/getinputme.c
+STATUS=0
+'['0-ne0']'
+cdlze-6.0
++/usr/bin/id-u
+'['500=0']'
++/usr/bin/id-u
+'['500=0']'
+/bin/chmod-Rfa+rX,g-w,o-w.
+exit0
Executing:%build
+umask022
+cd/usr/zzz/rpm/BUILD
+cdlze-6.0
+echo'编译连接脚本程序(build)开始执行'
编译连接脚本程序(build)开始执行
+make
cc-fwritable-strings-DUSE_AS_LZE-DFOR_LINUX-s-I/usr/zzz/src/include-DFOR_LZE_INPU
cc-DFOR_LINUX-s-I/usr/zzz/src/include-olzeime.wbgetinputme.clzeime.wb.c/usr/zzz/s
lzeime.wb.c:Infunction`do_service':
lzeime.wb.c:1409:warning:passingarg5of`bsearch'fromincompatiblepointertype
cc-DFOR_LINUX-s-I/usr/zzz/src/include-olzeime.pygetinputme.clzeime.py.c/usr/zzz/s
+exit0
Executing:%install
+umask022
+cd/usr/zzz/rpm/BUILD
+cdlze-6.0
+echo'安装脚本程序(install)开始执行'
安装脚本程序(install)开始执行
+mkdir-p/usr/zzz/tmp/usr/bin
+mkdir-p/usr/zzz/tmp/etc
+mkdir-p/usr/zzz/tmp/usr/doc/lze-6.0
+cplzelzeime.wblzeime.py/usr/zzz/tmp/usr/bin
+cpinputme.deffunkey.defwbzc.dat/usr/zzz/tmp/etc
+cp/usr/doc/lze-6.0/LICENSE/usr/doc/lze-6.0/README/usr/zzz/tmp/usr/doc/lze-6.0
+exit0
Processingfiles:lze
FindingProvides:(using/usr/lib/rpm/find-provides)...
FindingRequires:(using/usr/lib/rpm/find-requires)...
Provides:lze-edit
PreReq:/bin/sh
Requires:/bin/shld-linux.so.2libc.so.6libc.so.6(GLIBC_2.0)libc.so.6(GLIBC_2.1)
Wrote:/usr/zzz/rpm/RPMS/i386/lze-6.0-2.i386.rpm
Executing:%clean
```

```
+umask022
+cd/usr/zzz/rpm/BUILD
+cdlze-6.0
+echo'建包结束后清理脚本程序(clean)开始执行'
建包结束后清理脚本程序(clean)开始执行
+exit0
$
```

注:RPM 建包成功后,生成了/usr/zzz/rpm/RPMS/i386/lze-6.0-2.i386.rpm 包裹文件,
通过 RPM 的查询命令,我们就可以知道该包与由超级用户建成的包内容毫无二致:

```
$rpm-qplv/usr/zzz/rpm/RPMS/i386/lze-6.0-2.i386.rpm
-rw-r--r--rootroot128311 月 609:24/etc/funkey.def
-rwxr-xr-xrootroot25011 月 609:24/etc/inputme.def
-rw-r--r--rootroot81327411 月 609:24/etc/wbzc.dat
-rwxr-xr-xrootroot40863211 月 609:24/usr/bin/lze
-rwxr-xr-xrootroot8292011 月 609:24/usr/bin/lzeime.py
-rwxr-xr-xrootroot3856811 月 609:24/usr/bin/lzeime.wb
-rw-r--r--rootroot121511 月 609:24/usr/doc/lze-6.0/LICENSE
-rw-r--r--rootroot369011 月 609:24/usr/doc/lze-6.0/README
$
```

现在我们可以说,普通用户也可以自由地建立各种形式的 RPM 软件包了!

4.--target:设定目标平台
该选项的用法为:--target 体系-平台-操作系统
注:RPM 制作出来的软件包默认的体系为 i386,平台为 pc,操作系统为 linux。如果用户想加以改变,就需要使用此选项来确定一下,如下例:

```
#rpm-bb--targeti686-pc-solaris--quietlze-6.0-2.spec
创建目标平台:i686-pc-solaris
正在创建目标:i686-pc-solaris
#
```

本例设定生成的 RPM 包适用的目标平台为:i686-pc-solaris(即体系为 i686,平台为 pc,操作系统为 solaris)。注意:必须在 /usr/src/dist/RPMS 目录下创建一个 i686 的子目录,没有指定体系的子目录,RPM 将无法生成软件包。我们可以查询一下生成的软件包, 看其适用的体系与操作系统是什么:

```
#rpm-qp--qf"archis%{arch}\nosis%{os}\n"/usr/src/dist/RPMS/i686/lze-6.0-2.i686.rpm
```

```
archisi686
osissolaris
#
```

从输出上可以看到,其体系与操作系统正是我们所设定的。

5.--quiet:尽量减少信息输出
此选项的目的,是让 RPM 减少信息的输出。使用此选项后,如果没有错误发生,RPM 就不会输出多余的信息,这时的 RPM 也显得比较"安静"(quiet)了。

```
$rpm-bl--quietlze-6.0-2.spec
$
```

6.--clean:执行文件清理
如果软件包描述文件的清理段(%clean)没有删除临时文件的命令,那么 RPM 建包结束后那些临时文件还是存在的,占用了一定的空间。如果想让 RPM 自动删除那些临时文件,可以在建包时使用--clean 选项。此选项执行一条命令,即:
rm-rf 软件名-版本号

用它来删除"软件名-版本号"目录及该目录下的所有文件。这个"软件名-版本号"目录,也即 RPM 默认的存放解压后的源程序的目录。

```
#rpm-bl--cleanlze-6.0-2.spec
Processingfiles:lze
FindingProvides:(using/usr/lib/rpm/find-provides)...
FindingRequires:(using/usr/lib/rpm/find-requires)...
Provides:lze-edit
PreReq:/bin/sh
Requires:/bin/shld-linux.so.2libc.so.6libc.so.6(GLIBC_2.0)libc.so.6(GLIBC_2.1)
Executing:--clean
+umask022
+cd/usr/src/dist/BUILD
+rm-rflze-6.0
+exit0
```

注:例中的"Executing:--clean"表示 RPM 开始执行自己的文件清理操作。

7.--rmsource:删除源程序及描述文件

此选项用于指示 RPM 在建包后删除软件源程序和包描述文件,软件源程序由包描述文件中的 Source 域定义。此选项也可单独使用,如:

```
#rpm--rmsourcelze-6.0-2.spec
#lslze-6.0-2.spec../SOURCES/lze-6.0-2.src.tgz
ls:lze-6.0-2.spec:文件或目录不存在
ls:../SOURCES/lze-6.0-2.src.tgz:文件或目录不存在
#
```

8.--sign:软件包内置数字签名
此选项用于在软件包内添加 PGP 数字签名,通过数字签名,用户可以校验软件包是否原装,是否修改过。要使用 PGP 数字签名,必须先安装好 PGP 应用程序及产生自己的密钥对。(有关数字签名的内容,详见精通 RPM 之签名篇

使用此选项建包时,RPM 会要求输入密码,验证正确后开始执行建包的系列操作,在建包前会产生数字签名,以便写入包中。请看下例:(省略不少输出,.....表示)

```
#rpm-bb--signlze-6.0-2.spec
Enterpassphrase:mypass
Passphraseisgood.
Executing:%prep
......
Executing:%build
......
Executing:%install
......
Processingfiles:lze
......
Requires:/bin/shld-linux.so.2libc.so.6
Generatingsignature:1002
GeneratingsignatureusingPGP.
PrettyGoodPrivacy(tm)Version6.5.8
(c)1999NetworkAssociatesInc.
UsestheRSAREF(tm)Toolkit,whichiscopyrightRSADataSecurity,Inc.
ExportofthissoftwaremayberestrictedbytheU.S.government.
Wrote:/root/test/RPMS/i386/lze-6.0-2.i386.rpm
Executing:%clean
......
#
```

注:例中要求输入密码,输入 mypass 后校验成功,之后 RPM 开始建包操作。(实际上,mypass 在屏幕上并不显示出来)例中的 Generatingsignature 之下的几行就是产生数字签名时输出的东西。

其它建包相关的命令

其它建包相关的命令有两个,它们都是针对 RPM 格式的源码包进行操作的,执行时先安装源码包,然后再根据软件包描述文件进行下一步的工作。 1.重编译命令:
用法为:rpm--recompileRPM 源码包 1RPM 源码包 2...
如:

```
rpm--recompilelze-6.0-2.src.rpm
```

此命令执行时,首先安装此源码包,然后依次执行源码包内软件包描述文件的预处理段(%prep),编译段(%build),安装段(%install)的脚本程序。此命令相当于以下两条命令:

```
1)rpm-ilze-6.0-2.src.rpm
2)rpm-bilze-6.0-2.spec
```

2.重建立命令:
用法为:rpm--rebuildRPM 源码包 1RPM 源码包 2...
如:

```
rpm--rebuildlze-6.0-2.src.rpm
```

重编译命令执行后并不建立新的 RPM 软件包,而此重建立命令执行后则会制作出一个新的 RPM 软件包。重建立命令执行时,首先安装 RPM 源码包,然后依次执 行源码包内软件包描述文件的预处理段(%prep),编译段(%build),安装段(%install),清理段(%clean)的脚本程序,生成一个 新的 RPM 执行包,最后删除源程序包,描述文件及其它临时文件。此命令相当于以下两条命令:

```
1)rpm-ilze-6.0-2.src.rpm
2)rpm-bb--clean--resourcelze-6.0-2.spec
```

精通 RPM 之签名篇

数字签名(Digital Signature),是一种身份认证技术。软件包增加数字签名后,其它用户可以通过校验其签名,辨其真伪,看其是否原装,是否被修改过。RPM 采用的数 字签名为 PGP 数字签名,PGP 即 Pretty Good Privacy,它是采用 RSA 密码技术实现的,应用时要产生一个密钥对(两个密钥),一个为公开密钥(可告诉他人),一个为秘密密钥(自己保留)。秘密 密钥加密的文件任何有相应公开密钥的人均可解密,而用公开密钥加密的文件只有持有秘密密钥的人才可以解密。

PGP 的下载与安装

PGP 应用程序我是从 http://www.arges.tempo.at/pgp.download/main.html 网址处下载的。LINUX 版的下载文件为 PGPcmdln_6.5.8.Lnx_FW.rpm.tar,可用下面的命令解压此软件包,并安装 PGP:

```
# tar xvzf PGPcmdln_6.5.8.Lnx_FW.rpm.tar
-rwxr-xr-x root/other 2955703 2000-08-31 05:29 PGPcmdln_6.5.8_Lnx_FW.rpm
-rwxr-xr-x root/other 66 2000-08-31 10:09 PGPcmdln_6.5.8_Lnx_FW.rpm.sig
-rwxr-xr-x root/other10981 2000-08-30 21:58 WhatsNew.htm
-rwxr-xr-x root/other 8758 2000-08-30 21:58 WhatsNew.txt
#
# rpm -iv PGPcmdln_6.5.8_Lnx_FW.rpm
pgp-6.5.8-rsaref658
#
# rpm -ql pgp
/usr/bin/pgp
/usr/doc/pgp-6.5.8
/usr/doc/pgp-6.5.8/CREDITS
/usr/doc/pgp-6.5.8/IntrotoCrypto.pdf
/usr/doc/pgp-6.5.8/IntrotoCrypto.ps
/usr/doc/pgp-6.5.8/LICENSE
/usr/doc/pgp-6.5.8/PGPCmdLineGuide.pdf
/usr/doc/pgp-6.5.8/PGPCmdLineGuide.ps
/usr/doc/pgp-6.5.8/PGPCmdLineInstallGuide.pdf
/usr/doc/pgp-6.5.8/PGPCmdLineInstallGuide.ps
/usr/doc/pgp-6.5.8/RSALICENSE
/usr/doc/pgp-6.5.8/SampleKeys.asc
/usr/doc/pgp-6.5.8/WHATSNEW
/usr/doc/pgp-6.5.8/WhatsNew.htm
/usr/man/man1/pgp.1
#
```

注:第 1 条命令解压软件包,第 2 条命令安装 RPM 格式的 PGP 包,第 3 条命令则是查询一下 PGP 包里的文件,其执行文件是/usr/bin /pgp,说明文档在/usr/doc/pgp-6.5.8 目录下,有 man 手册(/usr/man/man1/pgp.1),可以用 man pgp 来获得 pgp 的帮助信息。

另外,还可以到网站 http://www.pgpi.com 查找或下载 PGP 程序,前面介绍的网站速度快,这个网站速度慢。

RPM 应用 PGP 所需的配置

1. 生成 PGP 密钥对

这需要使用 pgp -kg 命令来产生新的密钥对,以用于签名:

```
# pgp -kg
Pretty Good Privacy(tm) Version 6.5.8
(c) 1999 Network Associates Inc.
Uses the RSAREF(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.
Choose the public-key algorithm to use with your new key
1) DSS/DH (a.k.a. DSA/ElGamal) (default)
2) RSA
Choose 1 or 2: 1
Choose the type of key you want to generate
1) Generate a new signing key (default)
2) Generate an encryption key for an existing signing key
Choose 1 or 2: 1
Pick your DSS ``master key'' size:
1) 1024 bits- Maximum size (Recommended)
Choose 1 or enter desired number of bits: 1
Generating a 1024-bit DSS key.
You need a user ID for your public key. The desired form for this
user ID is your name, followed by your E-mail address enclosed in
<angle brackets>, if you have an E-mail address.
For example: John Q. Smith <jqsmith@nai.com>
Enter a user ID for your public key: zhsoft
Enter the validity period of your signing key in days from 0 - 10950
0 is forever (the default is 0):
You need a pass phrase to protect your DSS secret key.
Your pass phrase can be any sentence or phrase and may have many
words, spaces, punctuation, or any other printable characters.
Enter pass phrase: MYPASS
Enter same pass phrase again: MYPASS
PGP will generate a signing key. Do you also require an
encryption key? (Y/n) Y
Pick your DH key size:
1) 1024 bits- High commercial grade, secure for many years
2) 2048 bits- "Military" grade, secure for forseeable future
3) 3072 bits- Archival grade, slow, highest security
Choose 1, 2, 3, or enter desired number of bits: 1
Enter the validity period of your encryption key in days from 0 - 10950
0 is forever (the default is 0): 0
Note that key generation is a lengthy process.
PGP needs to generate some random data. This is done by measuring
the time intervals between your keystrokes. Please enter some
random text on your keyboard until the indicator reaches 100%.
Press ^D to cancel
100% of required data
```

```
Enough, thank you.
.******* ......................******* .
Make this the default signing key? (Y/n) Y
........******* ........................*******
Key generation completed.
#
```

注:例子中,在生成 PGP 密钥前有若干工作要做:
1) 选择哪种公开加密算法:第 1 个是 DSS/DH 算法(默认),第 2 个是 RSA 算法,本例用第 1 个。
2) 选择生成哪种类型的密钥:第 1 个是新的签名密钥(默认),第 2 个是为已存在的签名密
钥产生一个加密密钥,本例选第 1 个。
3) 选择 DSS 主密钥大小:最大为 1024 位(系统建议使用),本例选择用它,当然,也可以输
入其它位数。
4) 为公开密钥设置一个用户标识(user ID,格式为:用户名 <电子邮箱>),本例使用:
纵横软件制作中心 <zhsoft@371.net>。
5) 设定密钥适用天数(0-10950),本例选择 0,表示永远适用。
6) 为 DSS 秘密密钥设置一个密码,本例密码为 MYPASS,按要求输入了两次。
7) PGP 提示是否还需要为签名密钥加密,本例选择 Y(Yes)。

选择 DH 密钥大小:第 1 是 1024 位,第 2 是 2048 位,第 3 是 3072 位,还可输入其它数值,本例
选择 1,确下 DH 密钥大小为 1024 位。
9) 输入加密密钥适用天数(0-10950),本例选择 0,表示永远适用。
10) PGP 提示需要产生一些随机数据,用户需要随机输入一些,直到所需数据百分之百(100%)
完成才可。
11) PGP 生成签名密钥后,提示是否将此密钥设置为默认的密钥,本例选择 Y,将此密钥设
为默认使用的密钥。
12) PGP 密钥产生完成。
密钥生成后,PGP 会在用户主目录下建立一个.pgp 的子目录,用于存放密钥相关的文件。
下例列出了 root 用户.pgp 子目录下的文件:

```
# ls ~/.pgp
total 32
-rw------- 1 root root 2117 Nov 8 17:10 PGPMacBinaryMappings.txt
-rw------- 1 root root 146 Nov 8 17:10 PGPgroup.pgr
-rw------- 1 root root 151 Nov 8 17:12 PGPsdkPreferences
-rw------- 1 root root0 Nov 8 17:10 pgp.cfg
```

```
-rw------- 1 root root0 Nov 8 17:10 pubring-bak-1.pkr
-rw------- 1 root root 897 Nov 8 17:12 pubring-bak-2.pkr
-rw------- 1 root root 897 Nov 8 17:12 pubring.pkr
-rw------- 1 root root 512 Nov 8 19:34 randseed.rnd
-rw------- 1 root root0 Nov 8 17:10 secring-bak-1.skr
-rw------- 1 root root 984 Nov 8 17:12 secring-bak-2.skr
-rw------- 1 root root 984 Nov 8 17:12 secring.skr
#
```

文件之中:pubring.pkr 为公开密钥文件,secring.skr 为秘密密钥文件。

2. 配置 RPM 宏

RPM 要想使用 PGP 数字签名的功能,必须在/usr/lib/rpm/macros 宏文件或者在用户主目
录下的~/.rpmmacros 文件中,设置以下几个宏:(最好编辑自己的宏文件
--~/.rpmmacros 文件,
不要随意改变含 RPM 默认设置的宏文件/usr/lib/rpm/macros)
1) _signature :
此宏定义数字签名的类型,此类型只有一个,为 pgp(RPM 仅支持这一种)。其定义为:
%_signature pgp
2) _pgpbin :
此宏定义 PGP 执行程序名。其定义为:
%_pgpbin /usr/bin/pgp
3) _pgp_name :
此宏定义使用哪个 PGP 用户的公开密钥进行签名处理(PGP 可建立属于多个用户的密钥对)。
其定义格式为:
%_pgp_name 名字 <电子邮箱>
如:
%_pgp_name 纵横软件制作中心 <zhsoft@371.net>
4) _pgp_path :
此宏定义 RPM 使用的签名所在的目录,如:
%_pgp_path /root/.pgp
该宏定义 RPM 使用/root/.pgp 目录下的签名。
RPM 的 PGP 签名选项
1. --resign :
本选项用于为 RPM 软件包重新签名。如果原包没有数字签名,则为其添加签名;如果已有签名,则旧的签名将统统删除,之后再添加新的签名。其用法为:
rpm --resign 包裹文件 1 [包裹文件 2]...
2. --addsign :
本选项用于为 RPM 软件包添加数字签名(一个软件包可以有多个数字签名)。用法为:

rpm --addsign 包裹文件 1 [包裹文件 2]...
如:

```
# rpm --addsign lze-6.0-2.i386.rpm
Enter pass phrase: MYPASS
Pass phrase is good.
lze-6.0-2.i386.rpm:
Pretty Good Privacy(tm) Version 6.5.8
(c) 1999 Network Associates Inc.
Uses the RSAREF(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.
#
```

本例为 lze 软件包添加数字签名,输入密码为 MYPASS。数字签名也可以在建包时添加,这时需使用--sign 选项。

3. --checksig :

本选项用于校验 RPM 包的数字签名等内容,看其是否正常。用法为:
rpm --checksig [--nopgp] [--nogpg] [--nomd5] [--rcfile 资源文件] 包裹文件 1 [包裹文件 2]...
注:[]所括为可选项。--checksig 也可用-K 替换,效果相同。可选项中,--nopgp 选项指示 RPM 不校验 PGP 签名,--nogpg 选项指 示 RPM 不校验 GPG 签名,--nomd5 选项指示 RPM 不校验 MD5 检查和,--rcfile 选项则用于指定 RPM 所利用的资源配置文件。
如:

```
# rpm --checksig lze-6.0-2.i386.rpm
Pretty Good Privacy(tm) Version 6.5.8
(c) 1999 Network Associates Inc.
Uses the RSAREF(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.
lze-6.0-2.i386.rpm: pgp md5 OK
#
```

注:本例校验 lze 包的签名时,RPM 显示 pgp 校验 OK,md5 校验 OK,这表明 lze 包一切正常。

精通 RPM 之杂项篇

本系列的全部文章均来源于网上,原作者为雨亦奇和赵建利,这里只作整理之用。

系列的索引在此，由于文章内容较多，故使用 more 标签输出，如有不便，还请见谅。

一、数据库维护选项

1. --initdb :

本选项用于创建并初始化 RPM 数据库。如果默认目录下(/var/lib/rpm)已存在 RPM 数据库,则原有数据库予以保存,RPM 并不破坏,以防万一。如果没有数据库文件,则 RPM 创建并初始化这些文件。

例 1. 本例先列出 RPM 默认的数据库目录下的文件,然后运行初始化命令(initdb),最后再列一下同一目录下的文件,用户可以比较一下输出结果。

注: 通过比较可以看出,RPM 对已有的数据库并未进行初始化,数据库文件没有变化。

例 2. 本例通过创建一个新的目录并把它作为 RPM 数据库的目录,进而新建 RPM 数据库的所有文件。

注: 用 mkdir 建立目录时采用-p 选项表明,如果父目录不存在则建立父目录。RPM 创建数据库时使用--root 选项,将/usr/zhsoft 目录设定为 根目录,从而在 /usr/zhsoft/var/lib/rpm 目录创建 RPM 数据库文件。当然,也可使用--dbpath 选项设定创建 RPM 数据库的目 录,效果一样。从上看出,RPM 创建了 8 个数据库文件,其中:conflictsindex.rpm 是冲突索引文件,fileindex.rpm 是文件索 引文件,groupindex.rpm 是类别索引文件,nameindex.rpm 是软件名索引文件,packages.rpm 是已安装的软件包数据文 件,providesindex.rpm 是软件包提供的功能索引文件,requireby.rpm 是功能依赖文件,triggerindex.rpm 是触 发索引文件。

2. --rebuilddb :

本选项用于重组 RPM 数据库。一般情况下不需重组数据库,仅当用 RPM 安装或升级软件时提示"free list corrupt (42) - contact rpm-list@redhat.com",这说明 RPM 数据库索引出现故障了,重组一下数据库应该可以解决问题。

重组实例:

```
# rpm --rebuilddb -vv
D: rebuilding database in rootdir /
D: creating directory: /var/lib/rpmrebuilddb.690
D: opening old database
```

```
D: opening database mode 0x0 in //var/lib/rpm/
D: opening new database
D: opening database mode 0x42 in /var/lib/rpmrebuilddb.690/
#
```

注: 重组时选用-vv 选项以得知 RPM 具体在干些什么。从上例看出,重组过程中,RPM 利用临时目录,将老数据库中数据倒出来,存储到新的数据库中,索引因而得以重建。并且重组过程中 RPM 并没有报错,说明重组是成功的。

二、属性重置选项

1. --setperms :

此选项的作用是根据查询出的 RPM 数据库里的包文件属性,重新设置一下文件权限,恢复其本来的权限。
此选项的用法是:

```
rpm --setperms [-afpg] [软件标识 1 或包裹文件 1] [软件标识 2 或包裹文件 2]...
```

注: []所括为可选项,-a 选项用于重新设置所有已安装的软件包的文件权限,-f 选项用于重新设置拥有指定文件的已安装软件包的文件权限,-p 选项用于重新设 置与指定包裹文件中的文件同名的已在系统安装的文件的权限,-g 选项用于重新设置所有已安装的属于指定类别的软件包的文件的权限。

```
# rpm --setperms -f /bin/sh
#
```

注:本例重新设置拥有/bin/sh 文件的软件包内各文件的权限。

2. --setugids :

此选项根据查询出来的 RPM 数据库里的文件属性,重新设置一下文件的属主与属组,恢复其本来的属主与属组。
此选项的用法是:
rpm --setugids [-afpg] [软件标识 1 或包裹文件 1] [软件标识 2 或包裹文件 2]...
注:[]所括为可选项,-a,-f,-p,-g 四个选项的作用同上面的解释,恕不赘述。
本例重置属于"应用/编辑器"类(Applications/Editors)的软件包内各文件的属主与属组。

```
# rpm --setugids -g Applications/Editors
#
```

三、其它选项

1. --showrc :

本选项用以显示当前 RPM 使用的资源配置的内容,包括 RPM 环境变量及其值,CPU 体系兼容信息,操作系统信息和各种宏及其值等等。

注:例中选显示体系与操作系统信息,接着显示宏及其值(RPMRC VALUES),省略了不少输出(以.....表示)。

2. --querytags :

用此选项可输出 RPM 定制查询时可用的所有功能标签。

```
# rpm --querytags
NAME
VERSION
RELEASE
EPOCH
SERIAL
SUMMARY
DESCRIPTION
BUILDTIME
BUILDHOST
INSTALLTIME
......
#
```

3. --version :

用此选项可输出当前 RPM 的版本信息。

4. --help :

用此选项可输出 RPM 的帮助信息。LINUX 系统中的许多命令都有约定,用 --version 选项输出版本号,用--help 选项以输出帮助信息,这种约定很好,RPM 也加以遵循。