RPM 包的构建 - SPEC 基础知识

RPM 系列文章：

- Tag/RPM

spec 文件

制作 rpm 软件包并不是一件复杂的工作，其中的关键在于编写软件包的 spec 描述文件。

要想制作一个 rpm 软件包就必须写一个软件包描述文件 spec。这个文件中包含了软件包的诸多信息，如：软件包的名字、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列表等等。

实际过程中，最关键的地方，是要清楚虚拟路径的位置，以及宏的定义。

文件头

这个区域定义的 `Name`、`Version` 这些字段对应的值可以在后面通过 `%{name}`，`%{version}` 这样的方式来引用，类似于 C 语言中的宏

- `Summary`：用一句话概括该软件s包尽量多的信息。
- `Name`：软件包的名字，最终 rpm 软件包是用该名字与版本号（`Version`）、释出号（`Release`）及体系号来命名软件包的，后面可使用 `%{name}` 的方式引用
- `Version`：软件版本号。仅当软件包比以前有较大改变时才增加版本号，后面可使用 `%{version}` 引用
- `Release`：软件包释出号/发行号。一般我们对该软件包做了一些小的补丁的时候就应该把释出号加 1，后面可使用 `%{release}` 引用
- `Packager`：打包的人（一般喜欢写个人邮箱）
- `Vendor`：软件开发者的名字，发行商或打包组织的信息，例如 `RedFlagCo,Ltd`
- `License`：软件授权方式，通常是GPL（自由软件）或GPLv2,BSD
- `Copyright`：软件包所采用的版权规则。具体有：`GPL(自由软件)`，`BSD`，`MIT`，`Public Domain(公共域)`，`Distributable(贡献)`，`commercial(商业)`，`Share(共享)` 等，一般的开发都写 `GPL`。
- `Group`：软件包所属类别
  - Development/System（开发/系统）
  - System Environment/Daemons（系统环境/守护）
- `Source`：源程序软件包的名字/源代码包的名字，如 `stardict-2.0.tar.gz`。可以带多个用 `Source1`、`Source2` 等源，后面也可以用 `%{source1}`、`%{source2}` 引用

```
Source0: %{name}-boost-%{version}.tar.gz    ← 源码包名称(可以使用URL)，可以用SourceN指定多个，如配置文件
#Patch0: some-bugs0.patch                   ← 如果需要打补丁，则依次填写
#Patch1: some-bugs1.patch                   ← 如果需要打补丁，则依次填写
```

- `BuildRequires`：制作过程中用到的软件包，构建依赖
- `Requires`：安装时所需软件包
  - `Requires(pre)`：指定不同阶段的依赖
- `BuildRoot`：这个是安装或编译时使用的「虚拟目录」，打包时会用到该目录下文件，可查看安装后文件路径，例如：`BuildRoot: %_topdir/BUILDROOT`。
- `Prefix: %{_prefix}` 这个主要是为了解决今后安装 rpm 包时，并不一定把软件安装到 rpm 中打包的目录的情况。这样，必须在这里定义该标识，并在编写 `%install` 脚本的时候引用，才能实现 rpm 安装时重新指定位置的功能
- `BuildArch`：指编译的目标处理器架构，`noarch` 标识不指定，但通常都是以 `/usr/lib/rpm/marcros` 中的内容为默认值
- `%description`：软件包详细说明，可写在多个行上。这样任何人使用 `rpm -qi` 查询您的软件包时都可以看到它。您可以解释这个软件包做什么，描述任何警告或附加的配置指令，等等。
- `URL`：软件的主页

RPM 包信息查看

我通过命令查看了 nginx 包的信息，如下：

```
# 查看头部信息

$ rpm -qpi ./nginx-1.12.2-2.el7.x86_64.rpm
Name        : nginx
Epoch       : 1
Version     : 1.12.2
Release     : 2.el7
Architecture: x86_64
Install Date: (not installed)
Group       : System Environment/Daemons
Size        : 1574949
License     : BSD
Signature   : RSA/SHA256, Tue 06 Mar 2018 05:44:06 PM CST, Key ID 6a2faea2352c64e5
Source RPM  : nginx-1.12.2-2.el7.src.rpm
Build Date  : Tue 06 Mar 2018 05:27:44 PM CST
Build Host  : buildhw-02.phx2.fedoraproject.org
Relocations : (not relocatable)
Packager    : Fedora Project
Vendor      : Fedora Project
URL         : http://nginx.org/
Bug URL     : https://bugz.fedoraproject.org/nginx
Summary     : A high performance web server and reverse proxy server
Description :
Nginx is a web server and a reverse proxy server for HTTP, SMTP, POP3 and
IMAP protocols, with a strong focus on high concurrency, performance and low
memory usage.

# 查看脚本内容

$ rpm --scripts -qp ./nginx-1.12.2-2.el7.x86_64.rpm
postinstall scriptlet (using /bin/sh):

if [ $1 -eq 1 ] ; then
        # Initial installation
        systemctl preset nginx.service >/dev/null 2>&1 || :
fi
preuninstall scriptlet (using /bin/sh):

if [ $1 -eq 0 ] ; then
        # Package removal, not upgrade
        systemctl --no-reload disable nginx.service > /dev/null 2>&1 || :
        systemctl stop nginx.service > /dev/null 2>&1 || :
fi
postuninstall scriptlet (using /bin/sh):

systemctl daemon-reload >/dev/null 2>&1 || :

if [ $1 -ge 1 ]; then
    /usr/bin/nginx-upgrade >/dev/null 2>&1 || :
fi
```

RPM 包依赖

依赖关系定义了一个包正常工作需要依赖的其他包，RPM在升级、安装和删除的时候会确保依赖关系得到满足。

BuildRequires

定义构建时依赖的软件包，在构建机器编译 rpm 包时需要的辅助工具，以逗号分隔。

假如，要求编译 `myapp` 时，gcc 的版本至少为 4.4.2，则可以写成 `gcc >=4.2.2`

Requires

定义安装时的依赖包，该 rpm 包所依赖的软件包名称，就是指编译好的 rpm 软件在其他机器上安装时，需要依赖的其他软件包。

可以用 `>=` 或 `<=` 表示大于或小于某一特定版本。 `>=` 号两边需用空格隔开，而不同软件名称也用空格分开。

格式：

```
Requires:       libpng-devel >= 1.0.20 zlib
```

Copy

其它写法例如：

```
Requires: bzip2 = %{version}, bzip2-libs =%{version}
```

Copy

还有例如 `PreReq` 、 `Requires(pre)` 、 `Requires(post)` 、 `Requires(preun)` 、 `Requires(postun)` 、 `BuildRequires` 等都是针对不同阶段的依赖指定。

关于 `pre` 、 `post` 、 `preun` 、 `postun` 含义理解，感觉 `post` 有一种“完成”的意思：

```
# 安装前执行的脚本，语法和shell脚本的语法相同
%pre
# 安装后执行的脚本
%post
# 卸载前执行的脚本
%preun
# 卸载完成后执行的脚本
%postun
# 清理阶段，在制作完成后删除安装的内容
```

Copy

例如：

```
PreReq: capability>=version      #capability包必须先安装
Conflicts:bash>=2.0              #该包和所有不小于2.0的bash包有冲突
```

Copy

BuildRoot

该参数非常重要，因为在生成 rpm 的过程中，执行 `make install` 时就会把软件安装到上述的路径中，在打包的时候，同样依赖“虚拟目录”为“根目录”进行操作。后面可使用 `$RPM_BUILD_ROOT` 方式引用。

考虑到多用户的环境，一般定义为：

```
%{_tmppath}/%{name}-%{version}-%{release}-root
# OR
%{_tmppath}/%{name}-%{version}-%{release}-buildroot-%(%{__id_u} -n}
```

Copy

下面介绍 spec 脚本主体：

%prep 阶段

这个阶段是「预处理」，通常用来执行一些解开源程序包的命令，为下一步的编译安装作准备。

`%prep` 和下面的 `%build` ， `%install` 段一样，除了可以执行 rpm 所定义的宏命令（以 `%` 开头）以外，还可以执行 `SHELL` 命令，命令可以有很多行，如我们常写的 `tar` 解包命令。功能上类似于 `./configure` 。

`%prep` 阶段进行实际的打包准备工作，它是使用节前缀 `%prep` 表示的。主要功能有：

- 将文件（ `SOURCES/` ）解压到构建目录（ `BUILD/` ）
- 应用 `Patch` （打补丁）（ `SOURCES/ => BUILD/` ）
- 描述 `rm -rf $RPM_BUILD_ROOT`
- 描述或编辑本部分用到的命令到 `PreReq`
- 通过 `-b .XXX` 描述补丁备份

它一般包含 `%setup` 与 `%patch` 两个命令。 `%setup` 用于将软件包打开，执行 `%patch` 可将补丁文件加入解开的源程序中。

示例：

```
%prep
%setup -q                               ← 宏的作用是解压并切换到目录
#%patch0 -p1                            ← 如果需要打补丁，则依次写
#%patch1 -p1                            ← 如果需要打补丁，则依次写
```

Copy

%setup

这个宏解压源代码，将当前目录改为源代码解压之后产生的目录。这个宏还有一些选项可以用。例如，在解压后， `%setup` 宏假设产生的目录是 `%{name}-%{version}`

```
%setup -n %{name}-%{version} #把源码包解压并放好
```

Copy

主要的用途就是将 `%sourcedir` 目录下的源代码解压到 `%builddir` 目录下，也就是将 `~/rpmbuild/SOURCES` 里的包解压到 `~/rpmbuild/BUILD/%{name}-%{version}` 中。一般用 `%setup -c` 就可以了，但有两种情况：

- 一就是同时编译多个源码包，
- 二就是源码的 tar 包的名称与解压出来的目录不一致，此时，就需要使用 `-n` 参数指定一下。

```
%setup 不加任何选项，仅将软件包打开。
%setup -a 切换目录前，解压指定 Source 文件，例如 `-a 0` 表示解压 `Source0`
%setup -n newdir 将软件包解压在newdir目录。
%setup -c 解压缩之前先产生目录。
%setup -b num 将第 num 个 source 文件解压缩。
%setup -D 解压前不删除目录
%setup -T 不使用default的解压缩操作。
%setup -T -b 0 将第 0 个源代码文件解压缩。
%setup -c -n newdir 指定目录名称 newdir，并在此目录产生 rpm 套件。
%patch 最简单的补丁方式，自动指定patch level。
%patch 0 使用第0个补丁文件，相当于%patch ?p 0。
%patch -s 不显示打补丁时的信息。
%patch -T 将所有打补丁时产生的输出文件删除
```

Copy

应该 `-q` 参数给 `%setup` 宏。这会显著减少编译日志文件的输出，尤其是源代码包会解压出一堆文件的时候。

在 `~/rpmbuild/BUILD/%{name}-%{version}` 目录中进行，使用标准写法，会引用 `/usr/lib/rpm/marcros` 中定义的参数。

%patch

这个宏将头部定义的补丁应用于源代码。如果定义了多个补丁，它可以用一个数字的参数来指示应用哪个补丁文件。它也接受 `-b extension` 参数，指示 RPM 在打补丁之前，将文件备份为扩展名是 `extension` 的文件。

通常补丁都会一起在源码 `tar.gz` 包中，或放到 `SOURCES` 目录下。一般参数为：

- `%patch -p1` 使用前面定义的 `Patch` 补丁进行， `-p1` 是忽略 `patch` 的第一层目录
- `%Patch2 -p1 -b xxx.patch` 打上指定的补丁， `-b` 是指生成备份文件

%build 阶段

本段是「构建」阶段，这个阶段会在 `%_builddir` 目录下执行源码包的编译。一般是执行执行常见的 `configure` 和 `make` 操作。

该阶段一般由多个 `make` 命令组成。与 `%prep` 段落一样，这些命令可以是 `shell` 命令，也可以是宏。

记住两点：

1. `%build` 和 `%install` 的过程中，都必须把编译和安装的文件定义到“虚拟根目录” 中
2. `%file` 中必须明白，用的是相对目录

这个阶段我们最常见只有两条指令：

```
%configure
make %{?_smp_mflags} OPTIMIZE="%{optflags}"          ← 多核则并行编译
```

`%configure` 这个不是关键字，而是 rpm 定义的标准宏命令。意思是执行源代码的 `configure` 配置。会自动将 `prefix` 设置成 `/usr` 。

这个 `%{?_smp_mflags} %{optflags}` 是什么意思呢？

```
$ rpm --eval "%{?_smp_mflags}"
-j4
$ rpm --eval "%{optflags}"
-O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong --param=ssp-buffer-size=4 -grecord-
gcc-switches   -m64 -mtune=generic
```

所以，上面那个命令就是 `make -j4` 。问题又来了， `make -j4` 表示什么意思？

> 都是一些优化参数

- Linux kernel 笔记（26）——利用“make -j”提高编译 `kernel` 速度
- CSDB-make -j4是什么意思

%install 阶段

「安装」阶段，就是执行 `make install` 命令操作。开始把软件安装到虚拟的根目录中。这个阶段会在 `%buildrootdir` 目录里建好目录结构，然后将需要打包到 rpm 软件包里的文件从 `%builddir` 里拷贝到 `%_buildrootdir` 里对应的目录里。

在 `~/rpmbuild/BUILD/%{name}-%{version}` 目录中进行 `make install` 的操作。 `%install` 很重要，因为如果这里的路径不对的话，则下面 `%file` 中寻找文件的时候就会失败。

特别需要注意的是： `%install` 部分使用的是绝对路径，而 `%file` 部分使用则是相对路径，虽然其描述的是同一个地方。千万不要写错。

当用户最终用 `rpm -ivh name-version.rpm` 安装软件包时，这些文件会安装到用户系统中相应的目录里。

```
%install
if [-d %{buildroot}]; then
   rm -rf %{buildroot}                    ← 清空下安装目录，实际会自动清除
fi
make install DESTDIR=%{buildroot}         ← 安装到buildroot目录下
%{__install} -Dp -m0755 contrib/init.d %{buildroot}%{_initrddir}/foobar
%{__install} -d %{buildroot}%{_sysconfdir}/foobar.d/
```

「翻译」一下：

```
make install DESTDIR=/root/rpmbuild/BUILDROOT/%{name}-%{version}-%{release}.x86_64
make install DESTDIR=/root/rpmbuild/BUILDROOT/%{name}-%{version}-%{release}.x86_64
/usr/bin/install -d /root/rpmbuild/BUILDROOT/%{name}-%{version}-%{release}.x86_64/etc/foobar.d/
```

需要说明的是，这里的 `%install` 主要就是为了后面的 `%file` 服务的。所以，还可以使用常规的系统命令：

```
install -d $RPM_BUILD_ROOT/
cp -a * $RPM_BUILD_ROOT/
```

其中 `%{buildroot}` 和 `$RPMBUILDROOT` 是等价的， `%{buildroot}` 必须全部用小写，不然要报错。

注意区分 `$RPM_BUILD_ROOT` 和 `$RPM_BUILD_DIR` ：

- `$RPM_BUILD_ROOT` 是指开头定义的 `BuildRoot` ，是 `%file` 需要的。
- `$RPM_BUILD_DIR` 通常就是指 `~/rpmbuild/BUILD`

scripts section 没必要可以不填

- `%pre` 安装前执行的脚本
- `%post` 安装后执行的脚本
- `%preun` 卸载前执行的脚本
- `%postun` 卸载后执行的脚本
- `%pretrans` 在事务开始时执行脚本
- `%posttrans` 在事务结束时执行脚本

`%preun` `%postun` 的区别是什么呢？

- 前者在升级的时候会执行，后者在升级 rpm 包的时候不会执行

%files 阶段

本段是文件段，主要用来说明会将 `%{buildroot}` 目录下的哪些文件和目录最终打包到rpm包里。定义软件包所包含的文件，分为三类：

- 说明文档（doc）
- 配置文件（config）
- 执行程序

还可定义文件存取权限，拥有者及组别。

这里会在虚拟根目录下进行，千万不要写绝对路径，而应用宏或变量表示相对路径。

在 `%files` 阶段的第一条命令的语法是：

```
%defattr(文件权限,用户名,组名,目录权限)
```

注意点：同时需要在 `%install` 中安装。

```
%files
%defattr (-,root,root,0755)                        ← 设定默认权限
%config(noreplace) /etc/my.cnf                     ← 表明是配置文件，noplace表示替换文件
%doc %{src_dir}/Docs/ChangeLog                     ← 表明这个是文档
%attr(644, root, root) %{_mandir}/man8/mysqld.8*   ← 分别是权限，属主，属组
%attr(755, root, root) %{_sbindir}/mysqld
```

`%exclude` 列出不想打包到 rpm 中的文件。注意：如果 `%exclude` 指定的文件不存在，也会出错的。

在安装 rpm 时，会将可执行的二进制文件放在 `/usr/bin` 目录下，动态库放在 `/usr/lib` 或者 `/usr/lib64` 目录下，配置文件放在 `/etc` 目录下，并且多次安装时新的配置文件不会覆盖以前已经存在的同名配置文件。

关于 `%files` 阶段有两个特性：

1. `%{buildroot}` 里的所有文件都要明确被指定是否要被打包到 rpm里。什么意思呢？假如， `%{buildroot}` 目录下有 4 个目录 a、b、c和d，在 `%files` 里仅指定 a 和 b 要打包到 rpm 里，如果不把 c 和 d 用 `exclude` 声明是要报错的；
2. 如果声明了 `%{buildroot}` 里不存在的文件或者目录也会报错。

关于 `%doc` 宏，所有跟在这个宏后面的文件都来自 `%{_builddir}` 目录，当用户安装 rpm 时，由这个宏所指定的文件都会安装到 `/usr/share/doc/name-version/` 目录里。

%clean

清理段，可以通过 `--clean` 删除 `BUILD`

编译完成后一些清理工作，主要包括对 `%{buildroot}` 目录的清空(这不是必须的)，通常执行诸如 `make clean` 之类的命令。

%changelog

本段是修改日志段，记录 spec 的修改日志段。你可以将软件的每次修改记录到这里，保存到发布的软件包中，以便查询之用。

每一个修改日志都有这样一种格式：

- 第一行是：`* 星期 月 日 年 修改人 电子信箱` 。其中：星期、月份均用英文形式的前 3 个字母，用中文会报错。
- 接下来的行写的是修改了什么地方，可写多行。一般以减号开始，便于后续的查阅。

```
%changelog
* Fri Dec 29 2012 foobar <foobar@kidding.com> - 1.0.0-1
- Initial version
```

宏

在定义文件的安装路径时，通常会使用类似 `%_sharedstatedir` 的宏，这些宏一般会在 `/usr/lib/rpm/macros` 中定义。关于宏的语法，可以查看 Macro syntax

RPM 内建宏定义在 `/usr/lib/rpm/redhat/macros` 文件中，这些宏基本上定义了目录路径或体系结构等等；同时也包含了一组用于调试 spec 文件的宏。

所有宏都可以在 `/usr/lib/rpm/macros` 找到，附录一些常见的宏：

```
%{_sysconfdir}        /etc
%{_prefix}            /usr
%{_exec_prefix}       %{_prefix}
%{_bindir}            %{_exec_prefix}/bin
%{_lib}               lib (lib64 on 64bit systems)
%{_libdir}            %{_exec_prefix}/%{_lib}
%{_libexecdir}        %{_exec_prefix}/libexec
%{_sbindir}           %{_exec_prefix}/sbin
%{_sharedstatedir}    /var/lib
%{_datadir}           %{_prefix}/share
%{_includedir}        %{_prefix}/include
%{_oldincludedir}     /usr/include
%{_infodir}           /usr/share/info
%{_mandir}            /usr/share/man
%{_localstatedir}     /var
%{_initddir}          %{_sysconfdir}/rc.d/init.d
%{_topdir}            %{getenv:HOME}/rpmbuild
%{_builddir}          %{_topdir}/BUILD
%{_rpmdir}            %{_topdir}/RPMS
%{_sourcedir}         %{_topdir}/SOURCES
%{_specdir}           %{_topdir}/SPECS
%{_srcrpmdir}         %{_topdir}/SRPMS
%{_buildrootdir}      %{_topdir}/BUILDROOT
%{_var}               /var
%{_tmppath}           %{_var}/tmp
%{_usr}               /usr
%{_usrsrc}            %{_usr}/src
%{_docdir}            %{_datadir}/doc
%{buildroot}          %{_buildrootdir}/%{name}-%{version}-%{release}.%{_arch}
$RPM_BUILD_ROOT       %{buildroot}
```

利用 rpmbuild 构建 rpm 安装包时，通过命令 `rpm --showrc|grep prefix` 查看。

通过 `rpm --eval "%{macro}"` 来查看具体对应路径。

比如我们要查看 `%{_bindir}` 的路径，就可以使用命令 `rpm --eval "%{ _bindir}"` 来查看。

```
%{_topdir}            %{getenv:HOME}/rpmbuild
%{_builddir}          %{_topdir}/BUILD
%{_rpmdir}            %{_topdir}/RPMS
%{_sourcedir}         %{_topdir}/SOURCES
%{_specdir}           %{_topdir}/SPECS
%{_srcrpmdir}         %{_topdir}/SRPMS
%{_buildrootdir}      %{_topdir}/BUILDROOT

Note: On releases older than Fedora 10 (and EPEL), %{_buildrootdir} does not exist.
Build flags macros

%{_global_cflags}     -O2 -g -pipe
%{_optflags}          %{__global_cflags} -m32 -march=i386 -mtune=pentium4 # if redhat-rpm-config is installed
```

变量

`define` 定义的变量类似于局部变量，只在 `%{!?foo: ... }` 区间有效，不过 SPEC 并不会自动清除该变量，只有再次遇到 `%{}` 时才会清除

define vs. global

两者都可以用来进行变量定义，不过在细节上有些许差别，简单列举如下：

- `define` 用来定义宏， `global` 用来定义变量；
- 如果定义带参数的宏 (类似于函数)，必须要使用 `define` ；
- 在 `%{}` 内部，必须要使用 `global` 而非 `define` ；
- `define` 在使用时计算其值，而 `global` 则在定义时就计算其值；

可以简单参考如下的示例。

```
#--- %prep之前的参数是必须要有的
Name:           mysql
Version:        5.7.17
Release:        1%{?dist}
Summary:        MySQL from FooBar.
License:        GPLv2+ and BSD

%description
It is a MySQL from FooBar.

%prep
#--- 带参数时，必须使用%define定义
%define myecho() echo %1 %2
%{!?bar: %define bar defined}

echo 1: %{bar}
%{myecho 2: %{bar}}
echo 3: %{bar}

# 如下是输出内容
```

```
#1: defined
#2: defined
#3: %{bar}
```

3 的输出是不符合预期的，可以将 `%define` 修改为 `global` 即可

%{?dist} 表示什么含义？

不加问号，如果 `dist` 有定义，那么就会用定义的值替换，否则就会保 `%{dist}`；
加问好，如果 `dist` 有定义，那么也是会用定义的值替换，否则就直接移除这个tag `%{?dist}`

举例：

```
$ rpm -E 'foo:%{foo}'$'\n''bar:%{?bar}'
foo:%{foo}
bar:

$ rpm -D'foo foov' -E 'foo:%{foo}'$'\n''bar:%{?bar}'
foo:foov
bar:

$ rpm -D'foo foov' -D'bar barv' -E 'foo:%{foo}'$'\n''bar:%{?bar}'
foo:foov
bar:barv
```

小结

看了以上SPEC的总结，以为了解差不多了，结果看了网络域的SPEC文件，自己真实 `too young too simple`。

参考

- 开源中国-RPM包的制作 对 rpmbuild 目录的创建及生成介绍比较详细
- 夜鸣猪-RPM包rpmbuild SPEC文件深度说明
- OpenSUSE-spec 文件指南
- CSDN-RPM打包原理、示例、详解及备查 很详细，有示例
- Fedora-WIKI-How to create an RPM package/zh-cn 中文介绍
- IBM-RPM 打包技术与典型 SPEC 文件分析
- JIN-YANG-RPM 包制作 总结很全，还介绍了签名
- CSDN-RPM构建 - SPEC文件参数解析 该博主，总共围绕 RPM 构建写了几篇文章的
- 原-关于rpm打包中的条件判断 介绍了 spec 中条件判断的语法
- Whats the meaing of 1%{?dist} in SPEC file in RPM package

分类: Linux
标签: rpm, spec, build

« 上一篇： RPM Yum 相关命令及参数
» 下一篇： RPM 包的构建 - 实例

posted @ 2019-03-06 00:04 Michael翔 阅读(11375) 评论(0) 编辑 收藏 举报