Автоматический поиск зависимостей в rpm-пакетах

Алексей Турбин <at@altlinux.org>

5 июля 2007 г.

Аннотация

Разработана модульная система автоматического поиска зависимостей для сборочной среды на основе менеджера пакетов RPM. Использование автоматических зависимостей на основе виртуальных пакетов (Requires и Provides) необходимо для более точного учёта внутренних связей между пакетами на стадии их установки или обновления. Модульная система позволяет независимо разрабатывать и добавлять в сборочную среду новые типы зависимостей. В последней части доклада кратко рассмотрен алгоритм автоматического поиска и оптимизации сборочных зависимостей (BuildRequires) на основе трассировки сборки.

Введение

На последних стадиях сборки rpm-пакета происходит формирование зависимостей Requires (требуемые пакеты) и Provides (зависимости, предоставляемые пакетом). Помимо непосредственного указания (вручную) зависимостей в spec-файле пакета, предусмотрена процедура автоматического поиска зависимостей. Для этого `rpm-build` запускает скрипты `find-requires` и `find-provides`, результат работы которых добавляется к непосредственным зависимостям пакета.

Автоматические зависимости, как правило, формируются при помощи виртуальных пакетов, т. е. для них используется отдельное пространство имён, которое не пересекается с названиями самих пакетов. В частности, виртуальные пространства имён используются для поиска автоматических зависимостей между модулями интерпретируемых языков программирования. Так, виртуальные зависимости для модулей языка Perl имеют вид `perl(Getopt/Long.pm)`, а для языка Python (версии 2.4) — `python2.4(getopt)`. Использование виртуальных зависимостей позволяет более гибко изменять содержимое пакетов (и даже переименовывать пакеты), а также защищает от ошибок упаковки и несовместимых изменений.

К существенным недостатком системы автоматического поиска зависимостей можно было отнести её монолитность: добавление нового типа зависимостей требовало ad hoc модификации скриптов `find-requires` и `find-provides` в пакете `rpm-build`. Вместе с тем, разработка программ для поиска новых типов зависимостей велась независимо.

Внедрение модульной системы поиска зависимостей происходило в два этапа. На первом этапе некоторые алгоритмы поиска специальных зависимостей были выделены в отдельные программы; но условия запуска каждой из этих программ по-прежнему оставались жёстко закодированными в скриптах `rpm-build`. На следующем этапе система стала уже полностью модульной: процедуры вызова программ были полностью унифицированы, а в скриптах `find-requires` и `find-provides` остался только вспомогательный код диспетчеризации.

1 Модульная система поиска зависимостей

В новой системе скрипты `find-requires` и `find-provides` работают единообразно. Рассмотрим работу `find-requires`.

Этот скрипт получает на входе список всех файлов пакета, а на выходе формирует список зависимостей (всех типов). Сначала для каждого входного файла определяется его «тип» при помощи утилиты `file(1)`. Затем список файлов и их типов обрабатывается модульными программами поиска зависимостей.

Для каждого типа зависимостей существуют две программы. Первая программа — с суффиксом (расширением) `.req.files` — выделяет список файлов (на основе их «типа»), для которых будет производиться поиск зависимостей данного типа. Например, программа `perl.req.files` среди всех файлов пакета выделяет файлы с кодом на языке Perl. Вторая программа — с суффиксом `.req` (в данном случае `perl.req`) — последовательно производит поиск зависимостей данного типа для каждого файла из полученного списка.

Модульность системы состоит в том, что в процессе поиска зависимостей запускаются все доступные программы (по шаблону `*.req.files` и `*.req`) из каталога `/usr/lib/rpm`.

В скрипте `find-provides` аналогичным образом происходит диспетчеризация по суффиксу `.prov` (вместо `.req`). Таким образом, для добавления нового типа зависимостей нужно написать четыре программы: программу выбора файлов и программу обработки файлов, соответственно для Requires и Provides.

2 Модульные программы поиска зависимостей

Среди базовых модулей в пакете `rpm-build` выделим следующие (код был написан в основном Дмитрием Левиным и затем выделен в модули):

- `lib.req` и `lib.prov` — зависимости системных разделяемых библиотек; зависимости имеют вид `libc.so.6(GLIBC_2.0)`.
- `shell.req` — зависимости на исполняемые файлы в Bourne Shell скриптах; здесь не используются виртуальные пакеты, а все Requires-зависимости непосредственно отображаются в названия пакетов (так, использование программы `cat` порождает зависимость на пакет `coreutils`). Для реализации в пакет `bash` был добавлен специальный режим анализа кода (наподобие syntax check).

- `shebang.req` — анализирует первую строчку исполняемых скриптов (которая начинается с символов `#!`) и добавляет зависимость на соответствующий интерпретатор.

- `pam.req` и `pam.prov` — зависимости на модули подсистемы PAM в файлах `/etc/pam.d/*`. Зависимости имеют вид `PAM(pam_userpass.so)`.

- `pkgconfig.req` и `pkgconfig.prov` — отражает зависимости в файлах `/usr/lib/pkgconfig/*.pc`; это в основном зависимости между devel-пакетами, имеют вид `pkgconfig(glib-2.0)`, учитываются также версии.

Среди дополнительных модулей можно выделить следующие:

- `perl.req` и `perl.prov` – поиск зависимостей в скриптах и модулях Perl. Для поиска Requires-зависимостей реализован анализатор внутреннего дерева байткода `B::PerlReq`. Зависимости имеют вид `perl(Getopt/Long.pm)`, учитываются версии модулей. (Пакет `rpm-build-perl`, разработан автором доклада.)

- `python.req` и `python.prov` – поиск зависимостей в скриптах и модулях Python. В реализации использован стандартный модуль `parser`. Зависимости имеют вид `python2.4(getopt)`. (Пакет `rpm-build-python`, автор — Андрей Орлов.)

- `tcl.req` и `tcl.prov` – поиск зависимостей в скриптах и модулях Tcl. Зависимости имеют вид `tcl(Tk)`. (Пакет `rpm-build-tcl`, автор — Сергей Большаков.)

Разработан также поиск зависимостей в байткоде Mono (пакет `rpm-build-mono`, Ильдар Мулюков), предложена реализация поиска зависимостей в Java-классах.

3 Автоматический поиск сборочных зависимостей

Отдельной задачей является автоматический поиск сборочных зависимостей BuildRequires. Программа `buildreq`, написанная Дмитрием Левиным, использует `strace(1)` для трассировки доступа к файлам в процессе сборки пакета. По списку файлов, используемых в процессе сборки, `buildreq` строит список пакетов, необходимых для сборки. Как правило, этот список пакетов является очень большим и поэтому малоинформативен для человека. Этот список поддаётся оптимизации. Например, если при сборке были использованы пакеты `glibc` и `glibc-devel`, то в списке достаточно оставить `glibc-devel`, поскольку этот пакет зависит от `glibc`. Какой-то другой пакет, в свою очередь, может зависеть от `glibc-devel`, что даёт возможность исключить последний.

Для оптимизации списка сборочных зависимостей был разработан новый алгоритм, который обладает следующими свойствами. 1) Оптимизация является сильной, т. е. алгоритм находит почти оптимальное подмножество пакетов, замыкание которого по зависимостям даст исходное множество. Неоптимальность возможна лишь при разрыве циклов. 2) Оптимизация является почти корректной; существуют очень редкие «патологические» случаи двусмысленных виртуальных зависимостей, при которых список может быть оптимизирован некорректно.

Алгоритм работает следующим образом:

1. По списку пакетов на входе выстраиваются пары пакетов с непосредственными зависимостями (`A->B` — пакет A требует пакет B). Помимо непосредственных зависимостей, используется также соединение по Requires и Provides, т. е. учитываются варианты `A->v->B`, где `v` — виртуальный пакет, предоставляемый пакетом B и требуемый пакетом A.

2. Полученное множество пар задаёт частичный порядок, который можно дополнить до линейного, т. е. линейно упорядочить список пакетов по взаимным зависимостям между ними. Для этого используется программа топологической сортировки `tsort(1)`, которая также помогает корректно разорвать циклы (простейшим циклом является взаимная зависимость между пакетами `A->B`, `B->A`).

3. Последний проход алгоритма реализует идею т. н. «решета Эратосфена». Пакеты, расположенные в начале упорядоченного списка, содержат зависимости на другие пакеты, которые расположены в списке ближе к концу. Оптимизация состоит в «вычёркивании» из списка всех «зависимых» пакетов (аналогия с вычёркиванием составных чисел, которые «сводятся» к простым).

Полученный таким образом оптимизированный список BuildRequires добавляется в spec-файл пакета.

Заключение

Отвлекаясь от деталей, хочется подчеркнуть *технологичность* описанных решений. Программы не ошибаются, или, во всяком случае, не «опечатываются». В большинстве случаев это позволяет полностью освободить разработчика от указания зависимостей вручную.