/AOSCC 2021

apm: 实验性包管理器

Leo Shen

2021-09-20

/Motivation

- **apt 的依赖解析算法难以预测**
  - 在自动化的过程任务调度方面难以实现
  - 之前也出现过升级时卸穿系统的情况
- **apt 用户体验一般**
  - `update`, `upgrade` 和 `full-upgrade` 反直觉
  - `apt-get`, `apt-get` 和 `apt-file` 是分开的程序

/Architecture

- **软件包关系计算**
  - 下载并读取 deb 软件包数据库
  - 读取 blueprint，从本地数据库中选择最优版本的软件包
  - 将可用的软件包数据库和之前选出需要安装的包放入依赖解析器，算出初始版本的依赖树
    - 如果依赖无法满足，这一步就会报错
    - 如果依赖可满足，这一步会得出一个可行但不一定最优化的软件包列表
  - 调用优化算法得出最优的软件包列表
  - 将软件包列表根据依赖关系排序

/Architecture

```
kernel-base
linux+kernel(>3:5.13.0, <3:5.14.0)
util-base
shadow
dpkg
vim
sudo
# System utils
systemd-boot-friend
# Network
iwd
iw
```

A minimal blueprint

/Architecture

- **应用软件包列表**
  - 读取本机上的 dpkg 状态数据库（`/var/lib/dpkg/status`），算出需要更改的软件包（安装，升级或卸载）
  - 下载需要安装或升级的软件包
  - 调用 dpkg 安装，升级及卸载软件包

/Advantages

- **可复现**
  - 只需迁移 `apm.toml` 和 `blueprint` 即可
- **全部功能都包含在一个 binary 里，且命令结构清晰**
- **由于本身无状态，理论上稳定性会比使用本地数据库更好**

Demo Time!

- **布尔可满足性问题（SAT 问题）**
  - 定义变量 A, B, C
  - `(A|B)&(B|C)&!C==true`
  - 使 A 为真， B 为真， C 为否即可使等式成立
  - NP-hard 问题，但有相对优化的解决算法
  - 可以将依赖解析问题归纳到 SAT 问题，即可使用已有的高速解析器

/Algorithm

- **将依赖解析问题归纳到 SAT 问题**
  - A 依赖 B， B 有 B1, B2 版本可选 => `(!A | B1 | B2)`
  - B1， B2 和 A 冲突 => `(!A | !B1 | !B2)`
  - 同个包 (A) 的不同版本 (A1, A2, A3) 不可同时安装 => `(!A1 | !A2 | !A3)`
  - blueprint 上要求 A 安装 => `(A)`
    - 此项必须为真
  - 解析结果不一定为最简或最新，可能包含冗余包和旧包
    - 需要使用其他算法来保证解法最优

A. M. A. !

提问

提問