04-rpath解决so动态库依赖

1. 依赖库搜索流程

动态库或可执行文件，是如何找到它所依赖的库的？是按以下目录顺序来搜索：

- the RPATH binary header (set at build-time) of the library causing the lookup (if any)
- the RPATH binary header (set at build-time) of the executable
- the LD_LIBRARY_PATH environment variable (set at run-time)
- the RUNPATH binary header (set at build-time) of the executable
- base library directories (/lib and /usr/lib)

则当我们所依赖的so库无法找到指定库时，我们可以通过 `ldd xxx.so` 查看这个so库所依赖的其他库以及其路径

如：

```
 ty@ty-pc:~/anaconda3/envs/py3/lib/python3.7/site-packages/pcl$ ldd
_pcl.cpython-37m-x86_64-linux-gnu.so
    linux-vdso.so.1 =>  (0x00007ffe645f9000)
    libpcl_io.so.1.7 => /usr/lib/x86_64-linux-gnu/libpcl_io.so.1.7
    libOpenNI.so.0 => /usr/lib/libOpenNI.so.0 (0x00007f4d70bdc000)
    libpcl_registration.so.1.7 => /usr/lib/x86_64-linux-
gnu/libpcl_registration.so.1.7
    libpcl_segmentation.so.1.7 => /usr/lib/x86_64-linux-
gnu/libpcl_segmentation.so.1.7
    libpcl_features.so.1.7 => /usr/lib/x86_64-linux-
gnu/libpcl_features.so.1.7
    libpcl_filters.so.1.7 => /usr/lib/x86_64-linux-
gnu/libpcl_filters.so.1.7
    libpcl_sample_consensus.so.1.7 => not found
    libpcl_surface.so.1.7 => /usr/lib/x86_64-linux-
gnu/libpcl_surface.so.1.7
    ...
```

如果某一行末尾为 `not found` ，则说明有依赖库没有找到对应的文件。

2. 查看RPATH

当我们发现缺少so库时，说明此so的依赖库搜索路径中并不包含所需要的so。此时我们可以通过修改其rpath路径，将其指向包含依赖so的路径。首先查看其已有RPATH路径：

```
readelf -d _pcl.cpython-37m-x86_64-linux-gnu.so | grep RPATH
```

或

```
patchelf --print-rpath _pcl.cpython-37m-x86_64-linux-gnu.so
```

3. 修改RPATH

如果以上打印出的路径中不包含步骤1中 `not found` 所需要的so库，则可以通过 `locate xxx.so` 的方式，全局搜索这个so库，然后找到其路径，例如： `/usr/lib/OpenNI2/Drivers/libDummyDevice.so`

首先下载patchelf源码 [https://github.com/NixOS/patchelf/releases]，并进行编译安装patchelf工具：

```
./configure
make
sudo make install
```

然后执行命令，将这个依赖so所在目录设置给当前so：

```
patchelf --set-rpath /usr/lib/OpenNI2/Drivers ./_pcl.cpython-36m-x86_64-linux-gnu.so
```

4. 验证so依赖

再次执行 `ldd xxx.so` 如果之前not found的那一行正确链接，说明配置ok。