像极客一样提取Android的Root权限

原创

蒙娜丽宁 于 2020-10-19 10:24:00 发布 5530 收藏 77
分类专栏： Android 文章标签： linux java android rom

本文链接：https://blog.csdn.net/nokiaguy/article/details/109172949


Android 专栏收录该内容
360 篇文章 1 订阅
订阅专栏

本文将深入揭示提取Android ROOT权限的完整过程。这一过程与网上的方法有很大的差异。很多网上提取ROOT权限的方式都是使用别人做好的程序，有的甚至点击一下按钮就会自动完成所有的工作。这样做尽管可以成功提取ROOT权限，但读者并不能了解其中的原理，而且由于Android设备的千差万别，可能并不是每一种Android设备都可以很容易找到提取ROOT权限的工具。所以最通用的方法就是尽可能利用现成的工具来完成提取ROOT权限的工作。那么现成的工具有什么呢？其实主要就是Android源代码以及Linux内核源代码。也就是说，大多数工作都可以通过这些源代码来解决。当了解了这一过程的原理后，如果并没有找到合适的提取ROOT权限的工具，就可以通过本文介绍的方法很容易获取Android设备的ROOT权限。

本文介绍的提取ROOT权限的方法主要针对Nexus 7，对于其他Android设备也可以使用类似的方法。但必须满足如下两个条件。

> Android设备允许进入bootloader模式。有少数的Android设备关闭了bootloader模式，所以无法刷机。根据笔者的经验，Google的几个亲儿子（Nexus系列），HTC、三星的Android设备基本都没问题，其他厂商的设备就需要具体问题具体分析了。
>
> 可以找到合适的Recovery。有的Android设备自带的Recovery由于功能有限，在刷机上也有一些问题，所以不得不使用第三方的Recovery。通常大厂商生产的Android设备都能找到合适的第三方Recovery。但刷Recovery通常也必须满足第1个条件。

1. 提取ROOT权限的步骤

资深Android玩家和喜欢玩“酷”的Android用户在Android手机到手后的第一件事就是提取ROOT权限，因为Android设备有了ROOT权限，就完全在自己的控制之下了，尽管这样做会带来一定的安全隐患，但在他们看来，自由比任何东西都重要。

可能很多人是通过从网上下载的工具提取ROOT权限的。那么提取ROOT权限真的很复杂吗？No，其实提取ROOT权限远没有编写Android应用复杂，只需要一些简单的步骤就可以搞定，当然，如果有编程的基础，那就会更觉得提取ROOT权限的过程简直太“酷”了。那么在本文先看一下提取ROOT权限的基本步骤。

第1步：刷一个合适的Recovery

对于一部没有ROOT权限的Android设备，将文件复制到系统目录的方法有如下两个。

> 在bootloader模式下复制整个的文件系统。
>
> 在recovery模式下将文件复制到Android设备中的指定目录。

通常复制完整的系统目录结构会使用第1种方法，例如，复制整个system文件系统（system.img）。而要将少量的文件复制到某些文件系统，一般会使用第2种方法。因为这种方法可以通过脚本命令复制指定的文件到系统目录（如/system/xbin），而且不会影响其他无关的数据。

读者可以使用官方的Recovery，也可以下载第三方的Recovery。一般第三方的Recovery会更强大一些。在下一节会详细介绍如何使用第三方的Recovery。

第2步：破解su命令

提取ROOT权限的关键就是执行su命令。不过Android系统带的su命令在默认情况下只能由root用户调用，所以使用su命令之前需要先破解su命令，也就是修改su源代码，将检测调用权限的代码去掉，如果有必要，再加入满足自己需求的代码。也就是所，提取ROOT权限实际上使用的是已经破解了的su命令。在后面的内容会详细介绍如何修改su源代码，并重新生成su命令文件。

第3步：制作Recovery刷机文件（update.zip）

要想将破解后的su命令放到Android的系统目录（如/system/bin）目录中，需要制作Recovery刷机包，也就是一个普通的zip压缩文件。通常文件名为update.zip。不过很多高级的Recovery允许选择其他的zip文件。所以一般Recovery刷机包可以起任何文件名。

Recovery刷机包的内部结构有一定的规则，主要的工作就是编写updater-script脚本文件，该脚本文件的详细编写过程会在后面的内容节介绍。

第4步：执行su命令提取ROOT权限

到这一步就水到渠成了，直接执行su命令，当前的Shell就拥有root权限了（Shell标识符由$变成了#）。现在可以在Shell中浏览只有root权限才能看到的内容，例如，通过ls /data/data命令查看/data/data目录中的文件和目录列表。

第5步：使ROOT权限更完美

其实第4步已经成功使当前的Shell拥有了root权限，不过还有一点缺陷，就是当进入Android设备的Shell时每次都需要执行su命令才能获取root权限，这样有些麻烦。因此还需要修改配置文件，使得一进入Shell就可以立刻拥有root权限。在后面的内容会详细介绍如何修改，以及修改哪些配置文件。

2. 需要一个很酷的recovery

任何一个在Android设备上成功运行的ROM都会自带一个Recovery，通过Recovery，可以将一个zip格式刷机包中的内容复制到指定的系统目录。Nexus 7官方ROM也带了一个Recovery。不过这个Recovery功能有限，而且zip文件还需要签名才能刷机，很麻烦。当然，读者可以定制自己的Recovery，不过定制Recovery的工作量是非常大的，通常不可能在短时间内完成，所以本文暂不做深究，在后续的文章中会专门探讨与定制Recovery相关的问题。既然官方自带的ROM不给力，而定制Recovery又很费劲，那么最节省时间的方法就是使用第三方的Recovery。

现在有很多第三方的Recovery，其中比较著名的是Clockwork Recovery。目前大多数流行的Android设备都有对应的Clockwork Recovery。Clockwork Recovery不仅功能强大，而且zip格式的刷机包不需要签名就可以正常使用。

刷机之前首先应执行下面的命令进入bootloader模式。

```
adb reboot-bootloader
```

大概3到5秒时间，Nexus 7就会自动重启，并进入bootloader模式。要注意的是，在bootloader模式下adb命令不再起作用，取而代之的是fastboot命令，要想知道当前PC有多少Android设备处于bootloader模式，可以使用下面的命令。当然，如果Android设备处于正常模式下，可以使用adb devices显示当前连接到PC的Android设备列表。

```
fastboot devices
```

进入到bootloader模式后就可以随心所欲地刷机了。在本文只介绍如何刷Recovery，后续的文章会深入讲解如何刷各种镜像文件。假设下载的Recovery镜像文件是recovery-clockwork-touch-6.0.2.3-grouper.img。在bootloader模式下刷Recovery镜像文件的命令如下：

```
fastboot flash recovery recovery-clockwork-touch-6.0.2.3-grouper.img
```

如果刷机成功，会显示如图1所示的信息。

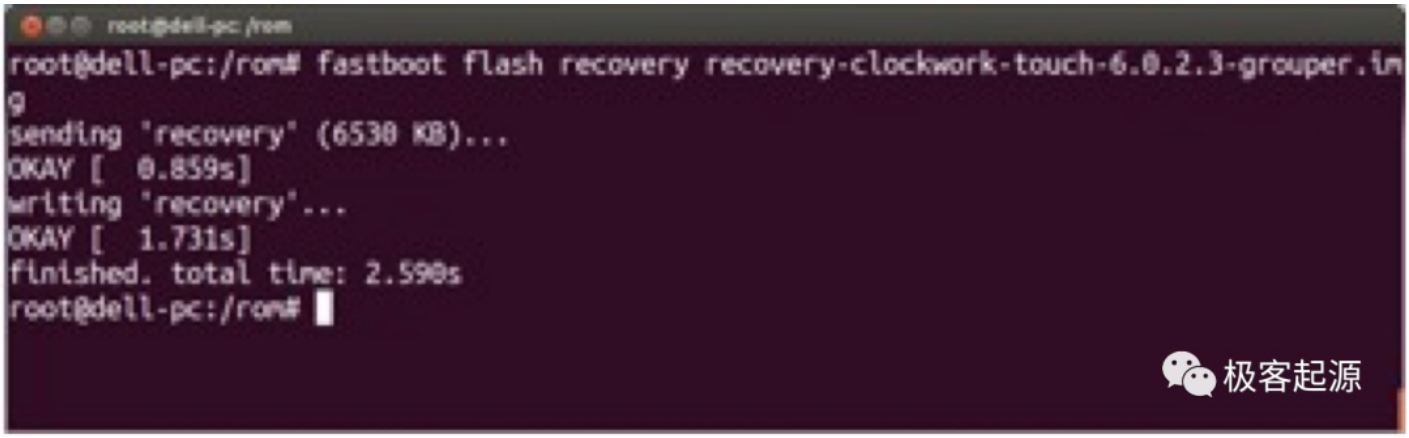

图1 成功刷Recovery

在bootloader模式下通过音量上下键切换到“Recovery mode”选择项（在屏幕右上角显示），然后按电源键，大概等5秒，Nexus 7就会进入Clockwork Recovery。如果想重新启动Nexus 7，并进入正常的模式，可以选择Recovery的“reboot system now”菜单项，然后按电源键即可。如果读者刷的是带触摸功能的Recovery，只要点击“reboot system now”菜单项即可重新启动Nexus 7。如果读者目前处于正常模式下，并且Nexus 7已通过USB线与PC连接，可以使用下面的命令进入Recovery模式。

```
adb reboot recovery
```

3. su命令源代码分析

刷完了Recovery后，就需要将su文件放到Android设备中的/system/bin或/system/xbin目录中，然后直接执行su命令即可使当前的Shell获得root权限（Shell提示符从$变成了#），以前很多不能做的事也可以做了，例如，普通用户不能查看/data/data目录中的内容，使用su命令提取root权限后也可以使用ls命令查看/data/data目录的内容了。

读者可以从网上下载合适的su文件，或直接从Android源代码中获取su文件。如果Android源代码还没有编译，需要按着1.3.2节的步骤编译整个Android源代码。成功编译Android源代码后，就可以在如下的目录找到编译好的su文件。

<Android源代码本目录>/out/target/product/generic/system/xbin

实际上这个su命令完全可以满足目前的需求，也就是提取Android设备当前Shell的root权限。不过先别忙将su文件弄到Android设备上。接下来先看一下su文件的源代码，了解一下su文件的运行原理以及为什么能在Android设备上成功执行。

读者可以从如下的目录找到su命令的源代码。

<Android源代码根目录>/system/extras/su

su是用C语言编写的普通可执行文件，主文件是su.c。读者可以打开该文件看一下su的源代码。

su.c文件中除了引用的一些头文件外，就只有一个main函数，代码如下：

源代码文件：<Android源代码根目录>/system/extras/su/su.c

```
#define LOG_TAG "su"
… …
/*  此处省略了#include … 语句  */
int main(int argc, char **argv)
{
    struct passwd *pw;
    int uid, gid, myuid;
    /* 获取用户ID，只有root和当前的Shell能执行su命令 */
    myuid = getuid();
    if (myuid != AID_ROOT && myuid != AID_SHELL) {
        fprintf(stderr,"su: uid %d not allowed to su\n", myuid);
        return 1;
    }

    if(argc < 2) {
        uid = gid = 0;
} else {
    /*  根据参数指定的用户名获取用户属性，如果getpwnam函数返回0，表示参数指定的是用户ID
         而不是用户名
    */
        pw = getpwnam(argv[1]);

        if(pw == 0) {
            uid = gid = atoi(argv[1]);
        } else {
            uid = pw->pw_uid;
            gid = pw->pw_gid;
        }
    }
/*
setgid函数要设置一个用户组ID，使用属于该用户组的用户执行任何文件时都拥有该文件所有者
的权限。setuid函数与setgid函数类似，需要设置一个用户ID。使用该用户执行任何可执行文件都
会拥有该文件所有者的权限。例如，sh命令的所有者是root用户，而当前登录用户是user，这时
使用setuid函数设置user的ID后，再执行sh命令，就相当于以root用户的身份执行sh命令，
所以进入新的Shell后就会拥有root权限
*/
    if(setgid(gid) || setuid(uid)) {
        fprintf(stderr,"su: permission denied\n");
        return 1;
    }

    /* 执行通过命令行参数指定的命令 */
    if (argc == 3 ) {
        if (execlp(argv[2], argv[2], NULL) < 0) {
            fprintf(stderr, "su: exec failed for %s Error:%s\n", argv[2],
```

```
                    strerror(errno));
            return -errno;
        }
    } else if (argc > 3) {
        /* Copy the rest of the args from main. */
        char *exec_args[argc - 1];
        memset(exec_args, 0, sizeof(exec_args));
        memcpy(exec_args, &argv[2], sizeof(exec_args));
        if (execvp(argv[2], exec_args) < 0) {
            fprintf(stderr, "su: exec failed for %s Error:%s\n", argv[2],
                    strerror(errno));
            return -errno;
        }
    }

/* 执行sh命令进入新的Shell，如果成功执行，当前程序会立刻退出，如果执行失败，会继续
   执行下面的语句
 */
    execlp("/system/bin/sh", "sh", NULL);

    fprintf(stderr, "su: exec failed\n");
    return 1;
}
```

从su.c文件的代码可以看出，su命令支持多个命令行参数。这些命令行参数分为如下两类。

第1类：su的第一个参数，该参数指定了要提升权限的用户ID或用户名，如果不指定，就是当前的用户。

第2类：其余的参数。表示提升权限后要立刻执行的命令和该命令的参数。

下面都是合法的su命令调用形式。

su

su user

su user ls –al /data/data/

su提升权限的核心有如下两个。

> 通过setgid和setuid函数提升权限，也就是使得任何用户在执行sh命令时都会拥有与sh命令拥有者同样的权限。由于sh命令的拥有者是root用户，所以自然就将新的Shell提升到了root权限。
>
> 通过execlp函数执行sh命令。由于前面已经调用了setgid和setuid函数，所以执行sh命令会进入新的Shell，并且该Shell与sh命令文件的所有者（root用户）拥有同样的权限。执行exit命令会退出拥有root权限的Shell，并重新回到原来没有root权限的Shell。再次执行exit命令后，就会退出Android Shell，回到Ubuntu Linux的终端。

4. 制作第一个Recovery刷机包（编写脚本文件）

Recovery使用的刷机包就是zip格式的压缩文件。根据不同的需求，刷机包中包含的文件不同，一个完整的刷机包非常复杂，不过本节的目的只是将su文件复制到/system/xbin目录中，所以暂时用不着那么复杂的刷机包。关于Recovery刷机包的详细制作过程将在后面跟的章节深入探讨。本文只做最基本的刷机包。

其实要做一个Recovery刷机包并不困难。制作Recovery刷机包之前通常要考虑使用的是哪个Recovery。例如，本书主要使用了Clockwork Recovery，所以可以利用Clockwork Recovery中的一些特性。

本文要制作的刷机包中只有两个目录：system和META-INF。其中system就是编译Android源代码后，在<Android源代码根目录>/out/target/product/generic目录中的system目录，也是进入Android设备的Shell后看到的“/system”目录，在该目录下包含了Android的系统文件，其中包括很多命令文件以及系统应用程序。不过本文制作的Recovery刷机包没这么复杂。由于只需要将su文件复制到/system/xbin目录，所以在system目录中只要有一个xbin子目录，并且在该目录中放一个在上一节获得的su文件即可。

可能很多读者会问，将su文件放到/system/xbin目录中，Recovery中刷机时就会将su文件复制到Android系统的/system/xbin目录中吗？答案很简单，Recovery当然不知道自己要做什么，具体要完成什么工作，如何来完成，玄机全在META-INF目录中。

在META-INF/com/google/android目录中有一个updater-script脚本文件（纯文本文件）和一个update-binary可执行文件。别看这两个文件一共不到200KB，它们却是整个Recovery刷机包中最核心的部分。尤其是update-binary，该文件通常在190KB上下，别看文件尺寸不大，这可是内嵌于Recovery的一种轻量级脚本语言的解析器，而updater-script脚本文件就是使用这种脚本语言写的。这种脚本语言就是edify。该语言除了定义一些简单的语句外，还定义了几十个用于各种操作的函数，例如，复制文件、删除文件、建立链接等。

Edify语言会在后面的章节介绍，在本文只介绍该语言的几个常用的函数。这些函数的原型、含义及其用法如下：

ui_print

原型：ui_print(msg1, ..., msgN);

含义：该函数用于在Recovery界面输出字符串。其中msg1、…、msgN表示N个字符串参数，该函数至少需要指定一个参数。如果指定多个参数，会将这些参数值连接起来输出。

用法：ui_print(" hello world ");

run_program

原型：run_program(prog, arg1, .., argN);

含义：该函数用于执行程序，其中prog参数表示要执行的程序文件（要写完整路径），arg1、…、argN表示要执行程序的参数。prog参数是必须的，其他参数都是可选的。

用法：run_program("/sbin/busybox","mount", "/system");

delete

原型：delete(file1, file2, ..., fileN);

含义：该函数用于删除一个或多个文件。其中file1、file2、…、fileN表示要删除文件的路径，至少需要指定一个文件。

用法：delete("/system/xbin/su");

package_extract_dir

原型：package_extract_dir(package_path, destination_path);

含义：用于提取刷机包中package_path指定目录的所有文件到destination_path指定的目录。其中package_path参数表示刷机包中的目录，destination_path参数表示目标目录。

用法：package_extract_dir("system", "/system");

set_perm

原型：set_perm(uid, gid, mode, file1, file2, ..., fileN);

含义：用于设置一个或多个文件的权限。其中uid参数表示用户ID，gid参数表示用户组ID。如果想让文件的用户和用户组都是root，uid和gid需要都为0。mode参数表示设置的权限，这个权限与chmod命令设置的权限完全一样，例如，如果将一个文件设为任何用户都可以读写和执行的权限值是0777。file1、file2、…、fileN表示要设置权限的文件的路径。

用法：set_perm(0, 0, 0777, "/system/xbin/su");

unmount

原型：unmount(mount_point);

含义：用于解除文件系统的挂载。其中mount_point参数表示文件系统。

用法：unmount("/system");

本文要编写的updater-script文件只会使用上面的函数。该脚本文件主要实现如下基本功能。

以读写模式挂载/system。

删除旧的su文件。

复制新的su文件。

修改su文件的权限。

卸载/system。

其中挂载/system调用了busybox命令，该命令并不属于Android。不过该命令十分强大，常被人称为Android的瑞士军刀。busybox是一个开源的命令集合，将多达上百个命令集成在了一个大概2MB的文件中。例如，本文要用的mount命令就是其中之一。尽管Android从本质上也属于Linux系统，但较其他Linux系统集成的命令是很少的，所以如果想在Android中执行各种操作，通常就需要将busybox文件复制到Android系统的/system/xbin或其他存储命令文件的系统目录。这样只需要一个busybox命令就可以搞定一切。如果读者想知道busybox到底支持多少命令，只需要直接执行busybox命令即可。

读者可以在http://www.busybox.net下载busybox最新版本的源代码，并按着说明使用交叉编译器编译busybox即可（在ARM架构的设备上运行必须要使用交叉编译器），为了方便读者，在随书光盘中带了一个编译好的busybox文件，路径是/tools/busybox。该文件只能在ARM架构的设备上运行，不能在X86 PC上使用。读者需要将busybox文件上传到Android设备的/system/xbin目录中（需要root权限），并执行 chmod 777 /system/xbin/busybox命令修改其权限后即可执行该命令。

现在看一下updater-script文件的代码。

```
ui_print("*********************");
ui_print("My First Recovery Update");
ui_print("*********************");

ui_print("----Mounting /system ----");
#  以读写模式挂载/system
run_program("/sbin/busybox", "mount","-o", "rw", "/system");

ui_print("----Delete /system/xbin/su ----");
#  删除旧的su文件
delete("/system/xbin/su");
ui_print("- Extracting files");
#  将刷机包中system目录的所有文件复制到/system目录中的相应位置
package_extract_dir("system", "/system");
ui_print("----- Setting permissions");
#  设置ui命令为任何用户都可执行
set_perm(0, 0, 0777, "/system/xbin/su");
#  卸载/system
unmount("/system");
ui_print("finished");
```

在updater-script文件中使用run_program函数调用了busybox命令，并通过该命令执行了mount操作，实际上，相当于在Linux终端执行如下的命令。

```
busybox mount –o rw /system
```

如果执行mount操作时未指定“-o rw”，默认是只读挂载，也就是说/system目录及其子目录是只读的，为了将/system目录变成读写模式的，需要再次执行如下的命令。

```
busybox mount –o rw, remount /system
```

其中“-o rw, remount”中的rw和remount是mount命令的两个选项，表示重新将/system目录mount成可读写的。

如果需要修改默认的挂载点对应的路径，例如，将/system挂载到/my_system目录（该目录必须存在），需要使用下面的命令。

```
busybox mount –o rw /system  /my_system
```

现在updater-script脚本文件已经编写完了，接下来还需要一个用于解析updater-script脚本文件的update-binary程序。读者可以从网上找一个Recovery刷机包，将其中的update-binary文件放到自己的刷机包中，或到<Android源代码根目录> /out/target/product/generic/system/bin目录寻找一个updater文件，将该文件改名为update-binary即可。其实updater文件也是Android源代码的某个子程序编译生成的，在后面深入探讨Recovery时再详细分析updater的实现原理。

现在Recovery刷机包的所有文件（su、updater-script和update-binary）都搞定了，接下来完成最后一步，就是将system和META-INF目录用zip格式压缩（压缩文件名任意取）。在下一节会介绍如何将这个zip格式文件中的内容刷到Nexus 7上。为了方便读者，在随书光盘上已经带了这个zip压缩文件（/recovery/su_update.zip），读者可以直接将该文件刷到Android设备上（除了Nexus 7外，其他使用Clockwork Recovery的Android设备也可以使用该刷机包）。

可能有的读者会发现，updater-script脚本中执行了/sbin目录中的busybox命令，但成功启动Android后进入Shell，在/sbin目录中并没有发现busybox命令，这到底是真么回事呢？

实际上，busybox命令的确存在，但却不在system文件系统里，而是在recovery文件系统中。在Android正常启动后，实际上挂载的是system文件系统。而进入Recovery模式后，系统会自动挂载recovery文件系统，而挂载system文件系统要在updater-script脚本文件中通过相应的函数来完成（如本文使用了run_program函数调用了busybox命令挂载system文件系统）。system和recovery文件系统都有一个sbin目录，但目录中的文件是不一样的。实际上，通过Android源代码编译生成的Recovery镜像是不包含busybox命令的，而Clockwork Recovery的busybox命令是自己添加的，也就是说，其他的Recovery镜像并不一定包含busybox命令，所以本文编写的updater-script脚本文件只适合Clockwork Recovery。如果要想使用其他的Recovery，或者将busybox命令添加到Recovery镜像中，或者使用edify脚本语言的mount函数挂载相应的文件系统（如/system、/data等）。在后面的章节会详细介绍Recovery镜像文件的生成，以及如何修改现成的Recovery镜像。

5. 首次通过DIY方式提取ROOT权限

到现在为止，一切准备工作都已经完成了，现在只剩下最后一步，就是提取Nexus 7的root权限。不过这次提取root权限完全是DIY方式的，这也显得更有意义，因为作为程序员来说，了解其中的过程比结果更重要。

现在需要用USB线连接Nexus 7和PC，然后将上一节生成的zip文件上传到Nexus 7的SD卡中。如果不想上传也没问题。通常在Recovery模式下选择sideloader上传方式，而不是事先上传到Nexus 7上，因为这样Nexus 7不需要在正常模式和Recovery模式之间来回切换，这也更适合需要频繁调试和刷机的程序员。

现在进入Nexus 7的Recovery模式（正常模式下执行adb reboot recovery命令），如果读者已经将zip压缩文件（这里假设为update.zip）上传到Nexus 7的SD卡中，选择“install zip from sdcard”（第2个菜单项），然后继续选择zip文件即可刷机。如果未将zip文件上传，选择“install zip from sideload”菜单项，然后Recovery模式会处于等待状态。现在adb的sideload上传模式可以工作了。在Linux终端输入adb sideload update.zip命令，即可成功刷机。在刷机的过程中会看到updater-script脚本文件中使用ui_print函数输出的字符串。updater-script脚本文件中的命令执行完后，就会回到Recovery主界面。然后选择“reboot system now”（第1个菜单项）重启Nexus 7。

当Nexus 7进入正常模式后，进入Shell，这时还没有执行su命令，所以当前Shell仍然没有root权限，现在执行su命令，会看到Shell提示符从“$”变为“#”，这说明当前Shell已经拥有了root权限。现在做一些以前做不了的事，例如，使用ls /data/data命令查看系统的data目录，现在可以成功列出该目录中所有的子目录（都是系统应用或普通应用建立的私有目录）。

6. 上传Android应用到/system/app目录

尽管提取root权限的目的很多，有的是为了调用Linux的命令，有的是为了直接访问系统的数据。不过本章提取root权限的目的主要是为了将系统应用的APK文件上传到/system/app目录中，以便可以在真机的环境下测试系统应用。

现在进入Android设备的Shell，并执行su命令提取root权限，然后看看/system/app目录是否可写。有的Android设备在挂载system文件系统时，/system及其子目录是只读的，如果是这种情况，执行如下的命令即可将/system目录及其子目录变成可读写的。

```
mount -o rw,remount /system
```

现在退出Android设备的Shell，重新回到Linux终端。然后找一个APK文件。但有一个问题，当执行adb shell命令进入Android设备的Shell时，一开始并没有root权限，需要执行su命令才能提权，所以就不能直接使用adb push命令将APK文件上传到/system/app目录中（因为没有root权限，该目录是只读的）。解决的方法也很简单，就是首先使用adb push命令将APK文件上传到Android设备的SD卡上，然后在执行adb shell的同时执行su命令提权。例如，下面的命令可以在Linux终端下删除Android设备中/system/app目录中的Test.apk文件。

```
adb shell su -c "mount -o remount,rw /system | rm -f /system/app/Test.apk "
```

下面的命令将SD卡根目录中的Test.apk文件复制到Android设备的/system/app目录中。

```
adb su -c "mount -o remount,rw /system | cp /sdcard/Test.apk /system/app/Test.apk "
```

为了方便，读者可以编写一个带参数的Shell脚本文件，将adb push和adb shell命令在一起使用。

- EOF -

推荐阅读 点击标题可跳转

1、连Python产生器（Generator）的原理都解释不了，还敢说Python用了5年？

2、牛掰了！鸿蒙与Android完美融合，将鸿蒙设备当Android设备用

3、【鸿蒙学院】鸿蒙App开发直播学员提问与回答

4、【鸿蒙学院】鸿蒙IDE：下载、安装DevEco Studio

5、 Python高效编程之88条军规（2）：你真的会格式化字符串吗？

关注「极客起源」公众号，加星标，不错过精彩技术干货