详解shell中>/dev/null 2>&1到底是什么

主要介绍了shell中>/dev/null 2>&1到底是什么，文中介绍的很详细，需要的朋友可以参考借鉴，下面来一起看看吧。

前言

相信大家经常能在shell脚本中发现>/dev/null 2>&1这样的语句。以前的我并没有去深入地理解这段命令的作用，照搬照用，直到上周我将这段命令不小心写成了2>&1 >/dev/null，出了一点小问题之后，我才开始去了解这段命令背后的“玄机”。

shell重定向介绍

就像我们平时写的程序一样，一段程序会处理外部的输入，然后将运算结果输出到指定的位置。在交互式的程序中，输入来自用户的键盘和鼠标，结果输出到用户的屏幕，甚至播放设备中。而对于某些后台运行的程序，输入可能来自于外部的一些文件，运算的结果通常又写到其他的文件中。而且程序在运行的过程中，会有一些关键性的信息，比如异常堆栈，外部接口调用情况等，这些都会统统写到日志文件里。

shell脚本也一样，但是我们一般在使用shell命令的时候，更多地还是通过键盘输入，然后在屏幕上查看命令的执行结果。如果某些情况下，我们需要将shell命令的执行结果存储到文件中，那么我们就需要使用输入输出的重定向功能。

文件描述符

当执行shell命令时，会默认打开3个文件，每个文件有对应的文件描述符来方便我们使用：

类型	文件描述符	默认情况	对应文件句柄位置
标准输入（standard input）	0	从键盘获得输入	/proc/self/fd/0
标准输出（standard output）	1	输出到屏幕（即控制台）	/proc/self/fd/1
错误输出（error output）	2	输出到屏幕（即控制台）	/proc/self/fd/2

所以我们平时在执行shell命令中，都默认是从键盘获得输入，并且将结果输出到控制台上。但是我们可以通过更改文件描述符默认的指向，从而实现输入输出的重定向。比如我们将1指向文件，那么标准的输出就会输出到文件中。

输出重定向

输出重定向的使用方式很简单，基本的一些命令如下：

命令	介绍
command >filename	把标准输出重定向到新文件中
command 1>filename	同上
command >>filename	把标准输出追加到文件中
command 1>>filename	同上
command 2>filename	把标准错误重定向到新文件中
command 2>>filename	把标准错误追加到新文件中

我们使用>或者>>对输出进行重定向。符号的左边表示文件描述符，如果没有的话表示1，也就是标准输出，符号的右边可以是一个文件，也可以是一个输出设备。当使用>时，会判断右边的文件存不存在，如果存在的话就先删除，然后创建一个新的文件，不存在的话则直接创建。但是当使用>>进行追加时，则不会删除原来已经存在的文件。

为了更好地理解输出重定向，感受重定向的“魅力”，我们看一下以下的例子：我们创建一个测试目录，目录下面仅有一个a.txt文件。

```
# tree
.
└── a.txt
0 directories, 1 file
# ls a.txt b.txt
ls: 无法访问b.txt: 没有那个文件或目录
a.txt
```

在我们执行ls a.txt b.txt之后，一共有两种输出，其中ls: 无法访问b.txt: 没有那个文件或目录是错误输出，a.txt是标准输出。

```
# ls a.txt b.txt 1>out
ls: 无法访问b.txt: 没有那个文件或目录
# cat out
a.txt
# ls a.txt b.txt >>out
ls: 无法访问b.txt: 没有那个文件或目录
# cat out
```

```
a.txt
a.txt
```

在上述命令中，我们将原来的标准输出重定向到了out文件中，所以控制台只剩下了错误提示。并且当执行了追加操作时，out文件的内容非但没有被清空，反而又多了一条a.txt。

同理，我们也可以将错误输出重定向到文件中：

```
# ls a.txt b.txt 2>err
a.txt
# cat err
ls: 无法访问b.txt: 没有那个文件或目录
# ls a.txt b.txt >out 2>err
# cat out
a.txt
# cat err
ls: 无法访问b.txt: 没有那个文件或目录
```

看到这里，朋友们可能会发现>out 2>err和我们在一开头提到的>/dev/null 2>&1已经很像了，别急，这待会再说。

输入重定向

在理解了输出重定向之后，理解输入重定向就会容易得多。对输入重定向的基本命令如下：

命令	介绍
command <filename	以filename文件作为标准输入
command 0<filename	同上
command <<delimiter	从标准输入中读入，直到遇到delimiter分隔符

我们使用<对输入做重定向，如果符号左边没有写值，那么默认就是0。

我们这次以cat命令为例，如果cat后面没有跟文件名的话，那它的作用就是将标准输入（比如键盘）回显到标准输出（比如屏幕）上：

```
# cat
123
123
test
test
```

我们可以将利用输入重定向，将我们在键盘上敲入的字符写入到文件中。我们需要使用ctrl+c来结束输入：

```
# cat >out
123
test
^C
# cat out
123
test
```

好了，此时我们觉得自己在键盘上敲比较累，还是直接让cat读取一个文件吧。那么我们需要利用输入重定向：

```
# cat input
aaa
111
# cat >out <input
# cat out
aaa
111
```

神奇的事情发生了，out文件里面的内容被替换成了input文件里的内容。那么<<又是什么作用呢？我们再看：

```
# cat >out <<end
> 123
> test
> end
# cat out
123
test
```

我们看到，当我们输入完cat >out <<end，然后敲下回车之后，命令并没有结束，此时cat命令像一开始一样，等待你给它输入数据。然后当我们敲入end之后，cat命令就结束了。end之前输入的字符都已经被写入到了out文件中。这就是输入分割符的作用。

高级用法

重定向绑定

好了，在有了以上知识的基础上，我们再来看开头提到的>/dev/null 2>&1。这条命令其实分为两命令，一个是>/dev/null，另一个是2>&1。

1. >/dev/null

这条命令的作用是将标准输出1重定向到/dev/null中。 /dev/null代表linux的空设备文件，所有往这个文件里面写入的内容都会丢失，俗称"黑洞"。那么执行了>/dev/null之后，标准输出就会不再存在，没有任何地方能够找到输出的内容。

2. 2>&1

这条命令用到了重定向绑定，采用&可以将两个输出绑定在一起。这条命令的作用是错误输出将和标准输出同用一个文件描述符，说人话就是错误输出将会和标准输出输出到同一个地方。

linux在执行shell命令之前，就会确定好所有的输入输出位置，并且从左到右依次执行重定向的命令，所以>/dev/null 2>&1的作用就是让标准输出重定向到/dev/null中（丢弃标准输出），然后错误输出由于重用了标准输出的描述符，所以错误输出也被定向到了/dev/null中，错误输出同样也被丢弃了。执行了这条命令之后，该条shell命令将不会输出任何信息到控制台，也不会有任何信息输出到文件中。

>/dev/null 2>&1 VS 2>&1 >/dev/null

再回到文章的开头，我说我弄反了>/dev/null和2>&1拼装的顺序，导致出了一点小问题。乍眼看这两条命令貌似是等同的，但其实大为不同。刚才提到了，linux在执行shell命令之前，就会确定好所有的输入输出位置，并且从左到右依次执行重定向的命令。那么我们同样从左到右地来分析2>&1 >/dev/null：

2>&1，将错误输出绑定到标准输出上。由于此时的标准输出是默认值，也就是输出到屏幕，所以错误输出会输出到屏幕。
>/dev/null，将标准输出1重定向到/dev/null中。

我们用一个表格来更好地说明这两条命令的区别：

命令	标准输出	错误输出
>/dev/null 2>&1	丢弃	丢弃
2>&1 >/dev/null	丢弃	屏幕

>/dev/null 2>&1 VS >/dev/null 2>/dev/null

那么可能会有些同学会疑问，为什么要用重定向绑定，而不是像>/dev/null 2>/dev/null这样子重复一遍呢。

为了回答这个问题，我们回到刚才介绍输出重定向的场景。我们尝试将标准输出和错误输出都定向到out文件中：

```
# ls a.txt b.txt >out 2>out
# cat out
a.txt
无法访问b.txt: 没有那个文件或目录
```

WTF？竟然出现了乱码，这是为啥呢？这是因为采用这种写法，标准输出和错误输出会抢占往out文件的管道，所以可能会导致输出内容的时候出现缺失、覆盖等情况。现在是出现了乱码，有时候也有可能出现只有error信息或者只有正常信息的情况。不管怎么说，采用这种写法，最后的情况是无法预估的。

而且，由于out文件被打开了两次，两个文件描述符会抢占性的往文件中输出内容，所以整体IO效率不如>/dev/null 2>&1来得高。

nohup结合

我们经常使用nohup command &命令形式来启动一些后台程序，比如一些java服务：

```
# nohup java -jar xxxx.jar &
```

为了不让一些执行信息输出到前台（控制台），我们还会加上刚才提到的>/dev/null 2>&1命令来丢弃所有的输出：

```
# nohup java -jar xxxx.jar >/dev/null 2>&1 &
```

总结

本文主要介绍了linux重定向的原理以及一些基本命令，并且详细地分析了>/dev/null 2>&1这个命令以及一些注意点。

总而言之，在工作中用到最多的就是nohup command >/dev/null 2>&1 &命令，希望大家能够好好掌握。

好了，以上就是这篇文章的全部内容了，希望本文的内容对大家的学习或者工作能带来一定的帮助，如果有疑问大家可以留言交流。