重定向中的秘密（句柄备份）

引用：

> 看了 [小竹英雄] 的 “ 重定向中的秘密 ” 一文，搞得我云里雾里，真就是那饭桌上的美食转哪转——晕菜了。
>
> 所以写此文与各位看看，有什么不对的地方请大家指正。

重定向是指使用重定向符号对输入或输出位置进行重新指定。

引用：

> 以下是理论说明，懂的可以跳过，也可以先看例子，遇到不明白的再倒回来看。
>
> 重定向符号有 <、>、>>、<&、>& 和 | 共六个。
>
> 重定向符号的作用如下表所示：
>
> 符号 作用
>
> ——————————————————————————
>
> < 从文件或设备（如默认的 con 键盘）中读取命令输入。
>
> > 将命令输出写入到文件或设备（如 prn 打印机）中。
>
> >> 将命令输出添加到文件末尾而不删除文件中的信息。
>
> <& 从后一个句柄读取输入并写入到前一个句柄输出中。
>
> >& 将前一个句柄的输出写成后一个句柄的输入。
>
> | 读取前一个命令中的输出作为后一个命令的输入。也称作管道符。
>
> ——————————————————————————
>
> 注意：句柄的输入输出只是其指向，真正的输入输出是靠设备。
>
> 重定向符号的默认句柄，<的默认句柄是 0，>的默认句柄是 1。
>
> 可用句柄（0 - 9）如下表：
>
> 句柄 句柄号 说明
>
> ——————————————————————————
>
> STDIN 0 标准输入，默认从键盘输入
>
> STDOUT 1 标准输出，默认输出到命令提示符窗口
>
> STDERR 2 标准错误输出，默认输出到命令提示符窗口
>
> UNDEFINED 3-9 这些句柄由应用程序和各个具体工具单独定义。
>
> ——————————————————————————

STD 即 Standard —— 标准的缩写。

设备

标准输入设备为键盘，用 con 表示。

标准输出设备为控制台（即命令提示符），默认也用 con 表示。

常用的还有：

空设备，用 nul 表示。

存储设备，文件就归到这类。

不常用的如：打印机 prn。

指向

句柄实际上也是一组 0 1 数据，是存储了一个指向。

句柄 0 默认指向 con，这里 con 为标准输入设备，即键盘。

句柄 1 默认指向 con，这里 con 为标准输出设备，即控制台。

句柄 2 默认也指向 con，同 1。

此外，我们对句柄的指向重新设定（即所谓的重定向）时，可以改变句柄的指向。

如 1>nul 是让“标准输出”数据输出到空设备，效果就是屏蔽输出。

特别之处：3 - 9 默认是没有指向的，我们可以用（空）来表示，注意是不同于空设备 nul 的哦！

（图示）

下面拿个简单的例子来说明

pause>nul

后面的 >nul 的作用就是屏蔽了 pause 命令的默认输出“请按任意键继续．．．”。

以上是简单的描述，而实际的过程可没那么简单：

首先确定重定向符号为 >；

接着检查句柄号，发现没有，控制台为其加上默认句柄 1，此时变成 pause 1>nul；

由于句柄 1 的默认指向 con，此时要被临时设定为指向 nul，为了之后取回原来的指向，所以要先对 1 的指向进行备份，备份到句柄 3，备份过后临时设定 1 指向 nul；

（为什么要备份，为什么要备份到 3 而不是 4? 见下文）

到这时这条语句才被执行，效果就是批处理暂停，没有输出只有一个光标闪阿闪。

执行完后 1 要取回原来的指向，1 的指向在上面被备份到 3，不管 3 的指向有没有改变 1 要取回它的指向来，取回后 1 指向 con；3 的指向被取走，由于 3 没有备份，所以恢复到原来的（空），即无指向。

至此整个过程描述完毕。

可以发现这时的情况与最初是相同的，所有句柄的指向都是其默认值。

接着引出 句柄（指向）备份理论。

复制内容到剪贴板

代码：

当指向为非空的句柄被重新设置时要进行备份，而且是备份到第一个没有指向的句柄。

语句运行完后要取回其备份，而不管备份中的句柄指向有没有改变。

如果当前状态下 3 - 9 这七个句柄都有了指向，这时新设置的句柄其原指向将不再备份。

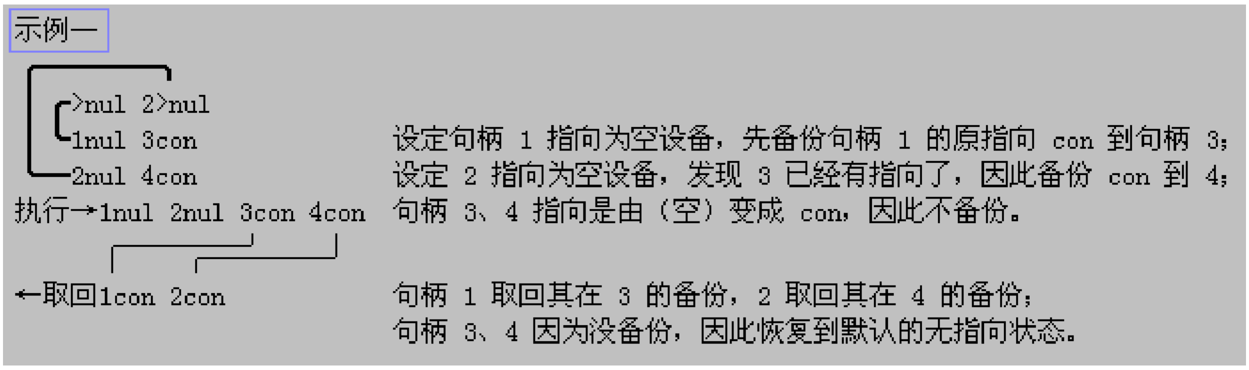
复制内容到剪贴板

代码：

另一种表述：也就是说当指向为（空）的句柄被设定时是不需要备份的。如：复制句柄时源句柄不备份，因为没有被再次设置。

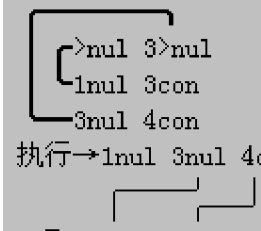
补充说明：一个句柄只能存储一个指向。

举例说明：



怎么一次性屏蔽所有输出呢?

示例二

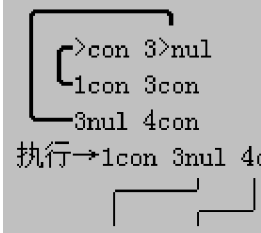


```
>nul 3>nul
1nul 3con          设定句柄 1 指向为空设备，先备份句柄 1 的原指向 con 到句柄 3；
3nul 4con          设定 3 指向为空设备，发现 3 已经有指向了，因此备份其 con 到 4；
执行→1nul 3nul 4con  句柄 4 指向是由（空）变成 con，因此不备份。
```



```
←取回1nul 3con       句柄 1 取回其在 3 的备份，3 取回其在 4 的备份；
                     句柄 4 因为没备份，因此恢复到默认的无指向状态。
```

示例三



```
>con 3>nul         虽然设定值与原值相同，但只要再次设定了，不管同不同都要备份。
1con 3con          设定句柄 1 指向为 con，先备份句柄 1 的原指向 con 到句柄 3；
3nul 4con          设定 3 指向为空设备，发现 3 已经有指向了，因此备份其 con 到 4；
执行→1con 3nul 4con  句柄 4 指向是由（空）变成 con，因此不备份。
```



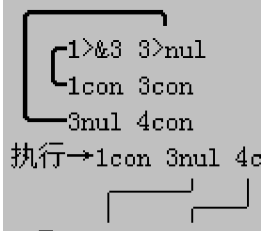
```
←取回1nul 3con       句柄 1 取回其在 3 的备份，3 取回其在 4 的备份；
                     句柄 4 因为没备份，因此恢复到默认的无指向状态。
```

示例三与示例二有什么异同呢？

基本上相同，只有执行的这句，示例三是有输出的，示例二却没有。

下面讲一个句柄复制的

示例四



```
1>&3 3>nul
1con 3con          设定句柄 1 指向为句柄 3 的指向，先备份句柄 1 的原指向 con 到句柄 3；
3nul 4con          设定 3 指向为空设备，发现 3 已经有指向了，因此备份其 con 到 4；
执行→1con 3nul 4con  句柄 4 指向是由（空）变成 con，因此不备份。
```



```
←取回1nul 3con       句柄 1 取回其在 3 的备份，3 取回其在 4 的备份；
                     句柄 4 因为没备份，因此恢复到默认的无指向状态。

设定句柄 1 指向为句柄 3 的指向，是不是相当于复制句柄 3 的指向给 1。
这也是为什么在复制时源句柄 3 不用备份的原因。
```

有没有发现示例四和示例三是一样的，那用复制句柄的方法能不能和示例二一样呢？

可以，自己想吧。 ^_^

还有一个特别的例子：

```
echo 你好！ >nul >hello.txt >prn >con
```

它使用的是相同的句柄 1，这时要以最后一个为准，其他的忽略。把它存为批处理运行，能够清楚的看到前面的三种指向直接被剪掉了。

同理：

echo 你好！ 1<&3 >nul

也以后面的 ”>nul“ 为准，即不显示 ”你好！“。

具体例子讲解

复制内容到剪贴板

代码：

```
@echo off
echo 哇，这就是传说中的树叶吗？>con 3>nul
echo 我有一片神奇的树叶（吹口哨），
echo 你看不见我，
echo 我现身，>con
echo 我又隐身啦。
echo 谁偷了我的叶子，快还我。>con 4>con
echo 没穿裤裤被你们看到了！
pause>nul
```

哇，这就是传说中的树叶吗？

这句用了 “ >con >nul” 同示例三，首先 句柄 1 指向要被设为 con，先备份到 3 再设定；第二步 这时 3 指向为 con，现在又要设为 nul，先备份 con 到 4；最后 1 指向 con，3 指向 nul，4 指向 con；所以这句执行时是要显示的。

执行完后返回，1 取回 3 的指向 nul，3 取回 4 的指向 con，4 恢复（空），所以返回结束时 1 指向 nul，3 指向 con。假设这时状态为 A。

因此接下来两句 “ 我有一片神奇的树叶（吹口哨），”，“ 你看不见我，” 是不显示的。“ 我现身，” 这句设定为 “ >con”。执行时 1 为 con，原来的 nul 备份到 4（因为前面 3 已经为 con 不是没有指向了）；执行完后 1 取回备份 nul，4 恢复（空）。这时的状态还是 A。

所以下一句 “ 我又隐身啦。” 还是不显示。

倒数第二句 “ 谁偷了我的叶子，快还我。 ”

设定为 “ >con 4>con”，首先 句柄 1 指向要被设为 con，3 已经不为（空）了，所以备份 nul 到 4；第二步 这时 4 指向为 nul，现在又要设为 con，先备份 nul 到 5；最后 1 指向 con（3 指向 nul）4 指向 con，5 指向 nul；所以这句显示。

返回时 1 取回 4 的指向 con，4 取回 5 的指向 nul，5 恢复（空）。而这时状态已经不

同于 A 了，设为状态 B，最后一句 “没穿裤裤被你们看到了！” 使用的是状态 B 的句柄指向，所以也显示。

到些讲解结束。（以上红色为显示的，淡蓝色为不显示的）

如果要用处理来表示就是这样

```
pause>nul                    例一
set 1=con
set 2=con                    载入默认值作为当前设定
set 3=%1%                    发现要设定 1 了，先备份到 3
  setlocal
  set 1=nul                  临时设定 1
  echo %1%
  endlocal
set 1=%3%                    取回 1 在 3 的备份
set 3=                       恢复（空）

echo a>con 3>nul             例二
set 1=con
set 2=con                    载入默认值作为当前设定
set 3=%1%                    发现要设定 1 了，先备份到 3
  setlocal
  set 1=con                  临时设定 1，虽然新设定会值与载入的当前设定值一样
  echo %1%
  endlocal
set 4=%3%                    发现 3 有指向且又要重新设定了，先备份到 4
set 3=nul                    临时设定 3
set 1=%3%                    取回 1 在 3 的备份
set 3=%4%                    取回 3 在 4 的备份
set 4=                       恢复（空）

发现最后 1 = nul，3 = con 了，即当前设定被更改了。
```

补充

我们可以比较一下下面两句的差别

echo 你好！ >con 2>con

echo 你好！

（*有人会问了 “你不是说句柄 1 和 2 默认指向为 con 吗？”*）

没错，我来解释一下。

第一句对句柄 1 和 2 进行了重新设定（虽然是和默认一样的），因此要有备份和取回的过程；

而第二句没有设定，直接取默认值作为当前设定（不需要备份什么的哦）；
这样第二句就比第一句快一点，如果在一个循环多次的程序中使用，速度差别就很明显了。

应用
一次性屏蔽错误输出（加在开始屏蔽的语句后面）
```
2>nul 3>nul
```

一次性将输出写入文件 a.txt（不包括错误反馈，用法同上）
```
>&3 3>a.txt
```

> 此帖中就有应用（是一次性屏蔽错误输出的，大大加快了程序的运行速度）
> http://bbs.bathome.net/viewthread.php?tid=2372&page=1#pid14024

以上一些个不实用的东西大家没必要深究，了解了解就行了，应用为上嘛。
实在需要研究的人例外。

问题：经过重定向后句柄的指向已经不同于默认指向了，还能恢复成原来的默认指向吗？
回答：因为指向不能设定或复制成空，所以语句运行完后的指向如果改变了是不能回到默认指向的。