<https://blog.csdn.net/hn_sun/article/details/73849528>

什么是ioctl  
设备的IO通道进行管理的函数。对设备的一些特性进行控制，传输波特率，马达转速。  
  
  
在用户空间，函数原型：

**[plain]** [view plain](https://blog.csdn.net/hn_sun/article/details/73849528) [copy](https://blog.csdn.net/hn_sun/article/details/73849528)

1. int ioctl(int fd, int cmd, ...) \\...可选参数：插入 \*argp，是cmd命令所需参数，至多指定一个参数，也可以不指定。该参数要么是一个指针，要么是一个整数。

fd是用户程序打开设备时使用open函数返回的文件标识符，cmd是用户程序对设备的控制命令，后面的补充参数，就可以把几个数据捆绑成一个结构，然后传递结构的指针强制转换。  
ioctl 函数是文件结构中的一个属性分量，驱动提供支持，就可以用户程序中使用ioctl函数来控制设备的IO通道。  
  
必要性  
write函数中检查一下是否有特殊约定的数据流通过，后面跟着控制命令。这样程序结构混乱，代码分工不明。用户程序所做的只通过 命令码(ioctl) 告诉驱动想做什么，解释命令和实现命令，驱动程序要做的。  
  
实现  
ioctl中命令码 是唯一联系用户程序命令和驱动程序支持的途径。一定要做到命令和设备是一一对应。linux核心定义一个命令码：  
设备类型 (8it) 序列号(8bit)

方向(2bit) 数据尺寸(8-14bit)  
  
  
一个命令就变成了一个整数形式的命令码，不直观的原因Linux Kernel提供了一些宏。“幻数”是一个字母，数据长度也是8，用一个特定的字母来标明设备类型，和用一个数字是一样的，更加利于记忆和理解。  
cmd 参数再用户程序端由一些宏根据设备类型、序列号、传送方向、数据尺寸等生成，这个整数通过系统调用传递到内核中的驱动程序，再由驱动程序解码宏从这个整数中得到设备的类型、序列号、传送方向、数据尺寸等信息。  
  
  
小结

理解cmd命令码 怎么在用户程序里生成并在驱动程序里解析，主要工作量在 switch结构中

这些宏都定义在<asm/ioctl.h>头文件中(<asm-generic.h>)  
驱动ioctl方法

**[plain]** [view plain](https://blog.csdn.net/hn_sun/article/details/73849528) [copy](https://blog.csdn.net/hn_sun/article/details/73849528)

1. int (\*ioctl)(struct inode \*inode, struct file \*flip, unsigned int cmd, unsigned long arg);

内核提供了一些宏来帮助定义  
\_IO(type, nr)    
\_IOR(type, nr, size)  
\_IOW(type, nr, size)  
\_IOWR(type, nr, size)

相对的，Linux内核提供了相应的宏来从ioctl来解析相应的域值  
\_IOC\_DIR(nr)  
\_IOC\_TYPE(nr)  
\_IOC\_NR(nr)  
\_IOC\_SIZE(nr)

一般使用，先指定各个IOCTL命令的顺序编号(从0开始)，然后根据环境使用这些宏自动生成IOCTL命令号  
ioctl函数使用，如果命令参数是一个指针，指向的地址是用户空间的虚拟地址，内核空间无法使用，用于在内核空间访问用户空间的数据，定义在 <asm/uaccess.h>

**[plain]** [view plain](https://blog.csdn.net/hn_sun/article/details/73849528) [copy](https://blog.csdn.net/hn_sun/article/details/73849528)

1. unsigned long \_\_must\_check copy\_to\_user(void \_\_user \*to, const void \*from, unsigned long n);
2. unsigned long \_\_must\_check copy\_from\_user(void \*to, const void \_\_user \*from, unsigned long n);
3. //copy\_from\_user 和 coyp\_to\_user 一般用于复杂的或大数据交换。

对于简单的数据类型，如int或char，内核提供简单宏（x是内核地址，ptr是用户空间地址）

**[plain]** [view plain](https://blog.csdn.net/hn_sun/article/details/73849528) [copy](https://blog.csdn.net/hn_sun/article/details/73849528)

1. #define get\_user(x,ptr)
2. #define put\_user(x,ptr)