# C语言中修改字符串的一个字符的值

字符串创建：

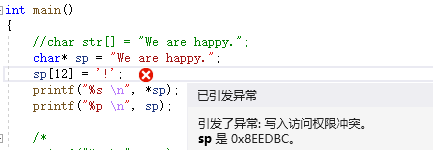
char str[] = "We are happy."; 和 char\* sp = "We are happy.";

结论：

char\* sp = "We are happy." 无法修改字符串内容，因为这个字符串存放在内存的常量区；

char str[] = "We are happy."可以修改字符串内容，因为这个字符串是存放在栈中；

(char\* sp会报错：)



解释：

(1)当执行语句 char\* sp = "We are happy." 时，计算机操作步骤：

1）程序加载字符串值，并存放到常量存储区，常量存储器的内容是只读的；

2）程序在栈上创建 sp 变量；

3）将 sp 变量设置为 "We are happy." 的地址；

3）若程序试图修改 sp 变量所指向的字符串内容就会报错，因为字符串在常量存储器，是只读的。

(2)当执行语句 char str[] = "We are happy." 时，计算机操作步骤：

1）程序加载字符串值，并存放到常量存储区，常量存储器的内容是只读的；

2）程序在栈上初始化一个数组（分配空间），并自动保证该数组足够大以容纳字符串；

3）程序将字符串的内容复制到栈的数组内。

总结：字符串存储在内存的常量区，不能修改，char\* sp是一个指针，指向字符串所在常量区的地址，而char str[]则是创建数组（在栈中分配了内存地址），将原字符串内容拷贝到这个地址中，所以可以修改内容。

char\* sp修改字符的方法：

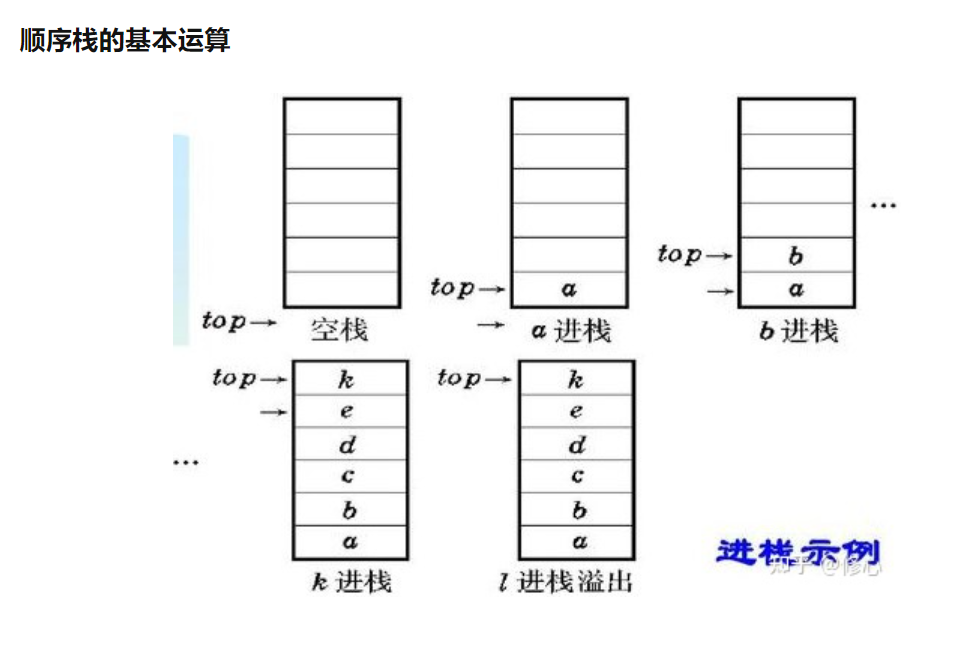
char\* sp = （char \*）malloc(sizeof(char)); //在栈中创建地址

stycpy (sp, "We are happy."); //拷贝字符串内容

\*(sp + 12) = ‘!’; //修改字符，>> "We are happy!"

# 栈

通俗来说，栈是一个数组(int arr[max] ) + 栈顶的index（int top）;



Struct stack

{

Int top;

int arr[max]

}

初始化：top一般初始化为-1；但是需要特别主义的是数组中arr[-1]是栈溢出；当然top也可以初始化0，后续数组注意index；

入栈：top = top + 1 ; arr[top] = xxx;

出栈：top = top-1;

# scanf()读取空格及多次使用scanf时，getchar()清除缓冲区

scanf()函数接收输入数据时，遇以下情况结束一个数据的输入：

① 遇空格、“回车”、“跳格”键。

② 遇宽度结束。

③ 遇非法输入。

Eg:

输入：I love you >>scanf(“%s”,arr);

输出：I >>printf(“%s\n”, arr);

Scanf如何读取空格呢？让scanf读取到“\n”结束。

Eg:

输入：I love you >>scanf(“%[^\n]”,arr);

输出：I love you >>printf(“%s\n”, arr);

**另外：**

在使用scanf（”%c”, first）;时，由于读取输入时会在缓冲区中留下一个字符'\n'（输入完first的值后按回车键所致），如果不清除缓冲区的回车符，在下一次scanf时，不会读取键盘输入的字符，而是读取回车符，从而导致读取有误。所以需要getchar()函数来从输入缓冲区顺序读走一个字符（包括空格，回车和TAB），相当于清除缓冲区。

# printf和整型

整型分为int、short、long、long long，在64位操作系统下其sizeof分别是4，2，4，8字节，以long long 为例，其又分为有符号和无符号，有符号的范围是：（-2（64-1））~ （2（64-1）-1），无符号的范围是：0~ （2（64）-1）。



Printf输出格式：%d——有符号整形（int）十进制（4个字节以

内的可以输出）

%u——无符号整形（int）十进制（和%d同理）

%ld——有符号长整形（long int）十进制

%lu——无符号长整形（long int）十进制

%lld——有符号long long int十进制

%llu——无符号long long int十进制

Eg:

unsigned long long aa = 12345678912345678912U;

printf("%llu\n", aa); //2（64）十进制是20位，所以输出时也必须

//以无符号形式输出，否则是乱码

另外：十进制，八进制，十六进制，输出控制符分别为%d, %o, %x。

# heap-buffer-overflow 堆溢出

**产生原因：**

一般是数组越界，可能初始化的数组不够长或者在程序中出现了下标越界的情况。

**解决方法：**

检查代码中数组下标是否使用正确，排除越界的可能性。

**注：**

在for循环中如果有数组指针，其index做减运算，必须在其判断条件中添加index > 0的条件，且将该条件放在s[index]或\*(s + index)条件之前，（因为a&&b 运算先执行a，a满足再执行b，如果a不满足，则不执行b），我们要保证s[index]或\*(s + index)的index不能为负数（小于0）。

Eg:

for (n = 10; n >= 0 && \*(s + n) != ' '; n--)

# new 和 malloc

函数原型：

（类型T\*）new 类型T（初始化表）；

void \*malloc (int size);

（1）申请内存位置所在：

new 是C++中用来分配内存的，但它不是从堆上分配内存，而是从自由存储区上分配内存的，自由存储区是C++的抽象概念，即用new分配的内存都是自由存储区，它可以是堆也可以不是，取决于operator new的实现细节；它搭配delete删除这块内存。

malloc是C语言中用来在堆上动态内存分配，堆是操作系统来维护的一块特殊内存，使用free来释放这块内存。

（2）返回类型

new如果成功返回的是对象类型的指针，不需要强制转换，所以new是符合类型安全性的操作符；如果内存分配失败，new会抛出bac\_alloc异常，不会返回NULL，所以在用new的时候不需要判空，而是catch (bad\_alloc)。

malloc如果成功返回void\*类型，需要强制转换为我们需要的类型；如果内存分配失败则返回NULL，所以在malloc时，需要判空。

（3）是否需要指定内存大小

new操作符在分配内存时无需指定内存大小，编译器会根据类型信息按类型的sizeof分配内存。

Eg：

int \*a = new int; //分配一个sizeof(int)的内存大小给a

int \*a = new int(5); //分配一个sizeof(int)的内存大小给a,并初始化为5

int \*a = new int[100]{0}; //分配100个整型内存空间给a，并初始化为0

//数组对应 new int [100]; delete [ ] a;

int (\*a)[20] = new int[100][20] //分配100个整型内存空间给\*a

注：二维数组在创建时要指定后面的大小

Malloc在申请内存时，需要自行指定内存大小。

Eg：

class A{...}

A \* ptr = new A;

A \* ptr = (A \*)malloc(sizeof(A)); //需要显式指定所需内存大小sizeof(A);

1. 是否调用构造函数、析构函数

new 和 delete 在申请、释放内存时，要分别调用类的构造函数和析构函数，在new的时候，如果构造函数有参数，还要赋予参数，否则找不到对应的构造函数。

malloc仅申请内存，并不初始化。

Eg：

team A("lizhi", 1);

team \*B = new team(A); //传A调用拷贝函数，

//team \*B = new team("lizhi", 1); //传参调用构造函数

team\* C = (team\*)malloc(sizeof(team));

delete(B);

数组：

team A("lizhi", 1);

team\* B = new team[2]{{"lizhi",1},{"jinhui",2}};

team\* C = (team\*)malloc(sizeof(team));

delete [] B;

使用malloc来处理C++并不合适，因为它不能调用构造函数和析构函数。另外，new和delete可以重载，malloc和free不能重载。

# Strcpy、srtncpy、memcpy、memset

## (1) char \*strcpy(char \*s2, const char \*s1);

**作用：**函数strcpy把s1指向的字符串，直到“\0”（空字符）复制到s2指向的数组中。所以strcpy函数操作后的字符串是带“\0”的。

**注意：**strcpy 是依据 “\0” 作为结束判断的，如果 s2 的空间不够，则会引起 buffer overflow。

**返回值：**函数strcpy返回s2的值。

## (2) char \*strncpy(char \*s2, const char \*s1, size\_t n);

**作用：**函数strncpy从s1指向的数组中最多复制n个字符（不复制“\0”后面的字符）到s2指向的数组中。如果s1指向的数组是一个比n短的字符串，则在s2定义的数组后面补“\0”，直到写入了n个字符;。

**注意：**如果s1指向的数组是一个比n长的字符串，则s2中复制的字符串则不以“\0”结尾，使用strncpy时，确保s2的最后一个字符是“\0”

**返回值：**函数strncpy返回s2的值。

## (3) void \*memcpy(void\*dest, const void \*src, size\_t n);

**作用：**复制 src 所指的内存内容的前 n个字节到 dest 所指的内存地址上。memcpy() 并不关心被复制的数据类型，只是逐字节地进行复制，这给函数的使用带来了很大的灵活性，可以面向任何数据类型进行复制。

**注意：**dest 指针要分配足够的空间，也即大于等于 num 字节的空间。如果没有分配空间，会出现断错误；dest 和 src 所指的内存空间不能重叠（如果发生了重叠，使用 memmove() 会更加安全）。

**另外：**与 strcpy() 不同的是，memcpy() 会完整的复制 num 个字节，不会因为遇到“\0”而结束。

**返回值：**返回指向 **dest 的指针**。注意返回的指针类型是 void，使用时一般要进行强制类型转换

Eg:

char\* p1 = "abcde";

char\* p2 = (char\*)malloc(sizeof(char) \* N);

char\* p3 = (char\*)memcpy(p2, p1, N);

## (4) void \*memset(void \*str, int ch, size\_t n)

**作用：**在一段内存块中填充某个给定的值，它是对较大的结构体或[数组](https://so.csdn.net/so/search?q=%E6%95%B0%E7%BB%84&spm=1001.2101.3001.7020)进行清零操作的一种最快方法。

memset函数是对地址为str开始的n个字节的内存块进行赋值，而且是将这个n个字节的每个字节赋值为整数ch。这一点要注意，使用memset函数对内存块赋值是以字节为单位来赋值。

**注意：**

可以使用memset函数对char类型数组的元素进行赋值（0值或非0值）。

可以使用memset函数对非char类型数组的元素进行赋0值，但不能使用memset函数直接对非char类型数组的元素赋非0值。这里的非char类型数组，是指数据类型大小不是1个字节的，比如short、int、long、占用字节数大于1个字节的结构体等。

**库文件：** [<string.h>](https://www.runoob.com/cprogramming/c-standard-library-string-h.html)

Eg:

int\* nums = (int\*)malloc(sizeof(int) \* (r\_s));

memset(nums, 0, sizeof(int) \* r\_s); //以字节为单位

## 总结：strcpy()和strncpy()只能复制字符串，而memcpy()可以复制任意内容，例如字符数组、整型、结构体、类等；

# strlen()和sizeof（）

（1）Strlen()是函数，在运行时计算字符串的大小，参数为char \* ，实际上传数组即指针地址。它的功能是：返回字符串的长度。实际工作是：从所给字符串的第一个地址开始遍历，遇到“\0”结束，不包括“\0”，当传传参数为char \* 时，用strlen()计算字符串长度，不包括“\0”，另，“\0”的意义是数字0。

Eg1：

char a[] = "asdfgg23sdd" ;

int len = strlen(a);

printf("%d\n", len); // 11

另：char arr[] = { 'a','b','c','d','e','f' };是数组，‘a’为一个字节，‘b’ 为一个字节,所以arr没有“\0”，char arr1[] = { 'ads','bfafd' };是错误的。

Eg2：

char xx[5] = { 0 };

xx[0] = 1;

xx[1] = 2;

xx[2] = 3;

xx[3] = 0;

xx[4] = 4;

printf("%d", strlen(xx)); //输出3；数值0相当于\0

char xs[5] = "1203";

printf("%d", strlen(xs)); //输出4；这里0相当于一个字符‘0’，以ASCII码值48存储

注：字符或者字符串是用‘’或“”表示，在内存中以ASCII值存储，而数字没有ASCII码，如：char a[0] = 1; 1表示数字1，在计算机中按数值1存储；而char a[0] = ‘1’ 或者char a[5] = “1234”; 中的‘1’表示字符‘1’，在计算机中按其ASCII值49存储；另外值得注意的是，\0表示的是数字1。

（2）sizeof是运算符，在编译期间就计算好了，它的功能是：获取能保证实现建立最大对象的字节大小。注：如果sizeof遇到字符串，它会加上‘\0’。

Eg：

char a[] = "asdfgg23sdd" ;

int len = sizeof(a);

printf("%d\n", len); //12

注：sizeof的参数：

1）数组名：表示整个数组所占空间的大小，单位字节；

2）指针：存储该指针的空间大小，32位机器任何类型的指针大小都为4,64位机器任何类型的指针大小都为8；

3）类型：该类型所占空间大小。

Eg1：

char str[] = { 0x01 , 0x50, 0x45, 0x33, 0x89 };

int len1 = sizeof(str) / sizeof(char);

printf("%d\n", len1); //5个字节

int str[] = { 0x01 , 0x50, 0x45, 0x33, 0x89 };

int len1 = sizeof(str);

printf("%d\n", len1); //20个字节, 4\*5; 0x01占了4个字节（一个int）

int len2 = sizeof(str) / sizeof(int);

printf("%d\n", len1); //5个数

Eg2：

char str[] = { 0x01 , 0x50, 0x45, 0x33, 0x89 };

char \*p = str;

int len2 = sizeof(p);

printf("%d\n", len2); //4个字节

另：

1）char 是有符号的 ，uchar (unsigned char) 是无符号的，8-bit无符号整形数据，里面全是正数。

2）两者当整数用时有区别： char 整数范围为-128到127 (0x80\_\_0x7F)，而unsigned char 整数范围为0到255 (0\_\_0xFF) 有时候想把整数数值限在255范围内，也用unsigned char

3）char str[] = { 0x01 , 0x50, 0x45, 0x33, 0x89 };

char \*p = str;

因为：

str + 1 = &str[1]；

\*（str + 1）= str[1]；

所以

p = str；

str + 1= p + 1 = &str[1]；

且指针也可以用p[1]

str + 1= p + 1 = &str[1] = & p[1].

4）想要表达字符串中的结束符（\0）时，是‘\0’而不是“\0”。

单引号是字符，双引号是字符串。

char z = 'a'; //正确，单个字符a

char z = "a"; //错误，这是个字符串，所占空间为2个字符，所以是错的

char z[2] = "a"; //正确，这是个字符串，所占空间为2个字符，a \0

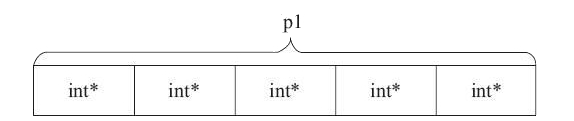
# 指针常量和常量指针

指针常量：指针类型的常量，即地址是一个常量，不可修改； int \* const p; p表示地址，所以const修饰地址p,表示地址是一个常量。

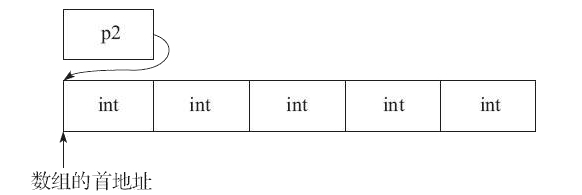
常量指针：指向常量的指针，即地址中的内容是一个常量； const int \* p 或int const \* p；const修饰\*p,\*p是地址中的值，所以表示地址中的内容是一个常量。

# 指针数组和数组指针

指针数组：指针的数组，即这个数组里放的全是指针；int \*p1[5]——[]比\*的优先级高，所以先是p1[5]数组，其内容是5个int \*的指针；



数组指针：数组的指针，即指针指向一个数组；int (\* p2)[5]——()比[]的优先级高，所以（\*p2）是一个指针，指向一个包含5个int类型的数组；



Eg：

int b[3][5] = { {1,2,3,4,5} ,{6,7,8,9,10}, {11,12,13,14,15} };

//int \*lj[5]; //lj是一个数组，放了5个int\*的指针

int (\*lz)[5] = b; //lz是一个指针，指向放了5个int类型的数组

int x = lz[1][5];

# 错误信息：对重载函数的调用不明确

有多个重载函数实例与参数列表匹配

错误原因：代码中可能存在多个函数名一致，但参数表不一致的函数；

解决办法：换个函数名，或明确函数的参数类型。

# 二维数组

二维数组定义时，不可忽略列数；数不够时，用0补。

Eg：

int a[3][2] = { {12,1}, {11,8}, {14,2} };

int b[][2] = { 12,1,11,8,14,2 }; //2列，n行，所以b[][2]是{{12,1},{11,8},{14,2}}

int bx[2][] = { 12,1,11,8,14,2 }; //错误的表达式

int c[][2] = { 12,1,11 }; //2列，n行，不够的用0补，所以c[][2]是{{12,1},{11,0}}

# 数据类型及其范围

32位系统中，int为4个字节，32位，范围为-231~（231-1），unsigned int的范围为0~（232-1）；char为1个字节，范围为-27~（27-1），unsigned char 的范围为0~（28-1）；另外，浮点型的float（4个字节） 和 double（8个字节（双字））都不能用unsigned来修饰，因为浮点型是有符号的，unsigned只能修饰整形。一个word是4个字节。编译器一般默认有符号类型。

有符号类型中，最高位为符号位，0表示正数，1表示负数；在计算机中，一般以补码的形式存储数据，正数的原码、反码、补码都是它本身，负数的反码是其原码中符号位不变，其余取反，补码为其反码加1。

Eg:

127

原码:0111 1111

反码:0111 1111

补码:0111 1111

-127

原码:1111 1111

反码:1000 0000

补码:1000 0001

所以-127在计算机中存储为1000 0001

在一个char类型中，-128不能用原码或反码表示，而是用-0来表示-128。（补码相加时，如果最高位——符号位有进位时，则进位被舍弃）。

Eg：

-0

原码：1000 0000

补码：1111 1111

反码：1000 0000

如果最高位（符号位）有进位，则进位被舍弃

# 位运算

（1）与&

两位同时为1，结果才为1，否则为0；

另：负数按补码形式进行按位“与”运算。

用途：

1）清零：任何二进制与0进行与运算，结果都为0；

2）取一个数中的指定位：令指定位为1，其余都为0；

例：设X=10101110，

取X的低4位，用 X & 0000 1111 = 00001110 即可得到；

还可用来取X的2、4、6位。

（2）或|

两个对象只要一个为1，则结果为1；

另：负数按补码形式进行按位“或”运算。

用途：常用来给一个数据中的某一位置1： 令指定位为1，其余都为0；

例：设X=10101110，

将X的最低位置1，用 X | 0000 0001 = 10101111 即可得到；

（3）异或^

两个相应位为“异”（值不同），则结果为1；相同则结果为0；（与1异或即取反，与0异或即不变）（异或相当于不进位的加法：1+1=0,0+0=0,1+0=1）

用途：

1）对一个数的指定位进行翻转：令指定位为1，其余都为0；

例：设X=10101110，

将X的低4位进行翻转，用 X ^ 0000 1111 = 1010 0001 即可得到；

2）与0异或，保留原值：

例：设X ^ 0 = X;

3）与自身异或等于0：

例：设X ^ X = 0;

（4）左移运算符（<<）

规则：各二进制位左移若干位，高位丢弃，低位补0（若左移时，舍弃的高位不包含1，则每左移一位，相当于该数乘以2）；

例：设X=10101110，

X << 2 ，注意左移并没有改变X的值，X还是10101110，如果b = X << 2,将X左移2位后赋值给b，此时b为10111000，X为10101110；

X <<= 2与b = X << 2意义相同，是将X左移2位，然后赋值给X。

（5）右移运算符（>>）

规则：各二进制位右移若干位，低位丢弃，高位正数补0，负数补1（注意：负数进行运算时，都是以补码形式进行运算）（操作数每右移一位，相当于该数除以2）

例：设X = 12，Y = -12，(X: 0000 0000 0000 1100; Y: 1111 1111 1111 0100)

X >>= 2 ，Y >>= 2 (X: 0000 0000 0000 0011; Y: 1111 1111 1111 1101)

# 浮点运算

float zz = 1 / 2 \* 13.114\* 10.2; >>zz = 0;因为1 / 2默认int型，所以1/2 = 0;

float zz = 1 .0/ 2 \* 13.114\* 10.2; >>zz = 66.88…;因为1.0 / 2是float型，所以1.0/2 = 0.500…

总结：做除法运算时，一般用float型。

# const char\*不能初始化char\* 解决方法

（1）改属性，c/c++ -> 语言 -> 符合模式改为“否”；

（2）强制类型转换——char \*arr = (char \*)"asdfghj";

（3）先用字符数组存储，再转换；

char pp[] = "asdfghj";

char \*arr = pp;

12.

（1）结构体或数组赋值：

1）清零：void \* memset(void \* dest, int val, size\_t size) (该函数在string.h中)

Eg：

Struct Student A;

memset(&A, 0, sizeof(struct Student));

2）逐个赋值

Eg：

A.name = (char \*)"aa";

A.old = 18;

A.sex = 'W';

（2）字符串赋值：

Strcpy(char \* dest, const char \* sourse);

Memcpy(char \* dest, const char \* sourse, size\_t size);

两者的区别在于Strcpy函数将sourse全部内容拷贝到dest中，没有size，可能造成buffer overflow（buffer overflow可能不会报错，但是字符数组里没有‘\0’），但是拷贝完后会自动添加‘\0’；而Memcpy函数有size，但是拷贝完不会添加‘\0’，所以Memcpy函数像修改连接作用。

注：dest如果是一个指针，必须先给其赋个地址才能使用，如果dest是一个数组名，就可以直接使用。

（3）数组打印：

1）如果printf打印的是字符数组，则遇到‘\0’结束；

2）如果printf打印的是数字数组，则必须循环打印size。

# typedef 和 初始化

int x, y, z = 0; //x,y,z都是int型，但只给z初始化为0；

int \*x, y, z = 0; //x为int \*型，而y,z都是int型，且只给z初始化为0；

其等价为：int \* x; int y; int z = 0;

typedef int a,b; //a和b都可以代表int

typedef struct

{ GLfloat x, y;

} vec\_t, \*vec; //将结构体重命名为vec\_t结构体类型 和 \*vec结构体指针类型

vec\_t ax; //定义一个vec\_t结构体类型变量

vec ay; //定义一个vec结构体指针类型变量

ax.x = 0;

ay->y = 1;

# 链表中对地址操作

Link first,tempt, node;

first = (Link)malloc(sizeof(struct link)); /\*首元结点\*/

first->val = arr[0];

first->next = NULL;

tempt = first; /\*头指针，中间变量\*/

for (int i = 1; i < size; i++)

{

node = (Link)malloc(sizeof(struct link)); /\*结点\*/

node->val = arr[i];

node->next = NULL;

tempt->next = node;

tempt = node;

}

如果直接对tempt直接操作，first中的值也会直接改变；所以第一次操作时是对tempt->next，也就是对first->next进行操作，接下来先让tempt 指向node的地址，后面的tempt的操作就是node的操作。

12.

串口通信：硬件设备的物理特性接口，如232,485,422，应用于工业控制领域；

网络通信：以太网，ip tcp网络，应用于物联网；