嵌入式C语言之-

字符串的本质

讲师：叶大鹏

助力你成为优秀的电子工程师！

PN学堂

字符串

- 字符串是一种特殊的数据，是一些ASCII码字符的集合，包含在" "里：

```
uint8_t *p = (uint8_t *)malloc(4);
if (p == NULL)
{
        printf("malloc for p failed!\n");
        return NULL;
}
```

注意：要和ASCII码字符区别开，'A'表示的是一个字符。

字符串

- C语言没有专用的字符串数据类型（C++有string数据类型），通常使用char []和 char *操作字符串数据：

```
char *version = {"V1.1.4"};
char version[] = {"V1.1.4"};
```

如果展开printf("malloc for p failed!\n"):

```
int printf(char *fmt, ...)
char *fmt = "malloc for p failed!\n"
```

为什么字符串能直接赋值给指针?

```
int main(void)
{
    char *fmt = "malloc for m failed!\n";

    return 0;
}
```

```
main
    0x000004d0:    a101          ..      ADR      r1,{pc}+8 ; 0x4d8
    0x000004d2:    2000          .       MOVS     r0,#0
    0x000004d4:    4770          pG      BX       lr
$d
    0x000004d6:    0000          ..      DCW      0
    0x000004d8:    6c6c616d      mall    DCD      1819042157
    0x000004dc:    6620636f      oc f    DCD      1713398639
    0x000004e0:    6d20726f      or m    DCD      1830842991
    0x000004e4:    69616620       fai    DCD      1767990816
    0x000004e8:    2164656c      led!    DCD      560227692
    0x000004ec:    0000000a      ....    DCD      10
```

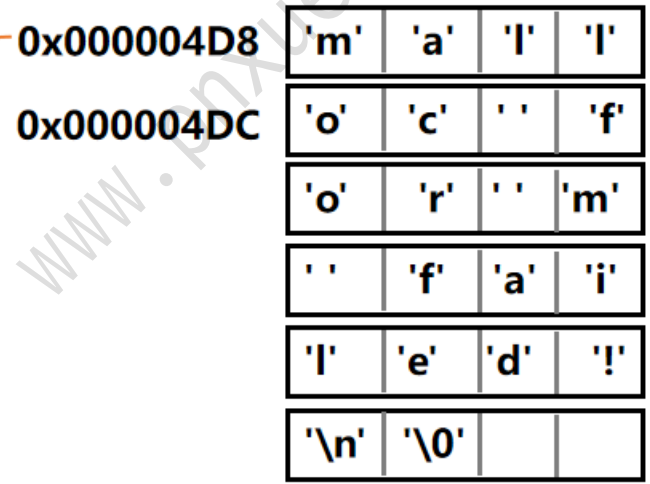
为什么字符串能直接赋值给指针?

```
char *fmt = "malloc for m failed!\n";
```

fmt | 0x000004D8

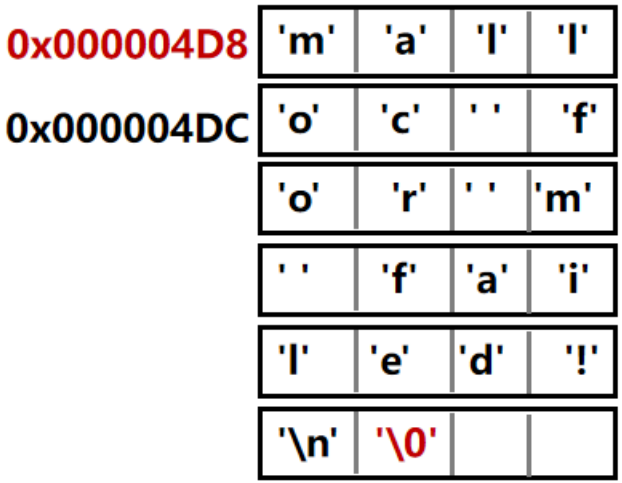
0x20000408~40C

"malloc for m failed!\n"



字符串本质

"malloc for m failed!\n"



FLASH空间

- 字符串本质：

1. 字符串保存在单片机FLASH空间中，数据像数组一样连续分布；
2. 字符串本身具有隐含功能，代表了存储空间的首地址，所以能直接向指针变量赋值；
3. 字符串实际数据不包含" "；
4. 编译器在处理字符串时，会自动的在数据末尾添加ASCII码'\0'，对应十进制0，便于程序对字符串解析；
5. 由于字符串数据保存在FLASH上，所以这段地址空间上的数据内容在运行时只能读取，不能修改。

指针与字符串

```
char *fmt = "malloc for m failed!\n";
printf("malloc for p failed!\n");
printf("%s", fmt);
printf("fmt[0] = %c, fmt[1] = %c, fmt[2] = %c\n",
        fmt[0], fmt[1], fmt[2]);
```

```
malloc for m failed!
fmt[0] = m, fmt[1] = a, fmt[2] = l
```

指针与字符串

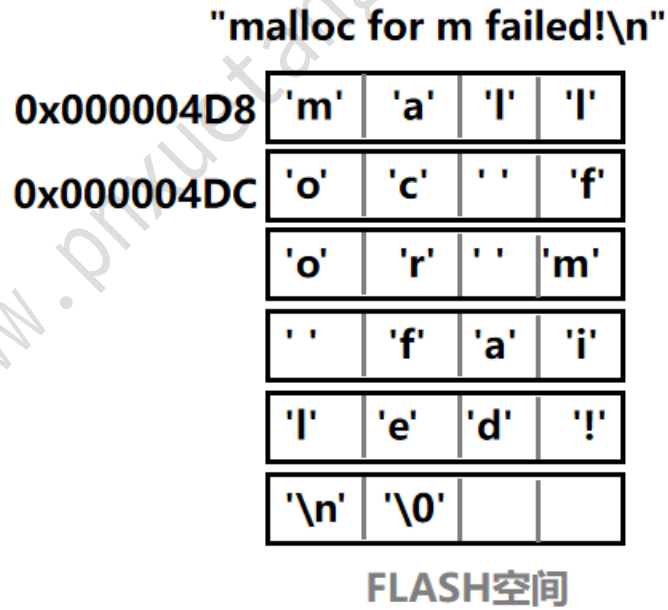
```
char *fmt = "malloc for m failed!\n";
fmt[0] = 'M';
printf("fmt[0] = %c, fmt[1] = %c, fmt[2] = %c\n",
        fmt[0], fmt[1], fmt[2]);
```

```
fmt[0] = m, fmt[1] = a, fmt[2] = l
```

指针与字符串

```
char *fmt = "malloc for m failed!\n";
fmt[0] = 'M';
printf("fmt[0] = %c, fmt[1] = %c, fmt[2] = %c\n",
        fmt[0], fmt[1], fmt[2]);
```

```
fmt[0] = m, fmt[1] = a, fmt[2] = l
```



5. 由于字符串数据保存在FLASH上，所以这段地址空间上的数据内容在运行时只能读取，不能修改；

PN学堂

如何修改字符串-1

```
char fmt[] = "malloc for m failed!\n";
fmt[0] = 'M';
printf("fmt[0] = %c, fmt[1] = %c\n", fmt[0], fmt[1]);
```

```
fmt[0] = M, fmt[1] = a
```

如何修改字符串-1

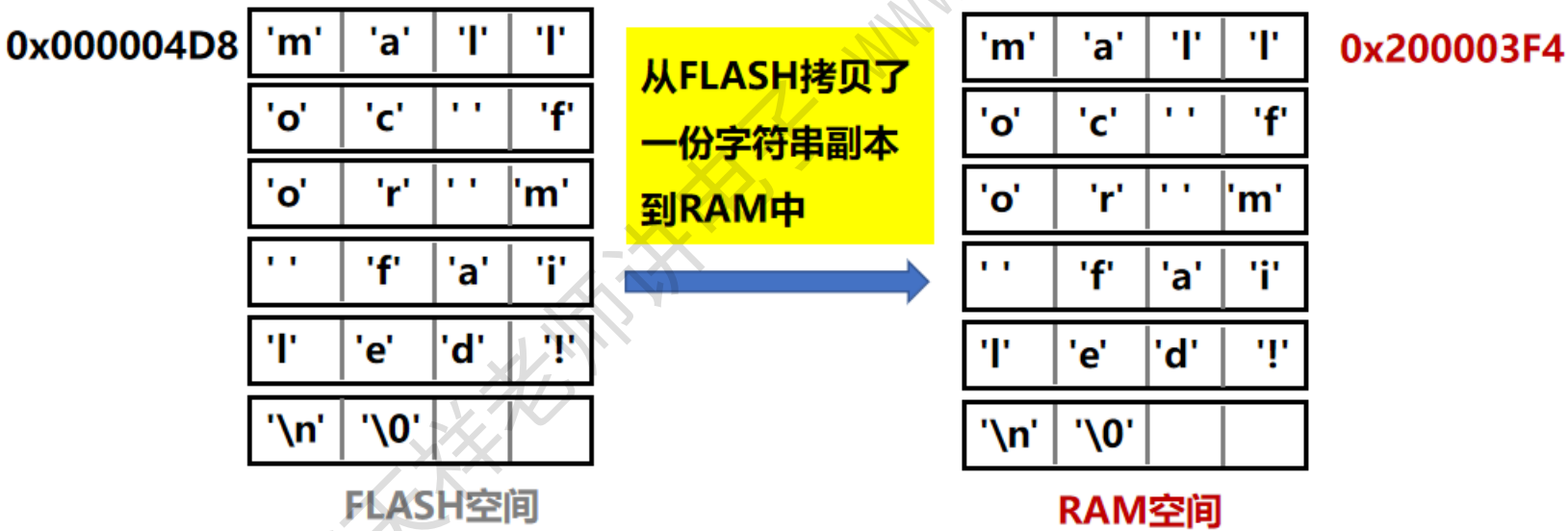
```
char fmt[] = "malloc for m failed!\n";
fmt[0] = 'M';
printf("fmt[0] = %c, fmt[1] = %c\n", fmt[0], fmt[1]);
```

```
ADR        r1,{pc}+0x22 ; 0x6fc
ADD        r0,sp,#4
BL         __aeabi_memcpy4 ; 0x572
```

如何修改字符串-1

```
ADR       r1,{pc}+0x22 ; 0x6fc
ADD       r0,sp,#4
BL        __aeabi_memcpy4 ; 0x572
```

"malloc for m failed!\n"



如何修改字符串-2

- 参照数组的方法，使用动态内存，手动拷贝字符串数据到内存中：

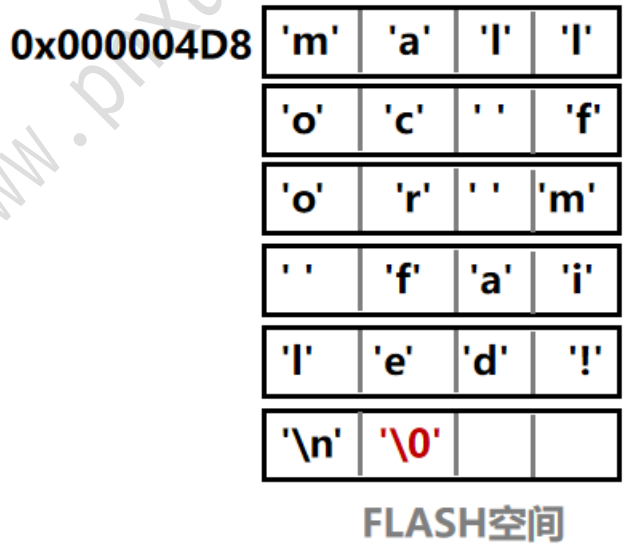
```
#include <stdlib.h>
#include <string.h>

char *fmt = (char *)malloc(strlen("malloc for m failed!\n") + 1);
strcpy(fmt, "malloc for m failed!\n");
fmt[0] = 'M';
printf("fmt[0] = %c, fmt[1] = %c\n", fmt[0], fmt[1]);
```

```
fmt[0] = M, fmt[1] = a
```

strlen()函数简析

- strlen()函数用于返回字符串的长度，不包括结尾\0；
- 原型可以简化为：

```
uint32_t strlen(char *str)
{
    uint32_t len = 0;
    while (str[len] != '\0')
    {
        len++;
    }
    return len;
}
```

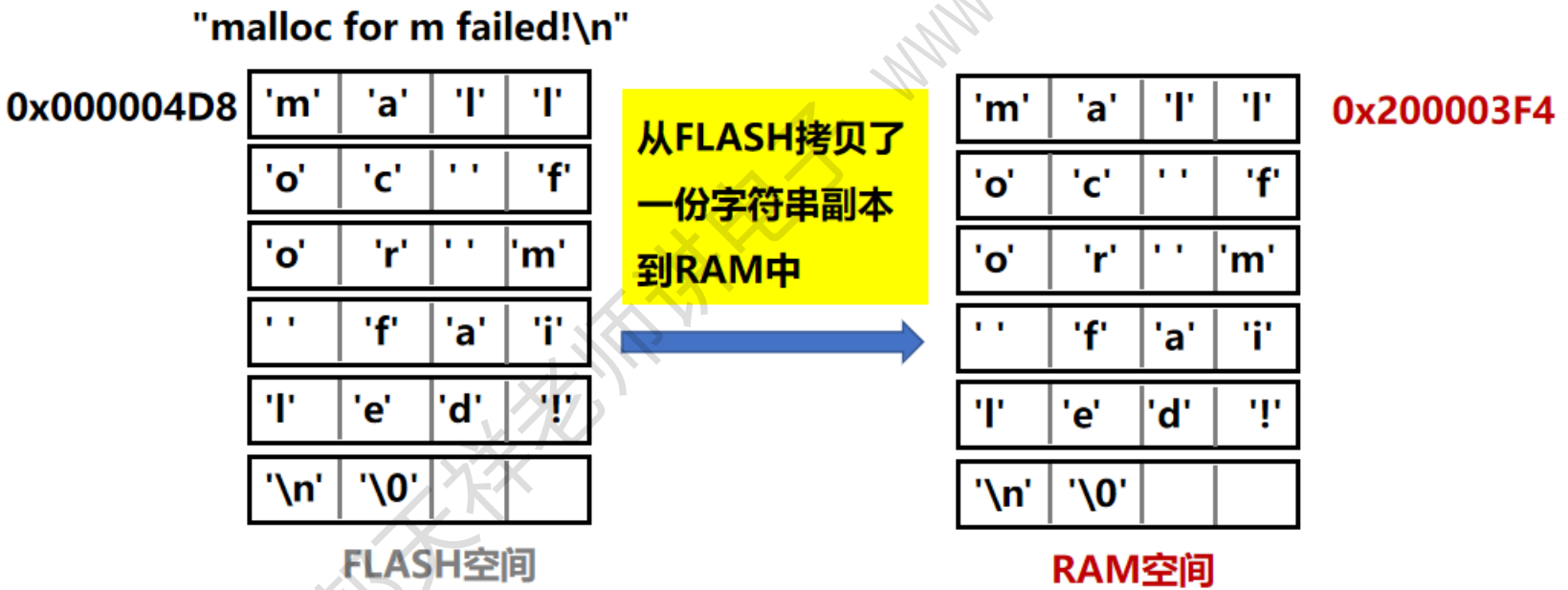


3. 编译器在处理字符串时，会自动的在数据末尾添加ASCII码'\0'，对应十进制0，便于程序对字符串解析。

注意事项

```
char fmt[] = "malloc for m failed!\n";
printf("strlen(fmt) = %d.\n", strlen(fmt));
printf("sizeof(fmt) = %d.\n", sizeof(fmt));
```

```
strlen(fmt) = 21.
sizeof(fmt) = 22.
```



注意事项

```
char fmt[] = "malloc for m failed!\n";
printf("strlen(fmt) = %d.\n", strlen(fmt));
printf("sizeof(fmt) = %d.\n", sizeof(fmt));
```

```
strlen(fmt) = 21.
sizeof(fmt) = 22.
```

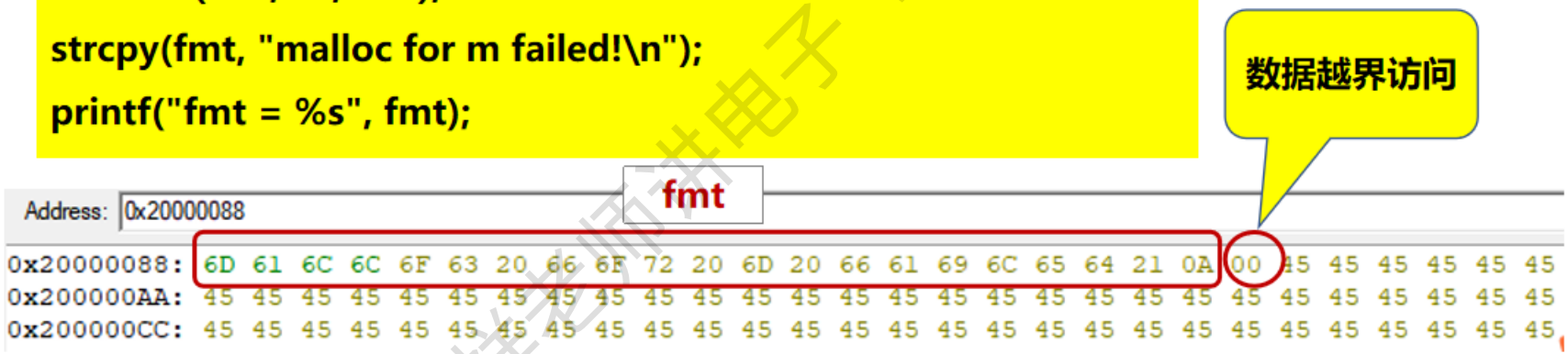
- strlen()计算的是字符串的长度（不包含\0）；
- sizeof()计算的是数组占用的内存空间大小。

注意事项

- 申请动态内存时，记得要给\0申请内存空间：

```
char *fmt = (char *)malloc(strlen("malloc for m failed!\n") + 1);
strcpy(fmt, "malloc for m failed!\n");
```

```
char *fmt = (char *)malloc(strlen("malloc for m failed!\n") );
memset(fmt, 'E', 100);
strcpy(fmt, "malloc for m failed!\n");
printf("fmt = %s", fmt);
```



注意事项

```
char fmt[] = "malloc for m failed!\n";
char fmt[22] = "malloc for m failed!\n";
char fmt[21] = "malloc for m failed!\n";
```

```
main.c(51): error:  #144: a value of type "char [22]" cannot
be used to initialize an entity of type "char [21]"
```

THANK YOU!