嵌入式C语言之-

宏定义中#和##的作用

讲师：叶大鹏

助力你成为优秀的电子工程师！

应用案例

- 在一些开源代码中，可以看到这样的代码：

```
#define STR(param)  #param
```

- 在stdint.h中，可以看到这样的代码：

```
#define UINT8_C(x)  (x ## u)
#define UINT16_C(x) (x ## u)
#define UINT32_C(x) (x ## u)
```

- 也可以看到这样的代码：

```
#define LOG(fmt, ...)   printf(fmt, ##__VA_ARGS__)
```

宏定义中#的作用

- 宏定义中#的作用，简单说就是将宏参数的左右各加上一个双引号，变成字符串。

```
#define STR(param)   #param
char *pStr = STR(hello);//展开后就是char *pStr = "hello";
```

宏定义中#，常用在输出日志场景

```
#include <stdio.h>
#define STR(param)  #param
#define LEN_MAX    10
int main()
{
    int array[LEN_MAX] = {0};
    int index = 15;
    if (index >= LEN_MAX)
    {
        printf("error: %s:%d is over %s:%d\n", STR(index), index, STR(LEN_MAX), LEN_MAX);
    }
    return 0;
}
```

```
error: index:15 is over LEN_MAX:10
```

宏定义中##的作用

- ##是连接符，可以在带参数的宏定义中将两个子串拼接到一起。

```
#define LINK(x, y, z)       x##y##z

int a = LINK(1, 2, 3);
```


```
#define VAR_COMB(name, size)      char name## _arr[size] = {0}
VAR_COMB(var, 10);
```

➢ 展开后:

```
char var_arr[10] = {0};
```

宏定义中##的作用

```
typedef struct {
    uint8_t *data;
    int32_t  dataSize;  /* number of byte real */
    int32_t  maxSize;   /* maximnm data size.*/
} DataType_t;
int main()
{
        DataType_t var1 = {0};
        uint8_t var1_arr[10] = {0};
        var1.data = var1_arr;
        var1.maxSize = 10;

        DataType_t var2 = {0};
        uint8_t var2_arr[15] = {0};
        var2.data = var2_arr;
        var2.maxSize = 15;
```

宏定义中##的作用

```
typedef struct {
    uint8_t *data;
    int32_t  dataSize;  /* number of byte real */
    int32_t  maxSize;   /* maximnm data size.*/
} DataType_t;

#define DataTypeCreate(name, size)  \
    DataType_t name = {0}; \
    uint8_t name## _arr[size] = {0}; \
    name.data = name## _arr; \
    name.maxSize = size;

int main()
{
        DataTypeCreate(var1, 10);
        DataTypeCreate(var2, 15);
```

宏定义中##的作用

- ##是连接符，可以在带参数的宏定义中将两个子串拼接到一起。

```
#define  A    1
#define  B    2
#define  C    3

#define LINK(x, y, z)        x##y##z

int v = LINK(A, B, C);
```

宏定义中##的作用

- #和##在宏定义中使用时，如果参数也是一个宏，那么是不会展开的；可以定义一个用于中间转换的宏来解决。

```
#define  A    1
#define  B    2
#define  C    3
#define _LINK(x, y, z)       x##y##z
#define  LINK(x, y, z)       _LINK(x, y, z)

int v = LINK(A, B, C);
```

在stdint.h中，可以看到这样的代码：

```
#define  __PASTE2(x, y)   x ## y
#define  PASTE(x, y)  __PASTE2(x, y)
```

宏定义中##的作用

```
#define LOG(fmt, ...)    printf(fmt,  ##__VA_ARGS__)
```

- 在宏定义中，也支持用...代表可变参数，为了引用可变参数，语言层面提供了可变宏(Variadic macros)__VA_ARGS__来引用它；
- 但是，在宏定义时，如果直接使用__VA_ARGS__来引用可变参数，一旦可变参数为空就会引起编译器报错，看看下面的例子：

```
#define LOG(fmt, ...)    printf(fmt,  __VA_ARGS__)
int main(void)
    LOG("%s\n", "Hello world");
    LOG("info\n");  //展开后printf("info\n",);
}
```

为了解决上面的问题，在__VA_ARGS__前面添加上##，这样的目的是告诉预处理器，如果可变参数为空，那么前面紧跟的逗号，在宏定义展开时需要被清理掉。

嵌入式C语言之-

宏定义中为什么使用(void)0

讲师：叶大鹏

助力你成为优秀的电子工程师！

(void)0的应用案例

```
#ifdef   LOG_ENABLED
      #define LOG(fmt, ...)    printf(fmt,  ##__VA_ARGS__)
#else
      #define LOG(fmt, ...)   ((void)0)
#endif
```

```
#ifdef  USE_FULL_ASSERT
      #define assert_param(expr)  ((expr) ? (void)0 : assert_failed((uint8_t *)__FILE__, __LINE__))
      void assert_failed(uint8_t* file, uint32_t line);
#else
     #define  assert_param(expr)  ((void)0)
#endif
```

宏定义中(void)0的作用

- (void)0表示将0强制类型转换为无类型数据，无实际意义，在这里只起到占位符的作用，编译器不会对它生成任何汇编代码。

- 看下面的例子，如果#define assert_param(expr) ((void)0)，将((void)0)去掉

```
#define assert_param(expr)
int main(void)
{
        int8_t foo;
        int8_t bar = 1;

        foo = 1, assert_param(bar == 1), bar = 2;
        // 展开后 foo = 1, , bar = 2; 编译报错

        return 0;
}
```