title	date	categories	tags
C/C++ 宏定义使用	Fri May 06 2016 23:51:02 GMT+0800 (CST)	C/C++	C/C++

宏定义在C语言占有举足轻重的地位。底层框架自不用说，为了编译优化和方便，以及跨平台能力，宏被大量使用，可以说底层开发离开**define**将寸步难行。使用宏的好处是不言自明，在节省工作量的同时，代码可读性大大增加。

通过本文你将了解到：

> - 宏定义入门
> - 对象宏
> - 函数宏
> - 宏定义变长参数

入门

宏定义简单点说就是查找替换，但是如果如此理解，那么只说对了一半，C中的宏分为两类，**对象宏**和**函数宏**。

对象宏

对象宏一般用来定义一些常数，比如：

```
#define PI 3.14159
```

`define` 关键字表明即将开始一个宏定义，仅接着 `PI` 是宏的名字，空格之后的数字是内容。类似这样 `#define X A` 的宏是比较简单的，会在编译期间，将X替换为A，这个过程称之为展开。

函数宏

函数宏是用得最多的，函数宏可以节省大量的工作量。

```
#define MIN(A, B)     A < B ? A : B

int a = MIN(1, 2);
printf("%d", a); => 1
```

输出正确，可以打包么？ 在实际使用中会出问题么，肯定会出问题。

```
int a = 2 * MIN(3, 4);
//=> int a = 2 * 3 < 4 ? 3 : 4;
//=> int a = 6 < 4 ? 3 : 4;
//=> int a = 4;
```

现在知道原因了，我因为展开后运算符优先级的问题，乘法先被运算了，修正简单加个括号就行了

```
#define MIN(A, B) (A < B ? A : B)
```

仅仅如此么，如果我们执行 `MIN(3, 4 < 5 ? 4 : 5)` 发现结果为4，展开宏后发现，展开式时链接符号和被展开式中的运算符号优先级相同，导致计算顺序发生变化。所以还得再严格点。

```
#define MIN(A, B) ((A) < (B) ? (A) : (B))
```

由上可以知道，函数宏可以简单的将公式展开，这样能节省工作量么，如果仅仅如此，当然不行，不过如果运用好#与##符号就能节省大量的工作。

#与##的使用

会将宏参数转换为一个字符串,简单理解就是出现在宏定义中的 # 是把跟在后面的参数转换长一个字符串

```
#define ERROR_LOG(msg) printf("error:#msg\n")
ERROR_LOG("add") => printf("error:"add"")
```

是一种分割链接方式，它的作用是先分隔，然后进行强制连接

```
#define TYPE(type, name) type name##_##type##_type
TYPE(int, a) => int a_int_type
```

如何节省工作量呢

比如有个数据结构体，对于结构体中的每一种数据都一个操作接口，常规实现方式是每个接口都实现一次，这样随着数据项的增加，工作量会直线增加，利用函数宏可以完美解决这个问题。

```
typede struct
{
    int nData1;
    int nData2;
    int nData3;
    ...
}TData;
```

```
#define DEFADDDATA(name, type) \
void Op##_##name(TData a,type data) \
{ \
    a.n##name += data;      \
}

DEFADDDATA(Data1, int) 展开后为
void Op_Data1(TData a, int data)
{
    a.nData1 += data;
}
```

按照上面所述，每增加一个数据项，只需要一次函数宏语句就可以解决，节省大量工作量。

函数宏的变长参数

宏可以像函数一样，带可变参数。语法如下：

```
#define debug(format, ...)  printf(format, __VA_ARGS__)
debug("A message") => printf("A message",)
```

展开还有问题，因为字符串后面没有逗号 解决方法：

```
#define debug(format, ...) printf(format, ##__VA_ARGS__)
```

这里，如果可变参数被忽略或为空，`##`操作将使预处理器去除掉它前面的那个逗号。

总结

宏定义是一把双刃剑，用好了节省大量工作量，并使代码结构清晰，如果不好，跳坑不断，代码结构混乱。 人生是在不断跳坑，填坑的循环中成长。加油！！