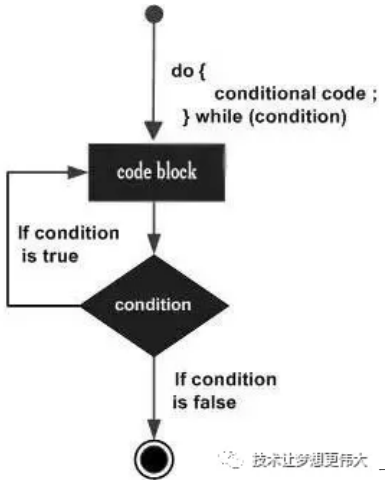
# do{}while(0)只执行一次无意义？你可能真的没理解！

*李肖遥*

在嵌入式开发中，宏定义非常强大也非常便捷，如果正确使用可以让你的工作事半功倍。然而，在很多的C程序中，你可能会看到不是那么直接的比较特殊一点的宏定义，比如`do{}while(0)`。

## do{conditional code}while(condition)结构




流程图如下: 一般结构如以下代码

```
do
{
  //循环体
}
while (条件表达式);
```

## do while/while do的区别

- do while（）

意思是先干了再说！！

- while（） do

意思是先看看能不能干!

## 初见do{...}while(0)

linux内核和其他一些开源的代码中，经常会遇到这样的代码:

```
do{
 ...
}while(0)
```

这样的代码一看就不是一个循环，do..while表面上在这里一点意义都没有，只执行一次而已，那么为什么要这么用呢?

## 总结了7种妙处

实际上，do{...}while(0)的作用可不止一点点，下面我列举了一些。

1. 有时候只是为了代码分块，比仅仅使用{}更直观些。例如在`cocos2d-x`代码中

```
do
{
    CCImage* pImage = new CCImage();
    CC_BREAK_IF(NULL == pImage);
    bRet = pImage->initWithString(text, (int)dimensions.width, (int)dimensions.height, eAlign, fontName, (int)fontSize);
    CC_BREAK_IF(!bRet);
    bRet = initWithImage(pImage);
    CC_SAFE_RELEASE(pImage);
} while (0);
```

2. 为了宏展开的时候不会出错。如果直接放在花括号里会出错的

举例来说，假设你需要定义这样一个宏：

```
#define DOSOMETHING() action1(); action2();
```

这个宏的本意是，当执行DOSOMETHING()时，action1(),action2()都会被调用。如果有判断，再执行这个宏的话，如下：

```
if(NULL == pPointer)
  DOSOMETHING();
else
  ...
```

这样宏在预处理的时候会直接被展开，放在花括号里，那么实际上写的代码如下：

```
if(NULL == pPointer)
    action1();
action2();
else
  ...
```

这展开存在两个问题：

- 因为if分支后面有两个语句，导致else分支没有对应的if，编译失败。
- 假设没有else分支，则DOSOMETHING中的第二个语句无论if测试是否通过，都会执行。

那么仅仅使用{}把action1()、action2()包起来行么？比如：

```
#define DOSOMETHING() { action1(); action2(); }
```

我们在写代码的时候都习惯在语句右面加上分号，如果在宏中使用{}，代码编译展开后宏就相当于这样写了：{...};，展开后如下：

```
if(NULL == pPointer)
{
    action1();
    action2();
};
else
  ...
```

这段代码中大括号后多了一个分号，如果有else，那么else又没有对应的if了,编译出错。

## 那么办法来了

如果我们使用do{...}while(0)来定义宏，即：

```
#define DOSOMETHING() \
        do{ \
          action1();\
          action2();\
        }while(0)\
```

宏被展开后，上面的调用语句会保留初始的语义，同时绝大部分编译器都能够识别do{...}while(0)这种无用的循环并进行优化，不会导致性能优化的降低。

## 小结

在Linux内核和驱动代码还有cocos2d-x中，很多宏实现都使用do{...}while(0)来包裹他们的逻辑，Google的Robert

Love（*先前从事Linux内核开发*）给我们解答如下：

> “
>
> 让你定义的宏总是以相同的方式工作，不管在调用代码中怎么使用分号和大括号，而该宏总能确保其行为是一致的。

3. 当你执行一段代码到一半，想跳过剩下的一半的时候，如果你正处于`do{...}while(0)`循环中，则能用break达到这个目的。

```
do
{
  执行.
  再执行…
  if (如果有什么条件满足)
  {
    我想跳到另外一段代码了，剩下的不执行了，可是不建议用goto语句，怎么办呢?
     break;/*搞定*/
  }
  我有可能被执行.
}while(false)
```

举个例子如下

```
do
{
  if(!a) break;
  //do something here
  if(!b) break;
  //do another thing here
}while(0);
```

4. 变形的goto，有些公司不让用goto。在一些函数中，需要实现条件转移，或者构成循环，跳出循环体，使用goto总是一种简单的方法，例如：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   char *str;

    /* 最初的内存分配 */
   str = (char *) malloc(15);
   if(str != NULL)
     goto loop;

    printf("hello world\n");

 loop:
   printf("malloc success\n");

   return(0);
}
```

但由于goto不符合软件工程的结构化，而且有可能使得代码难懂，所以很多人都不倡导使用，这个时候我们可以使用`do{...}while(0)`来做同样的事情：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  do{
      char *str;

      /* 最初的内存分配 */
      str = (char *) malloc(15);
      if(str != NULL)
       break;

      printf("hello world\n");
  }while(0);

   printf("malloc success\n");

   return(0);
}
```

这里将函数主体部分使用do{...}while(0)包含起来，使用break来代替goto，后续的清理工作在while之后，现在既能达到同样的效果，而且代码的可读性、可维护性都要比上面的goto代码好的多了。

5. 可以是兼容各种编译器

```
int a;
a = 10;
int b;
b = 20;
```

这种代码在只支持c89的编译器上是编译不过去的，比如ADS 2.0。

```
int a;
a = 10;
do
{
   int b;
   b = 20;
}while(0);
```

6. 避免由宏引起的警告内核中由于不同架构的限制，很多时候会用到空宏。在编译的时候，这些空宏会给出警告，为了避免这样的warning，我们可以使用do{...}while(0)来定义空宏：

```
#define DOSOMETHING() do{}while(0)
```

7. 定义单一的函数块来完成复杂的操作

如果你有一个复杂的函数，变量很多，而且你不想要增加新的函数，可以使用do{...}while(0)，将你的代码写在里面，里面可以定义变量而不用考虑变量名会同函数之前或者之后的重复，例如

```
int key;
string value;
int func()
{
    int key = GetKey();
    string value = GetValue();
    dosomething for key,value;
    do{
        int key;string value;
        dosomething for this key,value;
    }while(0);
}
```

但是为了代码的可读性，尽量声明不同的变量名，以便于后续开发人员欣赏

本文授权转载自公众号“技术让梦想更伟大”，作者李肖遥

**-END-**

推荐阅读

【01】C语言、嵌入式中几个非常实用的宏技巧
【02】阅读Linux内核时，有哪些常见宏？
【03】从宏观的角度讲解U-Boot的设备管理框架
【04】图文并茂，一次搞定C语言结构体内存对齐！（包含完整源码）
【05】C语言/C++基本语句编程风格