**FreeRTOS系列第4篇---FreeRTOS编码标准及风格指南**

2015-11-26 16:38:26 [zhzht19861011](https://me.csdn.net/zhzht19861011) 阅读数 24378更多

分类专栏： [FreeRTOS基础篇](https://blog.csdn.net/zhzht19861011/article/category/5950097) [FreeRTOS基础篇](https://blog.csdn.net/zhzht19861011/article/category/9265276)

版权声明：本文为博主原创文章，遵循 [CC 4.0 BY-SA](http://creativecommons.org/licenses/by-sa/4.0/) 版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/zhzht19861011/article/details/50057531>

**1．编码标准**

      FreeRTOS的核心源代码遵从MISRA编码标准指南。这个标准篇幅稍长，你可以在MISRA官方网站花少量钱买到，这里不再复制任何标准。

      FreeRTOS源代码不符合MISRA标准的项目如下所示：

* 有两个API函数有多个返回点。MISRA编码标准强制规定：一个函数在其结尾应该有单一的返回点。
* 指针算数运算，在创建任务时，为了兼容8、16、20、24、32位总线，不可避免的使用了指针算数运算。MISRA编码标准强制规定：指针的算术运算只能用在指向数组或数组元素的指针上。
* 默认情况下，跟踪宏为空语句，因此不符合MISRA的规定。MISRA编码标准强制规定：预处理指令在句法上应该是有意义的。

      FreeRTOS可以在很多不同编译器中编译，其中的一些编译器比同类有更高级特性。因为这个原因，FreeRTOS不使用任何非C语言标准的特性或语法。一个例外情况是头文件stdint.h。在文件夹FreeRTOS/Source/include下包含一个叫做stdint.readme的文件，如果你的编译器不提供stdint类型定义，可以将stdint.readme文件重命名为stdint.h。

**2命名规则**

      RTOS内核和演示例程源代码使用以下规则：

      1> 变量

* uint32\_t类型的变量使用前缀ul，这里’u’表示’unsigned’，’l’表示’long’
* uint16\_t类型的变量使用前缀us，这里’u’表示’unsigned’，’s’表示’short’
* uint8\_t类型的变量使用前缀uc，这里’u’表示’unsigned’，’c’表示’char’
* 非stdint类型的变量使用前缀x，比如基本的Type\_t和TickType\_t类型，这些类型在移植层定义，定义成符合处理器架构的最高效类型；
* 非stdint类型的无符号变量使用前缀ux，比如UbaseType\_t（unsigned BaseType\_t）
* size\_t类型的变量使用前缀x；
* enum枚举类型变量使用前缀e
* 指针类型变量在类型基础上附加前缀p，比如指向uint16\_t的指针变量前缀为pus
* 与MISRA指南一致，char类型变量仅被允许保存ASCII字符，前缀为c
* 与MISRA指南一致，char \*类型变量仅允许指向ASCII字符串，前缀为pc

      2> 函数

* 在文件作用域范围的函数前缀为prv
* API函数的前缀为它们的返回类型，当返回为空时，前缀为v
* API函数名字起始部分为该函数所在的文件名。比如vTaskDelete函数定义在tasks.c，并且该函数返回空。

      3> 宏

* 宏的名字起始部分为该宏定义所在的文件名的一部分。比如configUSE\_PREEMPTION定义在FreeRTOSConfig.h文件中。
* 除了前缀，宏剩下的字母全部为大写，两个单词间用下划线（’\_’）隔开。

**3数据类型**

      只有stdint.h和RTOS自己定义的数据类型可以使用，但也有例外情况，如下所示：

* char：与MISRA编码标准指南一致，char类型变量仅被允许保存ASCII字符
* char \*：与MISRA编码标准指南一致，char \*类型变量仅允许指向ASCII字符串。当标准库函数期望一个char \*参数时，这样做可以消除一些编译器警告；特别是考虑到有些编译器将char类型当做signed类型，还有些编译器将char类型当做unsigned类型。

      有三种类型会在移植层定义，它们是：

* TickType\_t：如果configUSE\_16\_BIT\_TICKS为非零（条件为真），TickType\_t定义为无符号16位类型。如果configUSE\_16\_BIT\_TICKS为零（条件为假），TickType\_t定义为无符号32位类型。注：32位架构的微处理器应设置configUSE\_16\_BIT\_TICKS为零。
* BaseType\_t：定义为微处理器架构效率最高的数据类型。比如，在32位架构处理器上，BaseType\_t应该定义为32位类型。在16位架构处理器上，BaseType\_t应该定义为16位类型。如果BaseType\_t定义为char，对于函数返回值一定要确保使用的是signed char，否则可能造成负数错误。
* UbaseType\_t：这是一个无符号BaseType\_t类型

3.4风格指南

* 缩进：缩进使用制表符，一个制表符等于4个空格。
* 注释：注释单行不超过80列，特殊情况除外。不使用C++风格的双斜线（//）注释
* 布局：FreeRTOS的源代码被设计成尽可能的易于查看和阅读。下面的代码片中，第一部分展示文件布局，第二部分展示C代码设计格式。

1. /\* 首先在这里包含库文件... \*/
2. #include <stdlib.h>
4. /\* ...然后是FreeRTOS的头文件... \*/
5. #include "FreeRTOS.h"
7. /\* ...紧接着包含其它头文件. \*/
8. #include "HardwareSpecifics.h"
10. /\* 随后是#defines, 在合理的位置添加括号. \*/
11. #define A\_DEFINITION ( 1 )
13. /\*
14. \* 随后是Static (文件内部的)函数原型,
15. \* 如果注释有多行，参照本条注释风格---每一行都以’\*’起始.
16. \*/
17. static void prvAFunction( uint32\_t ulParameter );
19. /\* 文件作用域变量（本文件内部使用）紧随其后，要在函数体定义之前. \*/
20. static BaseType\_t xMyVariable.
22. /\* 每一个函数的结束都有一行破折号，破折号与下面的第一个函数之间留一行空白。\*/
24. /\*-----------------------------------------------------------\*/
26. void vAFunction( void )
27. {
28. /\* 函数体在此定义，注意要用大括号括住 \*/
29. }
30. /\*-----------------------------------------------------------\*/
32. static UBaseType\_t prvNextFunction( void )
33. {
34. /\* 函数体在此定义. \*/
35. }
36. /\*-----------------------------------------------------------\*/
38. /\*
39. \* 函数名字总是占一行，包括返回类型。 左括号之前没有空格左括号之后有一个空格，
40. \* 每个参数后面有一个空格参数的命名应该具有一定的描述性.
41. \*/
42. void vAnExampleFunction( long lParameter1, unsigned short usParameter2 )
43. {
44. /\* 变量声明没有缩进. \*/
45. uint8\_t ucByte;
47. /\* 代码要对齐. 大括号占独自一行. \*/
48. for( ucByte = 0U; ucByte < fileBUFFER\_LENGTH; ucByte++ )
49. {
50. /\* 这里再次缩进. \*/
51. }
52. }
54. /\*
55. \* for、while、do、if结构具有相似的模式。这些关键字和左括号之间没有空格。
56. \* 左括号之后有一个空格，右括号前面也有一个空格，每个分号后面有一个空格。
57. \* 每个运算符的前后各一个空格。使用圆括号明确运算符的优先级。不允许有0
58. \* 以外的数字（魔鬼数）出现，必要时将这些数字换成能表示出数字含义的常量或
59. \* 宏定义。
60. \*/
61. for( ucByte = 0U; ucByte < fileBUFFER\_LENGTH; ucByte++ )
62. {
63. }
65. while( ucByte < fileBUFFER\_LENGTH )
66. {
67. }
69. /\*
70. \* 由于运算符优先级的复杂性，我们不能相信自己对运算符优先级时刻保持警惕
71. \* 并能正确的使用，因此对于多个表达式运算时，使用括号明确优先级顺序
72. \*/
73. if( ( ucByte < fileBUFFER\_LENGTH ) && ( ucByte != 0U ) )
74. {
75. ulResult = ( ( ulValue1 + ulValue2 ) - ulValue3 ) \* ulValue4;
76. }
78. /\* 条件表达式也要像其它代码那样对齐。 \*/
79. #if( configUSE\_TRACE\_FACILITY == 1 )
80. {
81. /\* 向TCB增加一个用于跟踪的计数器. \*/
82. pxNewTCB->uxTCBNumber = uxTaskNumber;
83. }
84. #endif
86. /\*方括号前后各留一个空格\*/
87. ucBuffer[ 0 ] = 0U;
88. ucBuffer[ fileBUFFER\_LENGTH - 1U ] = 0U;