

# 编程规范

## 1.4 总体原则

- 清晰,易于维护、易于重构。
- 简洁,易于理解,并且易于实现。
- 风格统一,代码整体风格保持一致。
- 通用性,遵循业界通用的编程规范。

## 1.4 命名

- 使用驼峰风格进行命名,此风格大小写字母混用,不同单词间通过单词首字母大写来分开,具体规则如下:

类型	命名风格	形式
函数,自定义的类型	大驼峰,或带有模块前缀的大驼峰	AaaBbb, XXX_AaaBbb
局部变量,函数参数,宏参数,结构体成员,联合体成员	小驼峰	aaaBbb
全局变量	带'g_'前缀的小驼峰	g_aaaBbb
宏,枚举值	全大写并下划线分割	AAA_BBB
内核头文件中防止重复包含的宏变量	带'_LOS'前缀和'H'后缀,中间为大写模块名,以下划线分割	_LOS_MODULE_H

- 全局函数、全局变量、宏、类型名、枚举名的命名,应当准确描述并全局唯一。
- 在能够准确表达含义的前提下,局部变量,或结构体、联合体的成员变量,其命名应尽可能简短。

## 1.4 排版与格式

- 程序块采用缩进风格编写,使用空格而不是制表符('t')进行缩进,每级缩进为4个空格。
- 在两个以上的关键字、变量、常量进行对等操作时,它们之间的操作符之前、之后或者前后要加空格;进行非对等操作时,如果是关系密切的立即操作符(如< >),后不应加空格

采用这种松散方式编写代码的目的是使代码更加清晰。

在已经非常清晰的语句中没有必要再留空格,如括号内侧(即左括号后面和右括号前面)不需要加空格,多重括号间不必加空格,因为在C语言中括号已经是最清晰的标志了。在长语句中,如果需要加的空格非常多,那么应该保持整体清晰,而在局部不加空格。给操作符留空格时不要连续留两个以上空格。

正确示例:

1、逗号、分号只在后面加空格。

```
1 int a, b, c;
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

3、"!","~","+","-","&"(地址操作符)等单目操作符前后不加空格。

```
1 *p = 'a'; // 内容操作"*"与内容之间
2 flag = !is_empty; // 非操作"!"与内容之间
3 p = &mem; // 地址操作"&" 与内容之间
4 i++;
```

4、">","<","="前后不加空格。

```
1 p->id = pid; // ">"指针前后不加空格
```

5、if、for、while、switch等与后面的括号间应加空格,使if等关键字更为突出、明显。

```
1 if (a >= b && c > d)
```

6、注释符(包括/\*\*/、//)与注释内容之间要用一个空格进行分隔。

- 采用K&R风格作为大括号换行风格,即函数左大括号另起一行放行首,并独占一行,其他左大括号跟随语句放行末,右大括号独占一行,除非后面跟着同一语句的剩余部分,如if语句的else/else if或者分号,比如:

```
1 struct MyType { // 左大括号跟随语句放行末,前置1个空格
2     ...
3 }; // 右大括号后面紧跟分号
1 int Foo(int a)
2 { // 函数左大括号独占一行,放行首
3     if (a > 0) { // 左大括号跟随语句放行末,前置1个空格
4         ...
5     } else { // 右大括号、"else"、以及后续的左大括号均在同一行
6         ...
7     } // 右大括号独占一行
8     ...
9 }
```

- 条件、循环语句使用大括号,比如:

```
1 if (objectIsNotExist) { // 单行条件语句也加大括号
2     return CreateNewObject();
3 }
1 while (condition) {} // 即使循环体是空,也应使用大括号
1 while (condition) {
2     continue; // continue表示空逻辑,使用大括号
3 }
```

- case/default语句相对switch缩进一层,缩进风格如下:

```
1 switch (var) {
2     case 0: // 缩进一层
3         DoSomething1(); // 缩进一层
4         break;
5     case 1:
6         DoSomething2();
7         break;
8     default:
9         break;
10 }
```

- 一行只写一条语句。
- 一条语句不能过长,建议不超过120个字符,如不能缩短语句则需要分行写。
- 换行时将操作符留在行末,新行进行同类对齐或缩进一层,比如:

```

1 // 假设下面第一行不满足行宽要求
2 if (currentValue > MIN && // 换行后,布尔操作符放在行末
3     currentValue < MAX) { // 与(&&)操作符的两个操作数同类对齐
4     DoSomething();
5     ...
6 }

1 // 假设下面的函数调用不满足行宽要求,需要换行
2 ReturnType result = FunctionName(paramName1,
3                                   paramName2,
4                                   paramName3); // 保持与上方参数对齐

1 ReturnType result = VeryVeryVeryLongFunctionName( // 写入第1个参数后导致过长,直接换行
2     paramName1, paramName2, paramName3);          // 换行后,4空格缩进一层

1 // 每行的参数代表一组相关性较强的数据结构,放在一行便于理解,此时可理解性优先于格式排版要求
2 int result = DealWithStructLikeParams(left.x, left.y, // 表示一组相关参数
3                                         right.x, right.y); // 表示另外一组相关参数

```

- 声明定义函数时,函数的返回类型以及其他修饰符,与函数名同行。
- 指针类型"\*"应该靠右跟随变量或者函数名,比如:

```

1 int *p1; // Good:右跟随变量,和左边的类型隔了1个空格
2 int* p2; // Bad:左跟随类型
3 int*p3; // Bad:两边都没空格
4 int * p4; // Bad:两边都有空格

```

当"\*"与变量或函数名之间有其他修饰符,无法跟随时,此时也不要跟随修饰符,比如:

```

1 char * const VERSION = "V100"; // Good:当有const修饰符时,"*"两边都有空格
2 int Foo(const char * restrict p); // Good:当有restrict修饰符时,"*"两边都有空格

```

- 根据上下内容的相关程度,合理安排空行,但不要使用连续3个或更多空行。
- 编译预处理的"#"统一放在行首,无需缩进。嵌套编译预处理语句时,"#"可以进行缩进,比如:

```

1 #if defined(__x86_64__) && defined(__GCC_HAVE_SYNC_COMPARE_AND_SWAP_16) // 位于行首,不缩进
2     #define ATOMIC_X86_HAS_CMPXCHG16B 1 // 缩进一层,区分层次,便于阅读
3 #else
4     #define ATOMIC_X86_HAS_CMPXCHG16B 0
5 #endif

```

## J4 注释

- 注释的内容要清楚、明了,含义准确,防止注释二义性。
- 在代码的功能、意图层次上进行注释,即注释解释代码难以直接表达的意图,而不是仅仅重复描述代码。
- 函数声明处注释描述函数功能、性能及用法,包括输入和输出参数、函数返回值、可重入的要求等;定义处详细描述函数功能和实现要点,如实现的简要步骤、实现的理由、设计约束等。
- 全局变量要有较详细的注释,包括对其功能、取值范围以及存取时注意事项等的说明。
- 避免在注释中使用缩写,除非是业界通用或子系统内标准化的缩写。
- 文件头部要进行注释,建议注释列出:版权说明、版本号、生成日期、作者姓名、功能说明、与其它文件的关系、修改日志等。
- 注释风格要统一,建议优先选择/\* \*/的方式,注释符与注释内容之间要有1空格,单行、多行注释风格如下:

```

1 /* 单行注释 */

1 /*
2  * 多行注释
3  * 第二行
4  */

```

- 注释应放在其代码上方或右方。

上方的注释,与代码行之间无空行,保持与代码一样的缩进。右边的注释,与代码之间至少相隔1个空格。如果有多条右置注释,上下对齐会更加美观,比如:

```
1 | #define A_CONST 100           // 此处两行注释属于同类
2 | #define ANOTHER_CONST 200    // 可保持左侧对齐
```

## ⚡ 宏

- 代码片段使用宏隔离时,统一通过`#ifdef`的方式,例如:

```
1 | #ifdef LOSCFG_XXX
2 | ...
3 | #endif
```

- 定义宏时,要使用完备的括号,比如:

```
1 | #define SUM(a, b) a + b       // 不符合本条要求
2 | #define SUM(a, b) ((a) + (b)) // 符合本条要求
```

但是也要避免滥用括号,比如单独的数字或标识符加括号毫无意义:

```
1 | #define SOME_CONST 100        // 单独的数字无需括号
2 | #define ANOTHER_CONST (-1)   // 负数需要使用括号
3 | #define THE_CONST SOME_CONST // 单独的标识符无需括号
```

- 包含多条语句的函数式宏的实现语句必须放在`do-while(0)`中,例如:

```
1 | #define F00(x) do { \
2 |     (void)printf("arg is %d\n", (x)); \
3 |     DoSomething((x)); \
4 | } while (0)
```

- 禁止宏调用参数中出现预编译指令。
- 宏定义不以分号结尾。

## ⚡ 头文件

- 设计原则
  - 头文件应当职责单一。
  - 一个模块通常包含多个.c文件,建议放在同一个目录下,目录名即为模块名;如果一个模块包含多个子模块,则建议每一个子模块提供一个对外的.h,文件名为子模块名。
  - 建议每一个.c文件应有一个同名.h文件,用于声明需要对外公开的接口。
  - 头文件中适合放置接口的声明,不适合放置实现。
  - 不要在头文件中定义变量。
  - 禁止头文件循环依赖,循环依赖指a.h包含b.h, b.h包含c.h, c.h包含a.h。
  - 头文件应当自包含,即任意一个头文件均可独立编译,但同时也要避免包含用不到的头文件。
  - 头文件必须用`#define`保护,防止重复包含,比如内核中统一使用以下宏定义保护:

```
1 | #ifndef _LOS_<MODULE>_H // 比如 _LOS_TASK_H
2 | #define _LOS_<MODULE>_H
3 | ...
4 | #endif
```

- 禁止通过声明的方式引用外部函数接口、变量, 只能通过包含头文件的方式使用其他模块或文件提供的接口。
- 禁止在 `extern "C"` 中包含头文件。
- 按照合理的顺序包含头文件:
  1. 源文件对应的头文件
  2. C标准库
  3. 需要包含的OS其他头文件

## ♪4 变量

- 一个变量只有一个功能, 不要把一个变量用作多种用途。
- 防止局部变量与全局变量同名。
- 不用或者少用全局变量。
- 定义函数的局部变量时, 控制变量的占用空间, 避免因占用过多栈空间导致程序运行失败。比如需要一个大数组, 可以通过动态分配内存的方式来避免栈空间占用过大。
- 在首次使用前初始化变量。
- 指向资源句柄或描述符的变量, 在资源释放后立即赋予新值, 包括指针、socket描述符、文件描述符以及其它指向资源的变量。
- 禁止将局部变量的地址返回到其作用域以外, 下面是一个错误示例:

```

1  int *Func(void)
2  {
3      int localVar = 0;
4      ...
5      return &localVar;  // 错误
6  }
7  void Caller(void)
8  {
9      int *p = Func();
10     ...
11     int x = *p;          // 程序产生未定义行为
12 }
```

正确代码示例:

```

1  int Func(void)
2  {
3      int localVar = 0;
4      ...
5      return localVar;
6  }
7  void Caller(void)
8  {
9      int x = Func();
10     ...
11 }
```

- 如果要使用其他模块的变量, 应尽量避免直接对变量进行访问, 而是通过统一的函数封装或者宏封装的方式, 比如mutex模块中:

```

1  // 私有头文件中引入全局变量, 但要避免直接使用
2  extern LosMuxCB *g_allMux;
3  // 通过GET_MUX的方式对g_allMux进行访问
4  #define GET_MUX(muxID) (((LosMuxCB *)g_allMux) + GET_MUX_INDEX(muxID))
```

## ♪<sup>4</sup> 函数

- 重复代码应该尽可能提炼成函数。
- 避免函数过长, 新增函数不超过 40-50 行。
- 内联函数要尽可能短, 避免超过 10 行 (非空非注释)。
- 避免函数的代码块嵌套过深。
- 函数应避免使用全局变量、静态局部变量和I/O操作, 不可避免的地方应集中使用。