TBQ C++编码规范

目录

[TBQ C++编码规范 1](#_Toc448165152)

[1、文件规范 2](#_Toc448165153)

[1.1 编码 2](#_Toc448165154)

[1.2 命名 3](#_Toc448165155)

[1.2.1 文件命名 3](#_Toc448165156)

[1.2.2 目录命名 3](#_Toc448165157)

[1.2.3 目录结构 3](#_Toc448165158)

[1.2.4 VC解决方案命名 4](#_Toc448165159)

[1.2.5 VC项目命名/ Qt项目命名 4](#_Toc448165160)

[1.2.6 头文件 4](#_Toc448165161)

[2、排版规范 4](#_Toc448165162)

[2.1 缩进 4](#_Toc448165163)

[2.2代码行 5](#_Toc448165164)

[3、命名规范 7](#_Toc448165165)

[3.1命名总则 7](#_Toc448165166)

[3.2文件名 8](#_Toc448165167)

[3.3命名空间 8](#_Toc448165168)

[3.4类、接口 8](#_Toc448165169)

[3.5方法 8](#_Toc448165170)

[3.6参数 9](#_Toc448165171)

[3.7属 10](#_Toc448165172)

[3.8常量和宏 10](#_Toc448165173)

[3.9变量 10](#_Toc448165174)

[3.10结构体、类型、枚举 10](#_Toc448165175)

[3.11关键字 11](#_Toc448165176)

[4、语句规范 11](#_Toc448165177)

[4.1 return 11](#_Toc448165178)

[4.2 if else 11](#_Toc448165179)

[4.3 循环语句 12](#_Toc448165180)

[4.3.1 for语句 12](#_Toc448165181)

[4.3.2 while与do while 12](#_Toc448165182)

[4.4 switch-case语句 13](#_Toc448165183)

[4.5 try-catch语句 14](#_Toc448165184)

[4.6 using语句，include文件包含引用顺序 14](#_Toc448165185)

[4.7 goto语句 14](#_Toc448165186)

[4.8 指针 14](#_Toc448165187)

[4.9 类型转换 14](#_Toc448165188)

[4.10 表达式 15](#_Toc448165189)

[4.11 声明语句 15](#_Toc448165190)

[4.12其他 16](#_Toc448165191)

[5、注释规范 16](#_Toc448165192)

[5.1注释总则 16](#_Toc448165193)

[5.2文件头部注释 17](#_Toc448165194)

[5.3数据结构注释 17](#_Toc448165195)

[5.4函数注释 18](#_Toc448165196)

[5.5代码注释 18](#_Toc448165197)

[5.6变量及常量注释 20](#_Toc448165198)

[6、附录 22](#_Toc448165199)

[6.1类型前缀列表 22](#_Toc448165200)

[6.2 C++关键字表 23](#_Toc448165201)

# 1、文件规范

## 1.1 编码

(1)文件编码：统一使用ANSI。

(2)字符编码：统一使用Unicode，特殊情况除外。

## 1.2 命名

### 1.2.1 文件命名

(1)文件名的每个单词首字母大写，禁止使用下划线。

(2)C++文件以.cpp结尾，头文件以.h结尾，内部使用的头文件可以用.hpp结尾。

(3)不要使用系统库中已存在的文件名，文件名需要有各自项目的前缀，例如TBQProject.h

### 1.2.2 目录命名

目录名的每个单词首字母大写，目录命名以产品名称+功能名称,例如：TBQClient。

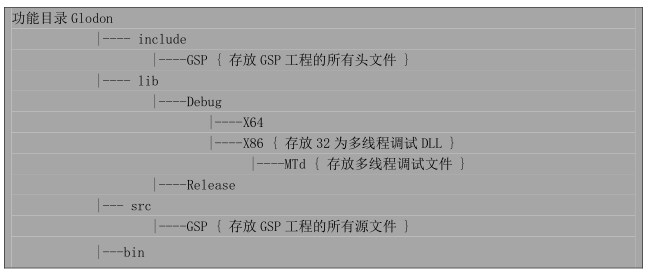
### 1.2.3 目录结构

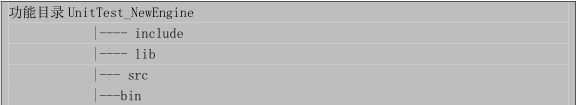
(1)头文件存放目录：头文件统一放在include目录下，不同的工程在include文件中用子目录区分，include目录下只包含一层子目录，即同一个工程的所有头文件统一平级放在该工程文件目录下。

(2)源文件存放目录：源文件统一放在src目录下，不同的工程在src文件中用子目区分。

(3)输出文件存放目录：lib目录下包含Debug和Release两个子目分别存放调试和发布版本的lib文件，在Debug和Release目录下包含X64和X86子目录，X86和X64下包含MTd子目录，运行库为MDd(多线程调试DLL)文件直接放在X64或者X86目录下，运行库为MTd(多线程调试)文件存放在MTd目录。

示例如下：





### 1.2.4 VC解决方案命名

命名的每个单词首字母大写，格式为“产品名称\_编译器版本名”或者“产品名称”，可以不加入编译器版本名。

### 1.2.5 VC项目命名/ Qt项目命名

命名的每个单词首字母大写，以库名称命名，如lib、exe、dll的名称。

### 1.2.6 头文件

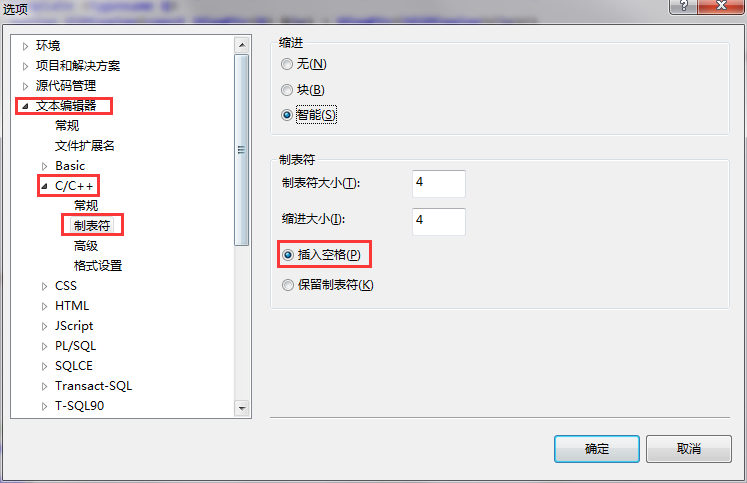
添加头文件包含顺序如下：OS SDK .h、C标准库、C++标准库、OpenGL的头文件、其它库的头文件、自己工程的头文件。

# 2、排版规范

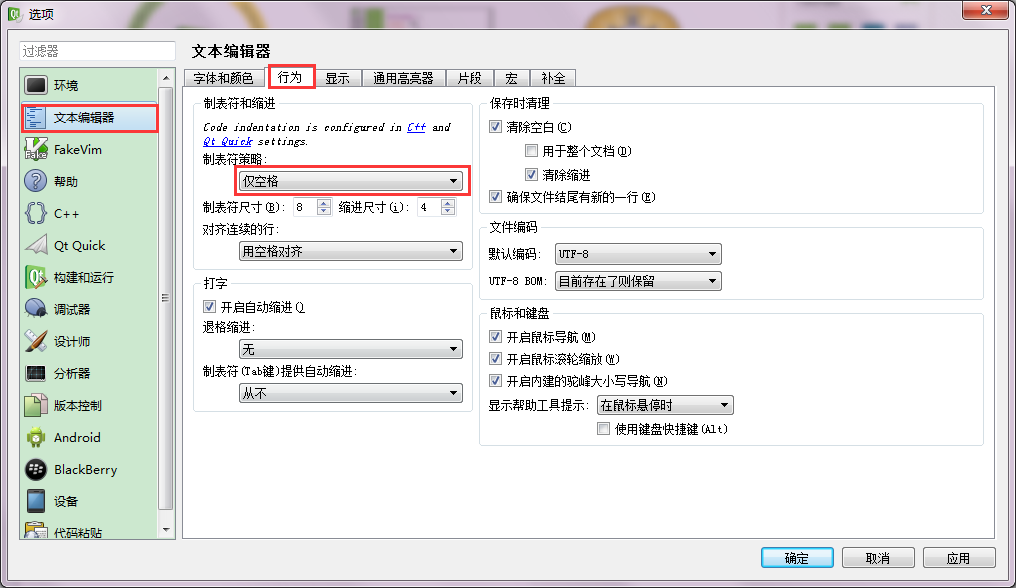
## 2.1 缩进

(1)程序块需要采用缩进风格编写，这样便于浏览代码结构。所有缩进规定使用4个空格字符，请不要使用Tab字符进行缩进，因为在不同的编辑器下Tab缩进可能会产生差异，导致排版混乱。

VS2010可以在[工具]-[选项]中设置制表符



Qt Creator可以在[工具]-[选项]中设置制表符



(2) 不要使用BC作为编辑器合版本，因为BC会自动将8个空格变为一个TAB键，因此使用BC合入的版本大多会将缩进变乱。

(3)函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case语句下的情况处理语句也要遵从语句缩进要求。

## **2.2代码行**

(1)换行：一行代码只完成一个功能，例如定义一个变量，或执行一条语句。这样的代码容易阅读并方便进行注释。

(2)相对独立的程序块之间、变量之间加空行进行分割。

下面的代码举例说明。

|  |  |
| --- | --- |
| 推荐 | double dLength = 0.0;  **// length**  double dWidth = 0.0; **// width**  double dArea = 0.0; **// area**  dArea = dLength \* dWidth; |
| 不推荐 | **// qty...**  int nQty = 0; double dRate = 0.0, dAmt = 0.0;  dAmt = nQty \* dRate; |

示例 2.1

(3)if、for、do、while、case、switch、default等语句自占一行，且if、for、do、while等。语句的执行语句部分无论多少都要加括号{}。

|  |  |
| --- | --- |
| 推荐 | for (int i = 0; i < nCount; ++i)  {  **// program code**  }  if (nCount < DEF\_MAX\_COUNT)  {  **// program code**  } |
| 不推荐 | for (int i = 0; i < nCount; ++i){  **// program code**  }  if (nCount < DEF\_MAX\_COUNT)  **// program code** |

示例 2.2

(4)较长的语句(>80字符)要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要缩进4个空格，使排版整齐，语句可读。

|  |  |
| --- | --- |
| 推荐 | perm\_count\_msg.head.len = NO7\_TO\_STAT\_PERM\_COUNT\_LEN \  + STAT\_SIZE\_PER\_FRAM \* sizeof(\_UL); |

示例 2.3

(5)循环、判断等语句中若有较长的表达式或语句，则要进行适应的划分，长表达式要在低优先级操作符处划分新行，新行要缩进4个空格，操作符放在新行之首。

|  |  |
| --- | --- |
| 推荐 | if (taskno < max\_act\_task\_number  && n7stat\_stat\_item\_valid(stat\_item)  && newKeyword.word\_length < DEF\_MAX\_LENGTH)  {  **// program code**  } |
| 不推荐 | if (taskno < max\_act\_task\_number  && n7stat\_stat\_item\_valid(stat\_item)  && newKeyword.word\_length < DEF\_MAX\_LENGTH)  {  **// program code**  } |

示例 2.4

(6)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。

说明：采用这种松散方式编写代码的目的是使代码更加清晰。由于留空格所产生的清晰性是相对的，所以，在已经非常清晰的语句中没有必要再留空格，如果语句已足够清晰则括号内侧(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/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。

1. 在逗号后加空格。

|  |  |
| --- | --- |
| 示例 | int nA, nB, nC; |

示例 2.5

1. 比较操作符, 赋值操作符"="、 "+="，算术操作符"+"、"%"，逻辑操作符"&&"、"&"，位域操作符"<<"、"^"等双目操作符的前后加空格。

|  |  |
| --- | --- |
| 示例 | if (current\_time >= MAX\_TIME\_VALUE)  a = b + c;  a \*= 2;  a = b ^ 2; |

示例 2.6

1. "!"、"~"、"++"、"--"、"&"(地址运算符)等单目操作符前后不加空格。

|  |  |
| --- | --- |
| 示例 | \*p = 'a'; **// 内容操作"\*"与内容之间**  flag = !isEmpty;  **// 非操作"!"与内容之间**  p = &mem;  **// 地址操作"&" 与内容之间**  i++;  **// "++","--"与内容之间** |

示例 2.7

1. 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。

|  |  |
| --- | --- |
| 示例 | if (a >= b && c > d) |

示例 2.8

# 3、命名规范

## 3.1命名总则

(1)避免使用不易理解的数字，用有意义的标识来替代。 应当符合“min-length && max-information”原则。

(2)涉及物理状态或者含有物理意义的常量，不要直接使用数字，必须用有意义的枚举或宏来代替。如果是true和false可以直接用，如果本意就是数字，允许直接用数字表示。

(3)尽量避免缩写，如果使用缩写，则要使用通用的、众所周知的缩写。

(4)类型前缀说明见附表6-1。

## 3.2文件名

(1)以大写字母开始并混合大小写，不含下划线，C++文件以.cpp结尾，头文件以.h结尾。

(2)文件名字要与文件中声明、定义的类的名字基本保持一致，使类名与类文件名建立联系。

(3)文件名应和类名类似，增加项目或功能相关的前缀信息，如：TBQxxx.h。

(4)窗体以Form;模板以Frame结尾

## 3.3命名空间

规定：命名空间的名称是全小写的。

## 3.4类、接口

(1)所有的类以”产品名称”开头，如TBQProject。

(2)接口必须以“I”开头，如ITBQBaseProject。

## 3.5方法

(1)方法命名一般用一个单词(一般是动词)或者一个词组(一般是一个动宾词组)组成，这些词或词组应该能够表达该函数的意义。

(2)以小写开头，后续单词使用大写做词义的划分，不能使用“\_”。

(3)存取函数：存使用set开头，如setName()；取不需要get，如name()，且要求与变量名匹配。

(4)bool类型的变量和函数返回值是bool的函数以is为前缀来命名，在某些情况下，有些比is前缀更适合(has, can, should)，如：bool isVisible(),bool hasLicense();

## 3.6参数

(1)参数首字母小写,后面单词首字母大写,如value、userName, 不用多余的前缀,如nValue、aValue等。

(1)参数的书写要完整，不要贪图省事只写参数的类型而省略参数名字。例如：

|  |  |
| --- | --- |
| **a、推荐** | **b、禁止** |
| void setValue(int width, int height);**// 良好的风格**  float value();  **// 良好的风格** | void setValue(int, int); **// 不良的风格**  float value(void);  **// 不良的风格** |

示例 3.1

(2)参数命名要恰当，顺序要合理，推荐先输出参数后输入参数。例如：

|  |  |
| --- | --- |
| **a、推荐** | **b、禁止** |
| void stringCopy(char \*strDestination, char \* strSource); | void stringCopy(char \*strSource, char \*strDestination);  void stringCopy(char \*str1, char \*str2); |
| 编写字符串拷贝函数stringCopy，它有两个参数，如果把参数名字起为str1和str2，  void stringCopy(char \*str1, char \*str2);  那么我们很难搞清楚究竟是把str1拷贝到str2中，还是刚好倒过来，可以把参数名字起得更有意义，如叫strSource和strDestination。这样从名字上就可以看出应该把strSource拷贝到strDestination。还有一个问题，这两个参数那一个该在前那一个该在后？参数的顺序要遵循程序员的习惯。一般地，应将目的参数放在前面，源参数放在后面。如果将函数声明为：  void stringCopy(char \*strSource, char \*strDestination);  别人在使用时可能会不假思索地写成如下形式，导致参数顺序颠倒：  char str[20];  stringCopy(str, “Hello World”); **// 参数顺序颠倒** | |

示例 3.2



(3)如果参数是指针，且仅作输入用，则应在类型前加const，以防止该指针在函数体内被意外修改。例如：

|  |  |
| --- | --- |
| **a、推荐** | **b、禁止** |
| void stringCopy(char\* destination, const char\* source); | void stringCopy(char \*destination, char \*source); |

示例 3.3

(4)对象作为输入参数的，使用‘&’传递，避免对象构造和析构，提高效率。

|  |  |
| --- | --- |
| **a、推荐** | **b、禁止** |
| void stringCopy(const QString &strSource); | void stringCopy(const QString& strSource); |

(5)避免函数有太多的参数，参数个数尽量控制在5个以内。如果参数太多，在使用时容易将参数类型或顺序搞错。

(6)尽量不要使用类型和数目不确定的参数。例如：

|  |
| --- |
| C标准库函数printf是采用不确定参数的典型代表，其原型为  int printf(const char\* format[, argument]);  这种风格的函数在编译时丧失了严格的类型安全检查。 |

示例 3.4

## 3.7属

C++中可以采用技巧实现类似其它语言中的属性，禁止使用这种技巧。

## 3.8常量和宏

(1)宏命名使用大写字母加下划线‘\_’分隔词义。

(2)常量命名以小写“c\_”开头。

## 3.9变量

(1)变量名以小写字母开始，范围前缀+类型前缀+表意单词，表意单词后续单词使用首字母大写做词义的划分。

(2)禁止取单个字符(如i、j、k...)，要有具体含义，但i、j、k作局部循环变量是允许的。

(3)类成员变量命名：以范围前缀“m\_”开头，如m\_strName。

(4)普通变量命名：无范围前缀，要求区分类型前缀，如nCount，strName。

(5)全局变量命名：g\_开头。

(6)静态变量命名：以s\_开头

(7)禁止定义否定的bool变量名，如：bool isNoError(错误)

(8)定义指针变量统一用这种方式：char \*a，杜绝使用char\* a;

## 3.10结构体、类型、枚举

结构体、类型定义(typedef)、枚举使用相同约定。(枚举推荐其中以Type、Kind等关键字结尾；如 TBQBillElementType)。

(1)类型定义每个单词以大写字母开头，不包含下划线。

类型前缀列表详见附录7.1

(2)所有的枚举以产品名称开头，例如：TBQBillElementType。

(3)枚举值命名方式为枚举的名称缩写(全部小写，最多3个字符)+名称。

示例如下：

|  |
| --- |
| enum TBQBillElementType  {  betNull ,  betBill ,  betElement  }; |

示例 3.4

## 3.11关键字

C++关键字表详见附录

# 4、语句规范

## 4.1 return

Return后面要加一个空格，简单的表达式不要加括号，但复杂度较高的表达式可以通过增加括号提升代码可读性。例如：

|  |  |
| --- | --- |
| a、推荐 | b、禁止 |
| **//好：圆括号提高复杂的表达式的可读性**  return (some\_long\_condition&&another\_condition);  return result; **//好：简单表达式不需要圆括号** | **// 坏：谁会写这样的表达式 var = (value);?**  return (value);  return(result); **// 坏：return不是函数** |

示例5.1

## 4.2 if else

(1)所有情况下if和左圆括号‘(’ 中间都要有个空格。

(2) else if 必须写在一行。例如：

|  |  |
| --- | --- |
| a、推荐 | b、禁止 |
| if (condition)  {  **//program code**  }  else  {  **//program code**  } | if(condition)  {  **//program code**  }  else  {  **//program code**  } |
| if (condition)  {  **//program code**  }  else if (condition)  {  **//program code**  } | if (condition)  {  **//program code**  }  else  if (condition)  {  **//program code**  } |

示例5.2

## 4.3 循环语句

(1)无论循环语句主体有几行，都必须使用花括号‘{ }’将循环主体括起来。

### 4.3.1 for语句

无论循环变量是否已经初始化，都要在循环语句中初始化。例如：（如果i是一个已经初始化的变量怎么办，所以我觉得这条规则不一定通用）

|  |  |
| --- | --- |
| a、推荐 | b、禁止 |
| int i = 0;  for (i = 0; i < kSomeNumber; ++i)  {  **//program code**  } | int i = 0;  for (; i < kSomeNumber; ++i)  {  **//program code**  } |

示例5.3

### 4.3.2 while与do while

如果while中的条件恒为真，使用‘true’而不是‘1’。例如：

|  |  |
| --- | --- |
| a、推荐 | b、禁止 |
| while (true)  {  **// Repeat**  } | while (1)  {  **// Repeat**  } |
| do  {  **//Repeat**  } while (true); | do  {  **//Repeat**  }while(1); |

示例5.4

## 4.4 switch-case语句

(1)通常情况下，每一个case代码段均需要break结尾，如果有特殊情况不适用break，需要注释说明。

(2)无论有多少个case，switch语句必须使用花括号‘{}’将所有的case括起来。

(3)无论一个case代码段中有几行语句，都必须使用花括号‘{}’将这些语句括起来；如果是多个case使用同一个代码段，则前面的case后面不需要花括号。

(4)所有的case最后，必须要有default分支。如果default确实不可能被用到，建议写一句简单的assert语句。例如：

|  |  |
| --- | --- |
| a、推荐 | b、禁止 |
| switch (var)  {  case 0:  {  **//program code**  **// 不使用break的原因**  }  case 1:  case 2:  case 3:  {  **//program code**  break;  }  default:  {  Assert(false);  Break;  }   |  |  | | --- | --- | |  |  | | switch (var)  {  case 0:  {  **//program code**  }  case 1:  case 2:  case 3:  {  **//program code**  }  default:  {  assert(false);  }  } |

示例5.5

## 4.5 try-catch语句

(1)禁止使用异常机制处理正确的逻辑，win32需要时可以采用结构化异常。

(2)如果需要使用C++异常，最终应该加一个捕获所有异常的catch(…)。例如：

|  |  |
| --- | --- |
| a、推荐 | b、禁止 |
| try  {  **//program code**  }  catch (std::exctpion \* e)  {  **//program code**  e.what();  }  catch (...)  {  assert(false);  } | try  {  **//program code**  }  catch(std::exctpion \* e)  {  **//program code**  e.what();  } |

示例5.6

## 4.6 using语句，include文件包含引用顺序

(1)引用顺序：先标准库，后自己定义的命名空间。

(2)using语句的位置紧跟在头文件后面。

## 4.7 goto语句

尽量避免使用goto语句。

## 4.8 指针

(1)指针初始化为空指针时使用NULL而不是0。

(2)在判断指针是否为空时，使用NULL而不是0。

## 4.9 类型转换

推荐使用C++ 的类型转换, 如static\_cast<>()。

优点: C 语言的类型转换问题在于模棱两可的操作；有时是在做强制转换(如(int)3.5)，有时是在作类型转换(如(int)"hello")。另外，C++的类型转换在查找时更醒目。

缺点: 恶心的语法。

(1)用static\_cast替代C风格的值转换, 或某个类指针需要明确的向上转换为父类指针时。

(2)谨慎使用以下3种类型转换：用const\_cast去掉const限定符；用reinterpret\_cast指针类型和整型或其它指针之间进行不安全的相互转换，仅在你对所做一切了然于心时使用；dynamic\_cast测试代码以外尽量不要使用，除非是单元测试，如果你需要在运行时确定类型信息，说明有设计缺陷。

## 4.10 表达式

(1)如果一个布尔表达式超过标准行宽，要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行缩进。例如：

|  |  |
| --- | --- |
| a、推荐 | b、禁止 |
| if (this\_one\_thing>this\_other\_thing  &&a\_third\_thing == a\_fourth\_thing  &&yet\_another&&last\_one)  {  **//program code**  } | if(this\_one\_thing>this\_other\_thing  &&a\_third\_thing == a\_fourth\_thing  &&yet\_another&&last\_one)  {  **//program code**  } |

示例5.7

(2)如果是常量和变量进行比较，将常量放在等于号的左侧，变量放在右侧。

(3)句点或箭头前后不要有空格，指针/地址操作符(\*，&)之后不能有空格。例如：

|  |
| --- |
| x = \*p;  p = &x;  x = r.y;  x = r->y; |

示例5.8

## 4.11 声明语句

(1)访问控制块的声明顺序依次是public：，protected：，private：。

(2)变量和成员函数不要混到一个访问控制块里声明，需要分开两个部分声明，成员函数放在前面。

(3)不修改任何成员变量时，函数声明为const。

(4)所有的基类名应在20字符限制。

(5)除第一个访问控制关键词(一般是public)外，其他访问控制关键词前要空一行。如果类比较小的话也可以不空。

(6)这些访问控制关键词后不要保留空行。

类声明(对类注释不了解的话，参考类注释)的基本格式如下：

|  |  |
| --- | --- |
| a、推荐 | b、禁止 |
| public:  MyClass();  explicit MyClass(int var);  ~MyClass() {}  void someFunction();  void someFunctionThatDoesNothing() {}  void setSomeVar(intvar) { m\_nSomeVar = var; }  int nSomeVar() const{ return m\_nSomeVar; }  private  bool someInternalFunction();  private:  int m\_nSomeVar\_;  int m\_nSomeOtherVar;  }; | public:  MyClass();  MyClass(int var);  ~MyClass() {}  void someFunction();  void someFunctionThatDoesNothing() {}  void setSomeVar(intvar) { m\_nSomeVar = var; }  int nSomeVar(){ return m\_nSomeVar; }  private:  int m\_nSomeVar\_;  int m\_nSomeOtherVar;  DISALLOW\_COPY\_AND\_ASSIGN(MyClass);  private  bool someInternalFunction();  }; |

示例5.9

## 4.12其他

连续的宏定义或者常量将=对齐。(=所处的位置尽量是34列。团队协作中，如果遵守相同的规则时，添加内容更加整齐。)。

# 5、注释规范

## 5.1注释总则

(1)// 或/\*\*/都可以，但// 更常用，要在如何注释及注释风格上确保统一。

(2)一般情况下，源程序有效注释量必须在20%以上。

(3)注释内容要清楚明了、含义准确、防止二义性。

(4)避免在注释中使用缩写，特别是非常用缩写，在使用缩写时或之前，应对缩写进行必要的说明。

(5)注释应考虑程序易读及外观排版的因素，使用语言统一为中文，对于类名、函数名、变量名和其它专用的或者不可翻译的缩写可以使用英文，出于对维护人员的考虑，建议使用中文。

(6)删除注释掉的代码，因为SVN上有历史记录，可以避免日后混淆。

(7)Doxygen风格注释

## 5.2文件头部注释

(1)头文件.h文件、.inc文件、.def文件、编译说明文件.cfg等文件的头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等信息。

(2)头文件的注释中应当包含函数功能简要说明。

(3)源文件头部应进行注释，列出：版权说明、生成日期、作者、模块目的/功能、修改日志等信息。

示例：下面这段源文件的头注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

|  |
| --- |
| /\*  \* @file 文件名称  \* @brief 文件说明  \* @author 作者  \* @date 日期  \* @remarks  \* @Copyright (c) 1998-2016 Glodon Corporation  \*/ |

示例 5.1

## 5.3数据结构注释

这里的数据结构包括数组、结构、类、枚举。

(1)如果数据结构的命名不是充分自注释的，必须加以注释。

(2)对数据结构的注释应放在其上方相邻的位置，不可放在下面；对结构中的每个域的注释放在此域的右方。

(3)如果是枚举的话则要求按照实例中给出的，设定每个枚举项的数值，防止将来因插入或者删除枚举项带来枚举值在不同版本中不一致的情况，且每个枚举项都必须附加详细的注释。

示例：

|  |
| --- |
| /\*!  \*@class function :[类功能]  \*@author :[开发者]  \*/ |

示例 5.2

|  |
| --- |
| /\*!  \*@brief :同步信号码  \*/  // Description  enum EnSynCode  {  N\_UNITDATA\_IND = 0, //sccp notify sccp user unit data come  N\_NOTICE\_IND = 1, //sccp notify user the No.7 network can not  N\_UNITDATA\_REQ = 2 //sccp user's unit data transmission request  }; |

示例 5.3

## 5.4函数注释

一般说来每个函数都应该做详细的注释。

(1)函数头的注释应写在头文件(.h文件)中，即该函数的声明处而不是实现处。

(2)函数头的注释的内容包括“功能”，“参数”，“返回值”，“设计思想”，“调用函数”，“日期”，“修改记录”等信息。

示例：下面这段函数的注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

|  |
| --- |
| /\*  \* @brief 说明  \* @returns 返回值  \* @parameter 参数  \*/ |

示例 5.4

## 5.5代码注释

对于代码中巧妙的,晦涩的,有趣的,重要的地方加以注释。

(1)边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

(2)注释应与其描述的代码相近，对代码的注释应放在其上方或右方(对单条语句的注释)相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

(3)注释与所描述内容进行同样的缩进。

|  |  |
| --- | --- |
| 1. **a、推荐** | **b、禁止** |
| otherStatement;  **// 复制对象信息**  m\_unObjID = Obj.m\_unObjID;  m\_unObjType = Obj.m\_unObjType;  return \*this; | **/\* 复制对象信息\*/**  m\_unObjID = Obj.m\_unObjID;  m\_unObjType = Obj.m\_unObjType;  return \*this; |
| m\_unObjID = Obj.m\_unObjID;  m\_unObjType = Obj.m\_unObjType;  return \*this;  **/\* 复制对象信息 \*/** |

示例 5.5

(4)对于if语句分支处，需通过注释，解释不同分支的意义。示例如下：

|  |
| --- |
| if(m\_iShortRadio == 0) **//三相的情况**  {  strvC.Format("%-10.6f", vC);  straC.Format("%-10.6f", aC);  }  else if(m\_iShortRadio == 1)  **//两相的情况**  {  strvC = \_T("");  straC = \_T("");  } |

示例 5.6

(5)对于switch语句下的case语句，如果因为特殊情况需要处理完一个case后进入下一个case处理，必须在该case语句处理完、下一个case语句前加上明确的注释(同5.5中switch-case规范)。示例如下：

|  |
| --- |
| switch(var)  {  case CMD\_DOWN:  {  processDown();  break;  }  case CMD\_FWD:  {  processFwd();  if (...)  {  //program code  break;  }  else  {  processCFW\_B(); // now jump into case CMD\_A  }  }  case CMD\_A:  {  processA();  break;  }  } |

示例 5.7

## 5.6变量及常量注释

(1)对变量的定义和分支语句(条件分支、循环语句等)必须编写注释。

(2)类数据成员：每个类数据成员(也叫实例变量或成员变量)都应该用注释说明用途。如果变量可以接受NULL或-1等警戒值,须加以说明。示例如下:

|  |
| --- |
| private:  **//Keeps track of the total number of entries in the table.**  **//Used to ensure we do not goover the limit. -1 means**  **//that we don’t yet know how many entries the table has.** |

示例 5.8

(3)对于有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义。变量、常量、宏的注释应放在其上方相邻位置或右方，示例如下：

|  |
| --- |
| **// active statistic task number**  #define MAX\_ACT\_TASK\_NUMBER 1000 |
| #define MAX\_ACT\_TASK\_NUMBER 1000 **// active statistic task number** |

示例 5.9

(4)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，示例如下：

|  |
| --- |
| **/\*!**  **\*@brief**  **\*The ErrorCode when SCCP translate**  **\*Global Title failure, as follows**  **\*@range**  **\*0 - SUCCESS 1 - GT Table error**  **\*2 - GT error Others - no use**  **\*/**  BYTE g\_GTTranErrorCode; |

示例 5.10

# 6、附录

## 6.1类型前缀列表

|  |  |  |
| --- | --- | --- |
| **前缀** | **变量类型** | **备注** |
| m\_ | 成员变量 | 作用域前缀 |
| g\_ | 全局变量 | 作用域前缀 |
| s\_ | 模块内静态变量 | 作用域前缀 |
| 无 | 局部变量 | 作用域前缀 |
| u | unsigned short  unsigned int  unsigned long | 无符号整数 |
| n | short  int | 带符号整数 |
| f | float |  |
| d | double |  |
| c | char | 字符 |
| uc  by | unsigned char  BYTE |  |
| l | long |  |
| ll | long long |  |
| p | 指针 | 一般p后面会跟随一个指示具体类型的前缀，  例如:  “pu”表示一个无符号数指针，  “po”表示一个对象指针 |
| pp | 指针的指针 | 其它更多指针依此类推 |
| sz | char \*  unsigned char \* | 程序中明确存放的是以0结尾的字符串 |
| bf | char \*  unsigned char \* | 程序中明确存放的是二进制数据 |
| b | bool  BOOL |  |
| h | HANDLE |  |
| w | unsigned short  WORD |  |
| dw | unsigned long  DWORD |  |
| a | 数组 |  |
| s  str | std::string  CString  QString  LPCSTR  LPCTSTR | 各种类库的字符串 |
|  |  |  |
| I | 纯虚类 | 面向对象中的接口 |
|  |  |  |
| U | union |  |

表格 6‑1

## 6.2 C++关键字表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| asm | auto | bad\_cast | bad\_typeid | bool | break | case | catch |
| char | class | const | const\_cast | continue | default | delete | do |
| double | dynamic\_cast | else | enum | except | explicit | extern | false |
| finally | float | for | friend | goto | if | inline | int |
| long | mutable | namespace | new | operator | private | protected | public |
| register | reinterpret\_cast | return | short | signed | sizeof | static | static\_cast |
| struct | switch | template | this | throw | true | try | type\_info |
| typedef | typeid | typename | union | unsigned | using | virtual | void |
| volatile | wchar\_t | while | export |  |  |  |  |

表格 6‑2