**C++编码规范**

此编码规范主要针对代码风格、头文件、命名、变量结构、函数过程等内容进行了定义和说明，此规范已假设读者熟悉C++。

内容简表：

|  |  |
| --- | --- |
| [代码风格](#代码风格) | [缩进](#缩进) [变量说明](#变量说明) [长语句](#长语句) [一行只写一条语句](#一行只写一条语句) [UTF-8 编码](#UTF8) |
| [头文件规范要求](#头文件规范要求) | [#define 保护](#define保护) [头文件依赖](#头文件依赖) [Inline 函数](#inline函数) [函数参数排序](#函数参数排序) [命名和排序](#命名和排序) |
| [标识符命名规则](#标识符命名规则) | [通常命名](#通常命名) [文件名](#文件名) [目录名](#目录名) [类型名](#类型名) [变量名](#变量名) [常量名](#常量名) [函数名](#函数名) [命名空间命名](#命名空间) [枚举名](#枚举名) [宏定义名](#宏定义名) [规则例外](#命名规则例外) |
| [注释](#注释) | [注释风格](#注释风格) [文件注释](#文件注释) [类注释](#类注释) [函数注释](#函数注释) [变量注释](#变量注释) [实现的注释](#实现的注释) [标点和语法](#标点和语法) [TODO注释](#todo注释) |
| [变量、结构](#变量结构) | [原则](#原则) [命名空间](#命名空间（变量和结构）) [嵌套classes](#嵌套classes) [静态和全局变量](#静态和全局变量) |
| [函数、过程](#函数和过程) |  |
| [类](#类) |  |
| [质量保证](#质量保证) |  |
| [其他c++建议](#其他c建议) | [聪明指针](#聪明指针) [reference参数](#reference) [缺省参数](#缺省参数) [变长数组和alloca()](#alloca)  [friends](#friends) [RTTI](#rtti) [类型转换](#类型转换) [streams](#streams) [Preincrement and Predecrement](#pandp) [const](#const) [整数类型](#整数类型) [64-位 移植性](#移植性) [预处理宏](#预处理宏) [0 和 NULL](#null) [Sizeof](#sizeof) |

1. **代码风格：**
   1. 程序块要采用缩进风格编写，缩进的空格数为4个。
      1. 预处理 directives 不要缩格，总要从行开头开始
      2. namespace 下内容不做进一步缩格, 在namespace结束时，注释一下 } // namespace vehicle
   2. 缩进或者对齐使用空格，或者将Tab字符转换为空格，不使用Tab。
   3. 相对独立的程序块之间、变量说明之后必须加空行。
   4. 较长的语句（>100字符）要分成多行书写。
   5. 不允许把多个短语句写在一行中，即一行只写一条语句。
   6. 避免使用非-ASCII 字符，如果使用，必须用 UTF-8 编码。
   7. if ... else...格式规范： if后面的大括弧可以跟if一行，也可以另起一行(但要统一)。格式如下：

if (xxxx) {

xxxx;

} else {

}

或者：

if (xxxx)

{

xxxxx;

}

else

{

}

1. **头文件规范要求：**

每个.cpp 文件通常有个 .h 文件. 通常的例外包括 unittests 和包含main()函数的小的 .cc 文件。

* 1. #define 保护：每个头文件都应有 #define 保护来避免多次包含。它的格式应为： <PATH>\_<FILE>\_H\_.
  2. 头文件依赖：如果 forward declaration 足够，不要用 #include。
  3. Inline 函数：只能把很小的函数，比如，短于10行的，定义为 ”inline“。
  4. 函数参数排序：函数参数的顺序是: 先输入, 后输出。
  5. 头文件包含的命名和顺序：使用标准顺序来增加可读性及避免隐含依赖: C library, C++ library, 其他库的 .h 文件, 你的项目的 .h文件.

1. **标识符命名规则：**
   1. 通常命名:规则函数名、变量名和文件名应当有意义，避免缩写。类型和变量应当是名词。
   2. 文件名:文件名应当全部都是 lowercase。可包含 underscore (\_) 或者 dashes (-)，参照你项目的约定。 如果项目没有约定，建议用 "\_"。
   3. 目录命名规则： 每个目录都用小写字母。因为linux对大小写敏感，都用小写，整齐划一。例如：src, conf。
   4. 类型名名以大写字母开始，每个新词都用大写字母开始，不用 "\_"： MyExcitingClass, MyExcitingEnum。类型包括 class, struct, enum, typedef 等。
   5. 变量名:变量名全用小写字母。词之间用下划线分开。普通类成员变量名加 m\_ 前缀。Static类成员变量加 s\_ 前缀。全局变量加 g\_ 前缀。
   6. 常量名：用首字母 k 加混合大小写: kDaysInAWeek.
   7. 函数名:函数用混合大小写，采用动宾结构。例如 GetProperty(), SetProperty(), SerializeIntoString()。
   8. 命名空间的命名 (namespace)的名字全用小写字母和下划线，基于项目名，亦可加上目录结构: websearch\_project.
   9. 枚举名：枚举变量应该象 常量 或者 宏定义那样: 或者 kEnumName， 或者 ENUM\_NAME.
   10. 宏定义名：不建议使用宏 。 如果你一定要用，它应该象: MY\_MACRO\_THAT\_SCARES\_SMALL\_CHILDRE
   11. 命名规则例外：如果你需要修改已存在大文件，最好遵循那个文件的命名规则。
2. **注释：**
   1. 注释很难写，但对你代码的可读性及其有用。请遵循下面的注释规则。 但请记住：虽然注释很有用，但最好的代码是那种可以自我注释的。 给变量起一个有意义的名字远比起一个看不懂的名字再加注释更重要。
   2. 注释的风格：可使用 // 或者 /\* \*/。但要保持一致性。
   3. 文件注释：每个文件开头要有版权通知和作者姓名。加下来是对文件内容的概要描述。
   4. Class 注释：每个class定义都应有注释解释它干什么和怎么用。
   5. 函数注释：函数宣布的注释描述函数的使用；函数定义的注释描述它的操作和实现。
   6. 变量注释：通常一个变量的名字应当能够解释它的作用。某些复杂名字应该加注释。
   7. 实现的注释：在你的实现代码里，对那些不容易看懂的、不直观的、有趣的和重要的部分 要加注释。
   8. 标点和语法：写注释时，注意标点和语法。写得好的、文法通顺的注释才容易读。
   9. TODO 注释：对那些临时的、短期的、可用但不是最有的代码，用 TODO 注释 来标志未来需要干的事。
3. **变量、结构**
   1. 尽量少使用全局变量，尽量去掉没必要的公共变量。
   2. 变量，特别是指针变量，被创建之后应当及时把它们初始化，以防止把未被初始化的变量当成右值使用。
   3. 防止局部变量与公共变量同名局部变量。
   4. 函数的变量应放在最窄的空间（scope）。且于宣布时初始化。
   5. 尽量减少没有必要的数据类型默认转换与强制转换。
   6. 当声明用于分布式环境或不同CPU间通信环境的数据结构时，必须考虑机器的字节顺序、使用的位域及字节对齐等问题。
   7. 命名空间：鼓励在.cc 文件中使用无名的 命名空间(namespace). 对于有名字的 namespace, 所取名字应当给予项目名或者文件路径名。 禁止使用 using-directive.
   8. 嵌套 Classes：避免使用 ”public“ 嵌套 classes。 考虑使用 namespace 来避免全局类定义。
   9. 静态和全局变量：禁止使用 ”class“ 类型的静态和全局变量。由于构建和破坏顺序的不确定性， 它们经常导致极难找到的bug
4. **函数、过程**
   1. 调用函数要检查所有可能的返回情况, 不应该的返回情况要用ASSERT来确认。
   2. 接口函数参数的合法性检查应由接口函数本身负责。ASSERT为运行时不出现的情况，if判断运行时可能出现的情况。
   3. 函数的规模尽量限制在200行以内。
   4. 一个函数仅完成一件功能。
   5. 尽量写类的构造、拷贝构造、析构和赋值函数 ，而不使用系统缺省的。析构函数要free应该free的资源。
   6. 检查函数所有参数与非参数的有效性。
   7. 函数实现中不改变内容的参数要定义成const。示例： int GetStrLen(const TCHAR\*);
   8. 函数的返回值要清楚、明了，让使用者不容易忽视错误情况。
   9. 减少函数本身或函数间的递归调用。
   10. 避免设计多参数函数，不使用的参数从接口中去掉。
   11. Constructors 里的内容：通常， constructors 只用来初始化成员变量。任何复杂的初始化应放于 专门的Init() 函数.
   12. 缺省 Constructors：如果你的class有成员变量而没有其他constructors，你必须定义缺省 constructor。否则compiler就会给你生成一个，极可能是你不想要的。
   13. Explicit Constructors：如果constructor只有一个参数，用 C++ 关键词 explicit。
   14. Copy Constructors：只有在必要时才提供 copy constructor and 赋值运算。 否则，把它们宣布在 ”private“ 里。
   15. 结构与类：只对简单被动数据结构用结构(struct)； 其他都用类 (class).
   16. Inheritance：Composition 通常比 inheritance 更合适. 如果用 inheritance, 把它定义为public.
   17. 多重Inheritance：尽可能避免多重Inheritance。如果用多重Inheritance，则只允许 其中的一个不是抽象class。
   18. 运算符重载：除非在非常特殊情况下，禁止重载运算符。
5. **质量保证：**
   1. 防止引用已经释放的内存空间。
   2. 过程/函数中动态分配的资源（包括内存、文件等），在过程/函数退出之前要释。
   3. 防止内存操作越界。
   4. 不允许嵌套使用goto语句。
   5. 打开编译器的所有告警开关对程序进行编译，并且要确认、处理所有的编译告警。
6. **其他C++的建议**
   1. 聪明指针：如果用指针(pointer)， scoped\_ptr 很好用. 避免使用 std::tr1::shared\_ptr， 除非在特殊情况下, 比如指针对象存于STL容器中时, 禁止使用auto\_ptr.
   2. Reference 参数：所有的”reference“ 参数必须是 ”const”。
   3. Friends：允许使用 friend classes 和函数，但在一定范围内。
   4. 类型转换：使用 C++ 式的类型转换，如 static\_cast<>(). 不要使用其它格式，比如： int y = (int)x; or int y = int(x);
   5. Preincrement and Predecrement：对于STL iterators 和其他 template 对象，使用 prefix(++i) 式的增减运算符。
   6. 使用const：强烈推荐在尽可能的情况下使用 const。
   7. 整数类型：C++ 有几种整数类型（short, int, long）。只准使用int。 如果一个程序需要一个不同的类型，则从<stdint.h> 挑选一个定长类型, 比如 int16\_t。
   8. 64-位 移植性：代码应为 64-bit 和 32-bit 友好. 注意打印、比较和 structure alignment 可能出现的问题.
   9. 预处理宏：小心宏的使用。尽可能用 inline functions, enums, 和 const 变量来取代宏定义。
   10. 0 和 NULL：整数用 0, 实数用0.0, 指针用NULL，字符用'\0'。
   11. Sizeof：尽可能用sizeof(type) 而非sizeof(varname)。