



C++ 编码规范



说明：

- 本编码规范仅在新的 C++ 工程中使用。在一个已有的工程中编写代码时，以与该工程现有编码风格保持一致为原则。
- 本编码规范中的部分条款代表着更好的编码习惯，而另一些仅仅是为了统一风格而做的人为规定。
- 本编码规范中的所有条款均可以在特殊情况下违背，但必须加上必要的注释说明理由。

1. 命名

1.1 文件命名

- 文件名全部使用小写字母。



示例：`kctlinechartview.h`

1.2 类命名

- 类的命名 **使用前缀 + 驼峰命名法**。
- 当一个类需要暴露给其它工程使用时，应加上工程约定的前缀，前缀全部使用大写字母。
- 类命名尽量使用完整英文单词，除非是常见的缩写



示例：

C/C++

```
1 KCTLineChartView
```

- Com 接口命名在普通前缀前加 I 作为标志。



示例：IKCTLineChartView

1.3 其他类型命名

- **struct**、**typedef** 与 **enum** 的命名原则与类相同。
- enum 成员 **采用大写字母开头的驼峰写法**，**且应在命名中体现 enum 名称**，具体位置不做强制要求，但同一个 enum 内的命名规则要保持一致。



示例 枚举类型

C/C++

```
1 enum ActionType
2 {
3     InsertActionType,
4     DeleteActionType,
5     MoveActionType
6 };
```

1.4 变量命名

- 变量命名**使用小写字母开头的驼峰写法**，不加类型信息前缀，命名应使用能表达变量涵义的完整英文单词。



示例：

```
int index = 0;

KCTLineChartView *lineChartView = NULL;
```

- 仅允许在 for 单层循环中的 i 和表示坐标的 x、y、z 上使用单字母变量**，其他情况下不允许使用。
- 非 static 的类成员变量** 前加上小写字母 **m_前缀**，**static 的类成员变量** 前加上 **小写字母 s_前缀**，这两种情况下变量名的 **第一个单词首字母小写**，后面单词首字母大写。



示例

C/C++

```
1 class GoodClass
2 {
3 private:
4     int m_dataMember;
5     static int s_staticDataMember;
6 };
```

1.5 函数命名

- 除构造函数和析构函数外，**类成员函数使用小写字母开头的驼峰写法**。
- 非类成员函数使用大写字母开头的驼峰写法。



示例：

```
lineChartView->lineCount();  
GetObjectCountInDocument();
```

- 名词属性的类成员函数命名分两种情况：
 - 若欲获取的信息当成返回值返回，则命名方式为名词或名词 + 修饰语 的样式。
 - 若欲获取的信息通过某个参数返回，则命名方式为 get+ 名词（+ 修饰语）的样式。



示例：

```
lineChartView->title();  
lineChartView->subViewAtIndex(index);  
lineChartView->getSubViews(&outPointer);
```

- 动词属性的类成员函数命名方式为动词 + 宾语，若宾语是 this, 应省略。



示例：

```
lineChartView->removeAllSubViews();  
lineChartView->setTitle(L" Hello World" );  
lineChartView->initialize();
```

- 形容词属性的类成员函数命名方式为 is/has 等修饰语 + 形容词 / 名词。



示例：

```
lineChartView->isVisible();
```

```
lineChartView->hasTitle();
```

- 返回对象指针的函数若内部分配了内存，应使用 copy, create 等关键字把该信息反映在函数名上。



示例：

```
lineChartView->subViews();    // 未重新分配内存
```

```
lineChartView->copySubViews(); // 重新分配了内存
```

1.6 宏命名

- 原则上**宏的命名全部大写，如有必要单词间用下划线分隔**。如无必要，不把一段代码定义成宏。



示例：

```
#define SOMETHING_USED
```

1.7 全局变量

- 关于使用全局变量的注意事项见后文。
 - **全局变量前加小写字母 g_ 前缀**
 - **static 全局变量前加小写字母 gs_前缀。**



示例：

```
bool g_inFileOpening = false;
```

```
static bool gs_staticInFileOpening = true;
```

1.8 名称空间命名

- 名称空间使用剪短的全小写英文单词命名。



示例：

C/C++

```
1 namespace chart
2 {
3     ....
4 }
```

1.9 不允许使用 **My** 或自己的姓名作为以上命名的前缀

- 禁止下列写法**

C/C++

```
1
2 Class MyStringProject
3 {
4     public:
5     ...
6 };
```

2. 头文件

2.1 头文件包含保护

- 在所有头文件中均使用 **#ifndef + #define + #endif** 来避免该文件被重复包含
- 宏的命名方式为 **__PROJECTNAME_FILENAME_H__**。





示例：

- KProject 项目有头文件名为 KHeader.h，则文件头部加上

```
#ifndef __KPROJECT_KHEADER_H__
```

```
#define __KPROJECT_KHEADER_H__
```

- 末尾加上

```
#endif //__KPROJECT_KHEADER_H__
```

2.2 前置声明

- 当使用一两个前置声明就能编译通过时，不要在头文件里包含另一个头文件。

2.3 头文件包含顺序

- 在代码文件里包含多个头文件时，应按照如下顺序将头文件分组，每组之间使用空行隔开：
 - a. 预编译头文件，通常是 stdafx.h
 - b. 与代码文件同名的头文件
 - c. 系统头文件
 - d. WPS 内部其他工程头文件
 - e. 本工程其他头文件

示例：

- KProject 工程中 用到如下头文件
 - kclass.cpp 实现了声明在 kclass.h 内的类，
 - C++ 标准库中的 Vector 容器、
 - ExternalProject 工程中的 externalclass.h 和 externalutils.h
 - KProject 工程中的 kfriendclass.h、kfilelohelper.h 与 kstringutils.h，其包含顺序如下：

C/C++

```
1 #include "stdafx.h"
2 #include "KClass.h"
3
4 #include <vector>
5
6 #include "externalclass.h"
7 #include "externalutils.h"
8
9 #include "kfirendclass.h"
10 #include "kfileiohelper.h"
11 #include "kstringutils.h"
```

3. 类

3.1 初始化

- **类的所有成员变量必须初始化，只有成员变量没有函数的类，必须定义默认构造函数。**
- 除非有特殊需求，在类的构造函数中仅进行不涉及具体功能的初始化操作，例如为成员变量赋零。
- 较复杂的初始化操作，应放在一个单独的 `init()` 方法中，**由类实例的创建者负责调用。**



解释：

- **构造函数内难以报告错误**，且构造函数中对虚函数的调用不会派发给子类，所以较复杂的初始化操作应在构造完成之后进行

3.2 初始化

- 若没有明确的将单参构造函数用于隐式类型转换的需求，应使用 **`explicit`** 关键字。

3.3 拷贝构造函数、赋值运算符

- **若没有明确的对类进行拷贝的需求，应在 `private` 段中声明拷贝构造函数和赋值运算符。**

3.4 继承

- **只使用 `public` 继承。**

- 在语义明确的时候从父类继承具体实现，其他时候从抽象接口继承。
- 若类有虚函数，则析构函数也定义为虚函数。
- 父类中声明为 **virtual** 的函数，子类声明中要明确标明为 **virtual** 以及 **override**。

3.5 继承

- 尽量避免运算符重载，除非是为了在容器类中使用而必须实现的。

3.6 访问权限声明

- 类声明中按 **public, protected, private** 的顺序声明函数和变量，
- 每个关键字仅占用一段，每一段中的声明顺序为：
 - typedef, enum,
 - Q_OBJECT 及类似声明
 - 嵌套类，
 - 常量，
 - 构造函数， 虚构造函数， 成员函数， 数据成员。

3.7 友元

- 友元的定义和友类的定义应放在同一个文件中。

4. 函数与实现

4.1 参数

- 参数排列顺序为输入参数在前，输出参数在后。
- 引用作为输入参数时应配合 **const** 使用。不使用缺省参数。
- 不允许一个参数即作为传入参数，又作为传出参数。

4.2 内联函数

- 内联函数不超过 10 行，推荐仅对 1 行的函数进行 inline。
- 不使用循环或 switch...case，不进行递归。

- 在类定义体内实现的函数，不要加 `inline` 关键字。

4.3 异常处理

- **避免使用异常（`try...catch`）。**使用 ATL 和 STL 时，关闭异常或限制异常作用域。

4.4 类型编程

- 不使用运行时类型信息。
- 明确使用 `static_cast`, `const_cast` 进行类型转换，避免使用 `dynamic_cast`, 小心使用 `reinterpret_cast`。

4.5 宏

- 尽量使用 `const`、`enum`、`inline` 替代 `#define`。

4.6 类数据成员的引用

- **避免成员函数返回指向类成员的指针或引用。**

5. 注释

5.1 文件头

文件头注释应包括文件名，创建者，创建时间，功能描述和版权信息。



C/C++

```
1 // -----  
2 // kctcodingguideline.h  
3 // 创建者： Tom Cat  
4 // 创建时间： 2013/5/24  
5 // 功能描述： The coding guideline for Chart project in WPS  
6 // Copyright 2013 Kingsoft  
7 // -----
```

5.2 代码注释

- 注释使用 **//** 风格，应简洁清晰，不写没必要的注释。代码中不是很简单直观的地方需要增加注释，



例如：

- 故意违背编码规范之处。
- 分成多步完成的任务。
- 较复杂的逻辑。
- 和常理不符的代码。
- 比较重要，需要引起注意的地方。

5.3 注释语言

- 默认注释语言为 **中文**。
 - a. 注释中的工程师姓名
- 在且仅在两种情形下把姓名加入注释中：
 - a. 文件头中的创建者信息。
 - b. 做 TODO 注释时。



示例：

//TODO(Tommy Zhang): 删掉这里的全局变量！

5.4 预处理宏注释

- 中间代码段较长的`#else` 与`#endif` 之后用注释标明宏的名字。



示例：

C/C++

```
1 #if TARGET_PLATFORM_WINDOWS
2 .....
3 #else //TARGET_PLATFORM_WINDOWS
4 .....
5 #endif //TARGET_PLATFORM_WINDOWS
```

6. 格式

6.1 对齐与缩进

- 使用 Tab 进行对齐。
- **花括号上下对齐，不允许将左花括号放在 if 等语句的末尾：**



示例如下：

C/C++

```
1 void KFunction(InputType type)
2 {
3     static const int someInt[] = {80, 40, 40, 50, ... 94, 94, 47
4     };
5     static const ScopeType scopeType[] =
6     {
7         wpsFindScope_MainText,
8         wpsFindScope_Selection,
9         wpsFindScope_HeaderFooters,
10        wpsFindScope_Footnotes,
11        wpsFindScope_Endnotes,
12        ...
13        wpsFindScope_Comments
14    };
15    switch (type)
16    {
17        case TypeA:
18            break;
19        case TypeB:
20            break;
21        default:
22            break;
23    }
24    if (...)
25    {
26    }
27    else
28    {
29    }
30
31    do
32    {
33    }
34    while (...);
35
36    for (int i = 0; i < 10; ++ i)
37    {
38
```

```
    }  
39 }
```

- 构造函数初始化列表和多个基类中每一行逗号放前面，和冒号对齐。

 示例：

C/C++

```
1 KGroupGrid::KGroupGrid(QWidget *parent)  
2         : QWidget(parent)  
3         , m_scrollBar(Qt::Vertical, this)  
4         , m_headerHeight()  
5         , m_itemSize(80, 64)  
6         , m_showTooltip(false)  
7         , m_flatFrame(true)  
8         , m_showSeperatorLine(true)  
9         , m_scrollBarPolicy(Qt::ScrollBarAsNeeded)  
10        #if X_OS_WINDOWS  
11            , m_firstRow(0)  
12        #endif  
13        {  
14        }  
15        class KxWppViewPages  
16            : public QStackedWidget  
17            , public KFakeUnknown<IShellPagesGetter>  
18            , public KFakeUnknown<IROShellPages>  
19            , public KxWppViewPagesCoreNotify  
20        {  
21        };
```

- 只有一行代码的if语句不加花括号。

 示例：

C/C++

```
1      if (inputType == badType)
2          DoSomethingBad();
3
4      if (inputType == badType)
5          DoSomethingBad();
6      else
7          DoSomethingElse();
```

- 当 if 语句中某一个分支有一行的以上代码时，if 的所有子句都要加括号。



示例：

C/C++

```
1  if (inputType == badType)
2  {
3      DoSomethingBad();
4  }
5  else
6  {
7      DoSomethingElse1();
8      DoSomethingElse2();
9  }
```

- 推荐一行代码不超过 120 个字符。
- 函数参数列表过长时，应换行：**返回值类型、左括号应保持与函数名同一行，具体换行的方案有两种：**



示例：

C/C++

```
1 HRESULT KClass::method(ParameterOne param1,  
2                          ParameterTwo param2,  
3                          ParameterThree param3)  
4  
5 HRESULT KClass::method(ParameterOne param1,  
6                          ParameterTwo param2, ParameterThree param3)
```

- 函数类型为 const 时，const 关键字与最后一个参数同行



示例：

C/C++

```
1 HRESULT KClass::constMethod(ParameterOne param1,  
2                             ParameterTwo param2,  
3                             ParameterThree param3) const
```

- 所有预编译宏顶格对齐。



示例：

C/C++

```
1      void KClass::methodWithMacros()
2      {
3          static int integer1 = 0;
4          integer1 ++;
5          if(integer1 >= 10)
6          {
7              #if PLATFORM_1
8                  integer1 = 1;
9              #else
10                 integer1 = 2;
11             #endif
12         }
13     }
```

- 分为多行的布尔表达式中 && 与 || 置于行首。
- **换行应在运算符优先级最低的地方进行，尽可能避免把配对的括号分成两行。**
- 当条件比较复杂时，应将高优先级的运算用括号明确标识出来。



示例：

C/C++

```
1  if ((valueGood >= somethingGood || otherContion)
2      && (valueBad + 1) <= somethingRealyBad)
```

- **命名空间 (namespace)内容不缩进**

6.2 空格

- **函数调用的括号前后之间不加空格，参数之间逗号之后加一个空格。**



示例：

C/C++

```
1 void KFunction(ParameterOne param1, ParameterTwo param2)
```

- if、while、for 等关键字与括号之间加一个空格，括号后不加空格。

 示例：

C/C++

```
1 if (NULL == thePointer)
2
3 for (int index = 0; index < maxIntValue; index++)
```

- 使用 Tab 键无法刚好对齐时，使用空格补齐。
- 没有参数的函数括号内不加空格。
- **二元运算符前后各加一个空格；**
- 自增减运算符与分号间不加空格，与变量之间也不加空格。

 示例：

C/C++

```
1 if (inputType1 == goodType && inputType2 == badType)
2
3 for (int index = 0; index < MAX_INT; index++)
```


- 引用符号（. ->）前后不加空格。

 示例：

C/C++

```
1 theSmartPointer->value();
```

- 地址、引用运算符（*, &）后不加空格；

 示例：

C/C++

```
1 *integerPointer = 1;
```

- 声明指针类型的 * 号前加一空格，后边不加空格

 示例：

C/C++

```
1 KClass *goodClass;
```

7. 作用域

7.1 嵌套类

- 不需要暴露嵌套类作为接口的时候，将嵌套类声明为 `private`。

7.2 局部变量

- 局部变量应在将要使用时进行声明，声明的同时初始化。



示例：

C/C++

```
1      void SomeMagicalFunction()
2      {
3          const int magicNumber = 1;
4          int integer1 = magicNumber;
5          integer1 ++;
6          if (integer1 <= 0)
7              DoSomethingWeird();
8
9          float floatingNumber1 = 1.0;
10         float *floatingPointer = &floatNumber1;
11         if (floatingPointer == NULL)
12             HowCouldThisHappen();
13     }
```

7.3 全局变量

- 除非特殊情况，不使用 `Class` 类型的全局变量，不使用 `Class` 类型的 `static` 类数据成员。
- 可以用单例模式替代 `Class` 类型的全局变量。
- 全局的字符串变量使用 C 风格的 `char`，不使用各种字符串类。

7.4 名称空间 (namespace)

- 允许在 cpp 文件中使用匿名名称空间进行保护，**不允许在头文件中使用匿名名称空间**。
- **避免使用 using namespace 将一个名称空间中的所有名称全部导入。**

8. 模板

8.1 模板声明

- 使用 typename 关键字声明模板，不使用 class 关键字。

8.2 模板适用范围

- **一般情况下仅使用模板来实现容器类或通用算法，如果需要用作其他用途，请增加注释说明必要性。**

8.3 模板中的嵌套类

- 模板中使用嵌套类时，加上 typename 关键字声明。



示例：

C/C++

```
1 template<typename T>
2 void someFunction(const T& container)
3 {
4     typename T::Iterator iter(container.begin());
5     iter.next();
6 }
```

8.4 模板继承

- 在模板派生类中调用模板基类函数时，明确在函数调用前声明该函数属于基类。



示例：

C/C++

```
1 template<typename T>
2 class DerivedClass::public BaseClass<T>
3 {
4 public:
5     void doAction()
6     {
7         BaseClass<T>::doBaseAction();
8     }
9 }
```

9. 内存

9.1 优先使用静态内存

- 编译期能够确定的常量数组，应声明为 static。



示例：

C/C++

```
1 void func(int type)
2 {
3     static const int someInt[] = {80, 40, 40, 50, ... 94, 94, 47};
4     .....
5 }
```

9.2 其次为栈内存

- 编译期能够确定长度的非常量数组，应使用栈内存。



示例：

C/C++

```
1 void func(int type)
2 {
3     const int length = 1024;
4     unsigned char buffer[length + 1];
5     DWORD dwReadSize = length;
6
7     while (::ReadFile(hFile, buffer, length, &dwReadSize, NULL))
8     {
9         .....
10        dwReadSize = length;
11    }
12 }
```

9.3 使用智能指针和容器管理堆内存

- 单个堆对象的生命周期用 `std::unique_ptr` 来管理。
- 连续的内存空间用 `std::vector` 或 `QVector` 来分配和管理，**不允许在代码中出现 `new[]` 和 `delete[]` 的显示调用。**



示例：

C/C++

```
1 void func(int type)
2 {
3     std::unique_ptr<CSomeObject> spObject = new CSomeObject();
4
5     int bufferLength = getBufferLength();
6     std::vector<SomeType> vecBuffer(bufferLength);
7
8     readBuffer(&vecBuffer[0], bufferLength);
9 }
```

- 字符串用 `std::basic_string<T>` 或 `QString` 来管理。



示例：

C/C++

```
1 void func(int type)
2 {
3     int stringLength = getStringLength();
4     std::string str;
5     str.resize(stringLength);
6     readStringInCStyleFunction(&str[0], stringLength);
7 }
```

- `delete` 不应该出现在析构函数或用于清理内存的函数之外的地方出现。
- 不允许使用 `malloc/free` 来分配和释放内存。

10. 跨平台

10.1. Windows 内建类型

- 调用 Windows API 时，使用 **DWORD**、**LPCTSTR** 等 Windows 类型，其它情况下避免使用，但可以使用 **HRESULT**。

10.2. Windows API

- 避免使用 Windows 系统 API，只能在限定模块内使用，比如一个封装操作系统 API 的中间层。

10.3. C++ 扩展库

- 不允许使用 Boost 库，除非是已经进入 C++11 标准的部分。

10.4. C++ 语法

- 可以使用 C++11 新语法中 Visual C++ 支持的部分，但须遵守如下原则：
 - 不使用 lambda 表达式。
 - 仅使用 auto 来简化模板变量的声明，不将基本类型或者表达式返回值赋给 auto 对象。
 - 不使用尾部返回类型声明，例如 auto Function() -> int;
 - 不使用 Raw String。
 - 禁用模版元编程。
 - 不允许使用逗号表达式。
- **不能使用 Visual C++ 专有语法**，例如：
 - **__finally**、**__super**、**__forceinline** 等关键字。
 - 省略类静态成员的外部定义。
 - 将函数指针 cast 成整型指针。
 - 重复包含有 extern 全局变量定义(非声明)的头文件。
 - goto 关键字跨越变量定义。
- **如果有可能，在 Visual Studio 中设置 /Za 标志关闭 Microsoft Extension to C++。**