# 头文件

## 1.1#include保护

为了保证代码只编译一次，可采用以下两种方式：

第一种：

#pragma once

第二种：

#ifndef \_\_FILE\_\_

#define \_\_FILE\_\_

……

#endif

## 1.2头文件的依赖

遵循一个原则：.h文件应该尽量的少，而要包含的头文件尽量放在.cpp文件中。有些情况是可以在.h文件中声明使用，然后再.cpp文件中包含该有的头文件。如在头文件如何做到使用类Foo而无需访问类的定义：

1) 将数据成员类型声明为Foo \*或Foo &；

2) 参数、返回值类型为Foo的函数只是声明（但不定义实现）；

3) 静态数据成员的类型可以被声明为Foo，因为静态数据成员的定义在类定义之外。 不计入类对象的大小。

另一方面，如果你的类是Foo的子类，或者含有类型为Foo的非静态数据成员，则必须为之包含头文件。当然，.cpp文件无论如何都需要所使用类的定义部分，自然也就会包含若干头文件。

## 1.3 #include的路径和顺序

路径符一律使用“/”，而使用“\\”。

头文件包含顺序：C库、C++库、其他库的头文件、自己工程的头文件。当然在.cpp文件中如test.cpp，最先包含的是test.h。

# 2.作用域

## 2.1名字空间

尽量不用using来简化代码，提高代码的可读性和防止二义性。

尽量不要使用全局函数。

应该使用命名空间中的非成员函数和类的静态成员函数。这样做的原因是在某些情况下，非成员函数和静态成员函数是非常有用的，将非成员函数置于命名空间中可避免对全尿作用域的污染。

有时，不把函数限定在类的实体中是有益的，甚至需要这么做，要么作为静态成员，要么作为非成员函数。非成员函数不应依赖于外部变量，并尽量置于某个名字空间中。相比单纯为了封装若干不共享任何静态数据的静态成员函数而创建类，不如使用名字空间。

## 2.2 局部变量

（1）将局部变量尽可能置于最小作用域内，在定义时将其显示初始化。

（2）构造数据类型的变量尽可能放在循环体外定义。

## 2.3 全局变量

（1）尽量不要定义构造类型的全局变量

构造类型的全局变量，如类对象的构造函数、析构函数以及初始化操作的调用顺序只是被部分规定，每次生成有可能会有发化，从而导致难以发现的bugs。

因此，应禁止使用class类型的全局变量（包括STL的string, vector等等），因为它们的初始化顺序有可能导致构造出现问题。内建类型和由内建类型构成的没有构造函数的结构体可以使用，如果一定要使用class类型的全局变量，请使用单件模式（singleton pattern）。

（2）对于全局的字符串常量，使用C风格的字符串，而不要使用STL的字符串。

虽然允许在全局作用域中使用全局发量，使用时务必三思。大多数全局变量应该是类的静态数据成员，或者当其只在.cpp文件中使用时，将其定义到不具名名字空间中，或者使用静态关联以限制变量的作用域。

记住，静态成员变量视为作用域限制在类域的全局变量，所以，也不能是class类型！

# 3.类

## 3.1 构造函数

（1）构造函数的初始化列表，应和类的顺序一致。  
（2）初始化列表中的每个项，应独占一行。  
（3）避免出现用一个成员初始化另一个成员。（或者定义一个默认构造函数）  
（4）构造函数应初始化所有成员，尤其是指针。  
（5）不要在构造函数和析构函数中抛出异常。

## 3.2 虚函数

接口类的虚函数应该设置为纯虚函数。

## 3.3 构造和析构函数

（1）如果类可以继承，则应将类析构函数设计为虚函数。  
（2）如果类不允许继承，则应将类析构函数设计为非虚函数。  
（3） 如果类不能被复制，则应将拷贝构造函数和赋值运算符设计为私有的。  
（4）如果为类设计了构造函数，则应有析构函数。

## 3.4成员变量

（1）尽量避免使用mutable和Volatile。

（2）尽量避免使用公有成员变量。建议多使用私有成员并提供存取控制。

## 3.5成员函数

（1）努力使类的接口少而完备。  
（2）尽量使用常成员函数代替非常成员函数，const函数。  
（3）除非特别理由，绝不要重新定义(继承来的)非虚函数。（这样是覆盖，基类的某些属性无初始化）

## 3.6 友元

没有特殊必要的情况下，尽量不要使用友元函数和友元类。

## 3.7 声明顺序

（1）按照public、protected、private的顺序声明。

（2）先成员变量，再成员函数

（3）成员变量顺序：static类数据、其他成员变量。

# 4 其他C++特性

（1）函数参数顺序先输入，后输出。如果需要输出复杂的类或者数据类型，应该在参数中输出，而不是在return输出。

（2）函数参数中不鼓励使用默认缺省参数。

（3）类型转换： dynamic\_cast、const\_cast、static\_cast、reinterpret\_cast。根据情况使用，简单类型使用C语言风格转换。

（4）前置自增和自减（迭代时建议使用，因为前置比后置效率高）

（5）不建议使用预处理宏，可用const代替。

（6）数据类型用0，地址和string类型使用NULL。

（7）尽量使用sizeof（变量）而不是sizeof（类型）。

# 5 命名规则

**类/结构**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 除了异常类等个别情况（不希望用户把该类看作一个普通的、正常的类之情况）外，C++类/结构  的命名应该遵循以下准则：   |  |  | | --- | --- | | C++类/结构的命名 | 类的名称都要以大写字母“C”开头，后跟一个或多个单词。为便于界定，每个单词的首字母要大写。 | | 推荐的组成形式 | 类的命名推荐用"名词"或"形容词＋名词"的形式，例如："CAnalyzer", "CFastVector" .... | |

**函数**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 函数的命名 | 函数的名称由一个或多个单词组成。为便于界定，每个单词的首字母要大写。 | | 推荐的组成形式 | 函数名应当使用"动词"或者"动词＋名词"（动宾词组）的形式。例如："GetName()", "SetValue()", "Erase()", "Reserve()" .... | | 保护成员函数 | 保护成员函数的开头应当加上一个下划线“\_”以示区别，例如："\_SetState()" .... | | 私有成员函数 | 类似地，私有成员函数的开头应当加上两个下划线“\_\_”，例如："\_\_DestroyImp()" .... | | 虚函数 | 虚函数习惯以“Do”开头，如："DoRefresh()", "\_DoEncryption()" .... | | 回调和事件处理函数 | 回调和事件处理函数习惯以单词“On”开头。例如："\_OnTimer()", "OnExit()" .... | |

**变量**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 变量应该是程序中使用最多的标识符了，变量的命名规范可能是一套C++命名准则中最重要的部分：   |  |  | | --- | --- | | 变量的命名 | 变量名由作用域前缀＋类型前缀＋一个或多个单词组成。为便于界定，每个单词的首字母要大写。  对于某些用途简单明了的局部变量，也可以使用简化的方式，如：i, j, k, x, y, z .... | | 作用域前缀 | 作用域前缀标明一个变量的可见范围。作用域可以有如下几种：   |  |  | | --- | --- | | **前缀** | **说明** | | *无* | 局部变量 | | m\_ | 类的成员变量（member） | | sm\_ | 类的静态成员变量（static member） | | s\_ | 静态变量（static） | | g\_ | 外部全局变量（global） | | sg\_ | 静态全局变量（static global） | | gg\_ | 进程间共享的共享数据段全局变量（global global） |   除非不得已，否则应该尽可能少使用全局变量。 | | 类型前缀 | 类型前缀标明一个变量的类型，可以有如下几种：   |  |  | | --- | --- | | **前缀** | **说明** | | n | 整型和位域变量（number） | | e | 枚举型变量（enumeration） | | c | 字符型变量（char） | | b | 布尔型变量（bool） | | f | 浮点型变量（float） | | p | 指针型变量和迭代子（pointer） | | pfn | 特别针对指向函数的指针变量和函数对象指针（pointer of function） | | g | 数组（grid） | | i | 类的实例（instance）  对于经常用到的类，也可以定义一些专门的前缀，如：std::string和std::wstring类的前缀可以定义为"st"，std::vector类的前缀可以定义为"v"等等。 |   类型前缀可以组合使用，例如"gc"表示字符数组，"ppn"表示指向整型的指针的指针等等。 | | 推荐的组成形式 | 变量的名字应当使用"名词"或者"形容词＋名词"。例如："nCode", "m\_nState"，"nMaxWidth" .... | |

**常量**

|  |  |  |
| --- | --- | --- |
| C++中引入了对常量的支持，常量的命名规则如下：   |  |  | | --- | --- | | 常量的命名 | 常量名由类型前缀＋全大写字母组成，单词间通过下划线来界定，如：cDELIMITER, nMAX\_BUFFER ....  类型前缀的定义与变量命名规则中的相同。 | |

**枚举、联合、typedef**

|  |  |  |
| --- | --- | --- |
| 枚举、联合及typedef语句都是定义新类型的简单手段，它们的命名规则为：   |  |  | | --- | --- | | 枚举、联合、typedef的命名 | 枚举、联合、typedef语句生成的类型名由全大写字母组成，单词间通过下划线来界定，如：FAR\_PROC, ERROR\_TYPE .... | |

**宏、枚举值**

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 宏、枚举值的命名 | 宏和枚举值由全大写字母组成，单词间通过下划线来界定，如：ERROR\_UNKNOWN, OP\_STOP .... | |

# 6 注释

## 6.1 头文件注释

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright:bupt

Author:

Date:2010-08-25

Description:描述主要实现的功能

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## 6.2 函数头部注释

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description: // 函数功能、性能等的描述

Table Accessed: // 被访问的表（此项仅对于牵扯到数据库操作的程序）

Table Updated: // 被修改的表（此项仅对于牵扯到数据库操作的程序）

Input: // 输入参数说明，包括每个参数的作

// 用、取值说明及参数间关系。

Output: // 对输出参数的说明。

Return: // 函数返回值的说明

Others: // 其它说明

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## 6.3 其他注释规范

（1）关键代码注释，包括但不限于：赋值，函数调用，表达式，分支等等。  
（2）善未实现完整的代码，或者需要进一步优化的代码，应加上 // TODO …  
（3）调试的代码，加上注释 // only for DEBUG  
（4）需要引起关注的代码，加上注释 // NOTE …

# 7 代码风格

## 7.1 Tab和空格

每一行开始处的缩进只能用Tab，不能用空格，输入内容之后统一用空格。除了最开始的缩进控制用Tab，其他部分为了对齐，需要使用空格进行缩进。这样可以避免在不同的编辑器下显示不对齐的情况。  
 在代码行的结尾部分不能出现多余的空格。  
 不要在"::","->","."前后加空格。  
 不要在"，","；"之前加空格。

## 7.2 类型定义和{

类，结构，枚举，联合：大括号另起一行

## 7.3 函数

（1）函数体的{需要新起一行，在{之前不能有缩进。  
（2）除了特别情况，函数体内不能出现两个空行。  
（3）除了特别情况，函数体内不能宏定义指令。  
（4）在一个函数体内，逻揖上密切相关的语句之间不加空行，其它地方应加空行分隔。  
（5）在头文件定义的inline函数，函数之间可以不用空行，推荐用一个空行。

## 7.4 代码块

（1）"if"、"for"、"while"、"do"、"try"、"catch" 等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加 “{ }” 。这样可以防止书写和修改代码时出现失误。  
（2）"if"、"for"、"while"、"do"、"try"、"catch" 的括号和表达式，括号可紧挨关键字，这样强调的是表达式。

## 7.5 switch语句

（1）case关键字应和switch对齐。  
（2）case子语句如果有变量，应用{}包含起来。  
（3）如果有并列的类似的简单case语句，可考虑将case代码块写为一行代码。  
（4）简单的case之间可不用空行，复杂的case之间应考虑用空行分割开。  
（5）case字语句的大括号另起一行，不要和case写到一行。  
（6）为所有switch语句提供default分支。  
（7）若某个case不需要break一定要加注释声明。

## 7.6 类

（1）类继承应采用每个基类占据一行的方式。  
（2）单继承可将基类放在类定义的同一行。如果用多行，则应用Tab缩进。  
（3）多继承在基类比较多的情况下，应将基类分行，并采用Tab缩进对齐。  
（4）重载基类虚函数，应在该组虚函数前写注释 // implement XXX  
（5）友元声明放到类的末尾。

## 7.7 宏

（1）不要用分号结束宏定义。  
（2）函数宏的每个参数都要括起来。  
（3）不带参数的宏函数也要定义成函数形式。

## 7.8 goto

尽量不要用goto。