**C++** 编码规范

# 简介

## 标准化的重要性：

标准化让每个人在某些方面按同一个标准，使得每个人都感觉是在同一个场地。本文中的建议涵盖了多个项目、多个公司，对每个建议都经过了长时间的讨论。本文中的建议不是某部分人的风格，对局部修改是很开放的。

## 标准化的好处

程序员可以查看代码中每一个地方并知道它的作用新人员能尽快地加快速度

新人员可以省去建立自己的个人开发风格，直到老死。 。新人员可以避免不断地重复犯同样的错误

在一个稳定的环境里可以尽量少的范错误程序员都有一个共同的敌人。 。

## 标准化的坏处

标准经常由不懂 c++的人指定，而变成糟糕的标准标准经常不是我所做的，而变成糟糕的标准

标准会减少创造力

由于人员是稳定的而不需要标准标准强制了太多的结构

人员忽略了标准

## 说明

惯例

必须：表示任何项目都必须遵守该规范应该：表示根据项目对规范进行裁减 可以：表示可选的

# 名称

## 取一个合适的名称

名称是程序设计的核心。 在过去， 人们相信知道某个人的真实的名字会比知道那个人更具有神奇的魅力。如果你能想出某个事物的合适名称，将会给你自己和后来者带来更好的代码权。

一个名称是对一个事物所在环境进行长时间深刻思考后的结果。只有把一个系统作为一个整体来理解的程序员才能给出一个适合该系统的名称。

类名称

清楚的理解了该类是什么后才给类命名，如果不清楚该类是什么说明你的设计作的不够好； 如果一个名称由超过三个词组成则说明在系统中有多个实体冲突，应该重新设计；

一个类命名不应该与它的基类相关，只与该类本身相关； 添加后缀有时候是有帮助的；

方法和函数名称

通常每个函数或方法是执行一个动作，因此方法或函数名要清楚的表达出该方法或函数做了什么。

例如：用 CheckForErrors() 代替 ErrorCheck() ， DumpDataToFile() 代替 DataFile() ，这使得函数或方法更好的进行区分；

后缀有时候是有用的，例如： Max ， Cnt ， Key ， RetryMax 表示最大重复数， RetryCnt 表示当前重复

数；

前缀有时候是有用的，例如： Is， Get， Set， IsHitRetryLimit

不是所有的缩略名字母都需大写

如果名称中包含缩略名，该缩略名不要全部都用大写字母，而只要第一个字母大写，例如：

class FluidOz //NOT FluidOZ

class NetworkAbcKey //NOT NetworkABCKey

# 命名

## 类名

使用大写字母作为单词分隔符第一个字母大写

不使用字符 ’\_’

例如： class NameOneTwo

作为库的类名称，如果没有使用命名空间，则在类的名称前加上前缀，前缀可以使用作者的名字缩写，例如：张三写的类， class ZsNameOneTwo

* 1. 类方法名

使用与类名相同的规则，例如：

class NameOneTwo

{

public:

int DoIt();

void HandleError();

}

* 1. 类属性名

类属性名在前面加 m

在 m 后面的规则与类名规则相同

m 总是在其它前缀之前，例如指针 ’p’

class NameOneTwo

{

public:

int VarAbc();

int ErrorNumber();

private:

int mVarAbc;

int mErrorNumber;

String\* mpName;

}

* 1. 方法参数名

第一个字符大写

在第一个字符后面的所有单词的第一个字符大写

class NameOneTwo

{

public:

int StartYourEngines(Engine& rSomeEngine, Engine& rAnotherEngine);

}

* 1. 局部变量名

全部使用小写字符

使用字符 ’\_’作为单词分隔符

int

NameOneTwo::HandleError(int errorNumber)

{

int error= OsErr(); Time time\_of\_error; ErrorProcessor error\_processor;

}

* 1. 指针变量名

加前缀 ’p’

‘\* ’应靠近类型名

String\* pName= new String; String\* pName, name, address; / 注/

* 1. 引用变量名和引用返回

加前缀 ’r ’

‘& ’应靠近类型名

class Test

{

public:

意：只有 pName 是指针

void DoSomething(StatusInfo& rStatus);

StatusInfo& rStatus(); const StatusInfo& Status() const;

private:

StatusInfo& mrStatus;

}

* 1. 全局变量名

在最前面加 ’g’ Logger gLog; Logger\* gpLog;

* 1. 全局常量名

全部字符大写

用’\_’作为单词分隔符

const int A\_GLOBAL\_CONSTANT= 5;

* 1. 静态变量名

加前缀 ’s’

class Test

{

public:

private:

static StatusInfo msStatus;

}

* 1. 类型名

在名字后面加上 ’Type’ typedef uint16 ModuleType; typedef uint32 SystemType;

* 1. #define 和宏名

全部字符大写

用’\_’作为单词分隔符 #define MAX(a,b) blah #define IS\_ERR(err) blah

* 1. C 函数名

在 c++工程中使用的 C 函数

全部字符小写，用 ’\_’作为单词分隔符

int

some\_bloody\_function()

{

}

* 1. 枚举名

enum PinStateType

{

PIN\_OFF, PIN\_ON

}

1. 其它
   1. {} 策略

if (condition) while (condition)

{ {

... ...

} }

if (condition) { while (condition) {

... ...

} }

## Tabs/Space

缩进用 Space 代替 Tabs，使用 3、4 或 8 个 Space Tabs应该固定为 8 个 Space

## 一个类必须定义的方法

默认构造函数

虚拟析构函数：如果该类可被其它类继承，则声明该类的析构函数为虚拟析构函数

拷贝构造函数：如果该类可拷贝，要么自己定义要么注释说明使用编译器生成的拷贝构造函数；如果该类对象不可拷贝，把拷贝构造函数声明为私有不要定义函数体；如果不确定则默认为不可拷贝赋值操作符：如果该类对象可赋值，要么自己定义要么注释说明使用编译器生成的赋值操作符；如果该类对象不可赋值，把赋值操作符声明为私有不要定义函数体；如果不确定则默认为不可赋值

## 构造函数使用

在构造函数中不要做任何实际的工作，在构造函数中初始化变量时确保这些操作不会失败创建一个 Open() 函数来完成构造， Open()函数应该在对象实例化后再调用，例如：

class Device

{

public:

Device() { /\* initialize and other stuff \*/ } int Open() { return FAIL; }

};

Device dev;

if (FAIL == dev.Open()) exit(1);

## 防止头文件重复包含

#ifndef ClassName\_h #define ClassName\_h

#endif // ClassName\_h

## Include 语句说明

每个包含的头文件都要说明原因，如果被包含的头文件有关联的类，则说明与该类得关系： ISA 、HASA

和 USES，例如： #ifndef XX\_h

#define XX\_h

// SYSTEM INCLUDES

//

#include // standard IO interface

#include // HASA string interface

## IF Then Else 格式

if (condition) // Comment

{

}

else if (condition) // Comment

{

}

else // Comment

{

}

条件格式：

if ( 6 == errorNum ) ...

## Switch 格式

switch (...)

{

case 1:

...

// FALL THROUGH

case 2:

{

int v;

...

}

break; default:

}

## Goto,continue,break,?: 的使用

Goto for (...)

{

while (...)

{

}

}

...

error:

...

if (disaster)

goto error;

clean up the mess Continue ， break

while (TRUE)

{

...

// A lot of code

...

if (/\* some condition \*/) { continue;

}

...

// A lot of code

...

if ( i++ > STOP\_V ALUE) break;

}

?:

(condition) ? funct1() : func2(); or

(condition)

? long statement

: another long statement;

## 声明块对齐

DWORD mDword

DWORD\* mpDword

char\* mpChar

char mChar

mDword = 0;

mpDword = NULL;

mpChar = 0;

mChar = NULL;

## 块注释

在块的开始和结束处都要注明，例如：

{

// Block1 (meaningful comment about Block1)

... some code

{

// Block2 (meaningful comment about Block2)

... some code

} // End Block2

} // End Block1

## 类头文件模板

/\*\*

* + - A one line description of the class.

\*

* + - #include "XX.h" <BR>
    - -llib

\*

* + - A longer description.

\*

* + - @see something

\*/

#ifndef XX\_h #define XX\_h

// SYSTEM INCLUDES

//

// PROJECT INCLUDES

//

// LOCAL INCLUDES

//

// FORWARD REFERENCES

//

class XX

{

public:

// LIFECYCLE

/\*\*

\* Default constructor.

\*/ XX(void);

file:///E|/Computer/Program%20Design/C%20And%20C++/C++%20Coding%20Standard.htm (22 of 73) [2003-4-9

20:52]C++ Coding Standard

/\*\*

\* Copy constructor.

\*

\* @param from The value to copy to this object.

\*/

XX(const XX& from);

/\*\*

\* Destructor.

\*/

~XX(void);

// OPERATORS

/\*\*

* Assignment operator.

\*

* + @param from THe value to assign to this object.

\*

* + @return A reference to this object.

\*/

XX& operator=(XX& from);

// OPERA TIONS

// ACCESS

// INQUIRY

protected:

private:

};

// INLINE METHODS

//

// EXTERNAL REFERENCES

//

#endif // \_XX\_h\_

## 类源文件模板

#include "XX.h" // class implemented

/////////////////////////////// PUBLIC ///////////////////////////////////////

//============================= LIFECYCLE ==================================== XX::XX()

file:///E|/Computer/Program%20Design/C%20And%20C++/C++%20Coding%20Standard.htm (29 of 73) [2003-4-9 20:52]C++ Coding Standard

{

}// XX XX::XX(const XX&)

{

}// XX

XX::~XX()

{

}// ~XX

//============================= OPERA TORS ====================================

XX& XX::operator=(XX&);

{

return \*this;

}// =

//============================= OPERA TIONS ===================================

//============================= ACESS ===================================

//============================= INQUIRY ===================================

/////////////////////////////// PROTECTED ///////////////////////////////////

/////////////////////////////// PRIV ATE ///////////////////////////////////

## 类方法模板

/\*\*

* + - Assignment operator.

\*

* + - @param val The value to assign to this object.

\*

* + - @return A reference to this object.

\*/

XX& operator=(XX& val);

可以添加附加说明：

/\*\*

* + - Copy one string to another.

\*

* + - PRECONDITION
    - REQUIRE(from != 0)
    - REQUIRE(to != 0)

\*

* + - WARNING
    - The to buffer must be long enough to hold
    - the entire from buffer.

\*

* + - EXAMPLES

\*

* + - strcpy(somebuf, "test")

\*

\*

* + - @param from The string to copy.
    - @param to The buffer to copy the string to.

\*

* + - @return void

\*/

void strcpy(const char\* from, char\* to);