C 和 C++编程规范

作者：		日期：	
审批：		日期：	

目 录

1 引言

1.1 文档用途

本文对 C 以及 C++ 的编程的规则和惯例进行说明，以规范后台 Unix 环境软件的编码。以下定义的各种规则在某些情况下并不是绝对适用，如果按照规则来编写 C 和 C++的源程序，可以避免很多不必要的错误。遵循以下的规则将有助于源程序的阅读和修改，增强软件的可读性、可维护性和移植性。

1.2 阅读对象

此文档适合以下人员阅读：

- 产品开发工程师
- 产品维护人员

1.3 参考资料

2 变量命名规则

2.1 构成方式

命名的构成方式为[变量作用域] + 变量类型 + [基本名]

说明：

1. 变量作用域表明变量的作用范围，用小写，其后跟“_”。
2. 变量类型表明变量的数据类型，用小写；
3. 基本名，由一个或数个单词组成，构成“主谓”、“动宾”等形式，单词的第一个字母大写，其他字母小写，如 Count、RatedCharge。

2.2 变量作用域

变量类型	前缀	举例	注释
全局变量	g	g_nMaxBorrowDay	1. g 表明此变量是全局变量 2. n 指此变量是一整数 3. MaxBorrowDay 是基本名
局部变量	无	nCount	1. n 指此变量是一整数 2. Count 是基本名
类成员变量	m	m_nStudentCount	1. m 表明此变量是类成员变量 2. n 指此变量是一整数 3. StudentCount 是基本名

2.3 变量类型

标志	类型描述	举例
b	boolean flag (TRUE, FALSE)	BOOL bAbort;
c	Character	char cInput;
n	16-bit signed integer	int16 nCount;
i	32-bit signed integer	int32 iAmount;
ll	64-bit signed integer	int64 llAmount;
str	String	string strBuf;
sz	zero-terminated character array	char* szString;
p	Point	int32* pNumber;
a	array	asLine[2];
f	Float	fAmount;

d	double	dExRate
fp	filepointer(File *)	fpFileHandle;

2.4 特别情况

在函数、事件等过程中，例如，循环中用到的临时变量可直接用 i、j、k 等形式命名变量。

3 其他命名规则

3.1 函数和过程

动词小写后加“_”分隔符，“_”后所跟的第一个词小写其他字头大写,只允许一个分隔符。如：get_inputString()

3.2 Class(类)

系统表类，每单词的首字母大写，其它字母小写，不加分隔符，前跟“AI”如：AILocalPlan；其它类，字头大写，不加分隔符，前跟“C”。

3.3 Structure(结构)

每单词的首字母大写，其它字母小写，不加分隔符，前跟“S”如：SRatedCdr；说明：如无特别必要，Struct 的命名可以遵循 Class 的命名。Struct 中的数据的命名遵循 Class 的 Member 的命名。

3.4 Interface(Corba 接口)

每单词的首字母大写，其它字母小写，不加分隔符，前跟“I”如：IBusiAccept；说明：如无特别必要，Interface 的命名可以遵循 Class 的命名。Interface 中的数据的命名遵循 Class 的 Member 的命名。Interface 中使用到的 Structure（结构）的命名规则同 3.3 中的说明

3.5 常数

全部大写，词与词之间以“_”为分隔符。如：MAX_DAY_COUNT

3.6 全局标识符

所有的全局类、全局枚举类型、全局类型定义、全局函数、全局常数、全局变量必须带模块前缀，以保证其全局唯一性。

4 注释

4.1 源程序头

每个源代码文件(包括.c, .cpp, .h, .hpp, .x, .xpp)必须在文件的头部描述文件名、内容、修改记录。修改记录包括日期(YYYY/MM/DD)、创建者/修改者、修改内容，新增的修改记录排在最前。

源程序头的注释格式采用 doxygen 风格的形式：

例如：

```
/**
 * @file          cccom.cpp
 * @brief
 * This file defines the proxy for all communication classes. Also possible
 * events and errors are defined.
 *
 * History
 *    1996/10/25 ReWe    integration of missing return values
 *    1996/10/14 Andreas Welsing #pragma pack() inserted
 *    1996/06/13 ReWe all timers are set to -1
 *    1995/10/25 ReWe first release
 */
```

4.2 函数或过程

在每个函数或过程的前头，要对函数或过程进行以下方面的些注释：

注释标记	注释描述
@author	作者说明
@brief	函数或过程的功能描述
@param	参数说明
@return	函数返回值的说明
@see	相关文件说明

例如：

```
/**
 * @brief
 * Draws as much of the specified image as is currently available
 * with its northwest corner at the specified coordinate (x, y).
 * This method will return immediately in all cases, even if the
 * entire image has not yet been scaled, dithered and converted
 * for the current output device.
 *
 * @author          Sami Shaio
 * @author          Arthur van Hoff
 * @param img       the image to be drawn
 * @param x         the x-coordinate of the northwest corner
 *                  of the destination rectangle in pixels
 * @param y         the y-coordinate of the northwest corner
 *                  of the destination rectangle in pixels
 * @return          <code>true</code> if the image is completely
 *                  loaded and was painted successfully;
 *                  <code>false</code> otherwise.
 * @see             Image
 * @see             ImageObserver
 */
```

4.3 代码修改

代码更新时，需要在 SVN 的 Log Info 中说明姓名、日期、更新原因。

4.4 其他注释

- 变量注释和一般语句注释，注释内容直接跟在语句后。

  例：`char sUserName[30];  // the user name`

  `if( condition )  //your comments`

- 对代码段的注释信息在代码段前说明。

  例：

```
/* …… */
if（ a == b ）
{
.....................
}
```

5 代码风格和版式约定

5.1 一般约定

1) 每个函数的代码行数控制在 60 左右，最好不要超过 300 行。

2) 类的成员个数控制在 15 个以内，最好不要超过 30 个。

3) 注释不允许嵌套。

4) 对多条件的判断语句，每个条件语句和子条件要用附上括号。

如：if((a == 0) && ((b == TRUE) || (c == d)))

5) 不同作用域的变量名的基本名不许相同。

6) 每行的代码长度最好小于 80 字符，若超过 80，用续行符换行或自然的逗号分隔处换行。

7) 同一层次的代码要有相同的缩进值，用 TAB 控制缩行值，不要 TAB 和 SPACE 混用，TAB 使用缺省值 4。

8) 同一层次的‘{’、‘}’不能在同一行代码中。

9) 在一个函数或过程中的嵌套层次最好不要超过三层，最多不超过六层。

10) 要显式地给出函数或过程的返回值类型。

11) 避免语句中字符和整型变量的直接比较，要显式地进行类型转换。

12) 常量定义应该用 const，如： const unsigned MAX_TBL_COUNT = 10;或使用 enum 类型（参考下条中的例子）

13) 若常量只在类中使用，应定义在类中

如：

```
class CBSRevApi
{
public:
enum { MAX_TBL_COUNT = 10;    }
…
};
```

14) 指针和引用：在定义指针和引用时，*符号和&符号紧跟在类型名后面，每一行只能定义一个变量，否则会产生错误，如：

```
// NOT RECOMMENDED
char* i,j; // i is declared pointer to char, while j is declared char
```

15) 函数参数尽可能使用引用，对于输入型的参数（不修改参数状态），用 const 引用，

例如：

```
// a. A copy of the argument is created on the stack.
// The copy constructor is called on entry,
// and the destructor is called at exit from the function.
// This may lead to very inefficient code.

void foo1( String s );
String a;
foo1( a ); // call-by-value

// b. The actual argument is used by the function
// and it can be modified by the function.

void foo2( String& s );
String b;
foo2( b ); // call-by-reference

// c. The actual argument is used by the function
// but it cannot be modified by the function.

void foo3( const String& s );
String c;
foo3( c ); // call-by-constant-reference
```

16) 内存申请和释放：尽可能避免使用 c 风格的 malloc,realloc,free，使用 new/delete；在释放数组时，用 delete[]；

5.2 头文件

为避免头文件多次包含，在文件头注释之后包含下面两行：

```
#ifndef _FILENAME_H
#define _FILENAME_H
```

在文件末尾包含下面一行：

```
#endif // _FILENAME_H
```

其中 FILENAME 为头文件名，如果头文件在该工程的 include 目录的下一级子目录，则

在文件名前增加“子目录名_”，如：

include/util

则用下面的格式：

```
#ifndef _UTIL_FILENAME_H
#define _UTIL_FILENAME_H
……
#endif // _UTIL_FILENAME_H
```

5.3 类风格约定

1）不要定义 public 或 protected 数据成员，以充分发挥 C++中的数据封装功能；

2）const 成员函数：不改变对象类别数据和状态的成员函数应该被定义为 const，例如：

```
class SpecialAccount : public Account
{
public:
int insertMoney();
// int getAmountOfMoney(); No! Forbids ANY constant object to
// access the amount of money.
int getAmountOfMoney() const; // Better!
// ...
private:
int moneyAmount;
};
```

3）构造和析构函数：需要用 new 分配实例的类应该定义 copy 构造函数；

4）赋值操作符：

需要用 new 分配实例的类应该定义赋值操作符，同时该操作符应该返回一个 rhs 对象的一个 const reference，以组织 a=b=c 这种连等的用法，例如：

```
const MySpecialClass&
MySpecialClass::operator=( const MySpecialClass& msp ); // Recommended
```

5）成员函数：

public 成员函数不能返回一个 non-const 型的成员数据的引用或指针，例如：

```
class Account
{
public:
    Account( int myMoney ) : moneyAmount( myMoney ) {};
```

```
    const int& getSafeMoney() const { } //  正确
    int& getRiskyMoney() const { return moneyAmount; } //  不允许!
    // ...
 private:
    int moneyAmount;
 };

inline const int&
Account::getSafeMoney() const
{
    return moneyAmount;
}
```

6) 在类的定义中不要写实现，用 inline 函数实现

6 关于跨平台和编译器的处理

6.1 系统配置头文件 compile.h

compile.h 是类似与 autoconfig 生成的配置头文件，用于定义操作系统、硬件平台、编译器等相关的宏，针对不同的平台，会有一个不同的 compile.h。考虑到操作的方便性，在本次 OPENBOSS 的开发中不使用 autoconfig 和 automake，compile.h 手工构造。

每一个 c/c++源文件的头上，必须第一个包含 compile.h。

6.2 关于 int32 和 int64

在程序中对外部接口函数和数据库数据交换，必须显式使用 int32 和 int64 数据类型，不要用 int,long,long long 这三种数据类型，原则如下表：

ORACLE 数据类型	C 数据类型
number(1) – number(9)	int32
number(10) – number(15)	int64

其中 int64 只能用到 number(15)主要是因为在 ORACLE 的接口中，number 是用 double 实现的，而 double 只有 15 位有效整数位数。

int32 和 int64 定义在 compile.h 中。

7 其他参考惯例

除了上述约定外，其他未涉及的内容请参阅下面的附件，有矛盾或冲突的地方，以本文为主。

附件：《Programming in C++, Rules and Recommendations》

附件文件名：Ellemtel-rules-mm.html