**C++代码的编写规范：**

头文件

1、#define保护

所有头文件应使用#define防止文件被多重包含，命名格式应为

<PROJECT>\_<PATH>\_<FILE>\_H。

如：项目foo中的foo/src/bar/baz.h的保护应为

#ifndef FOO\_BAR\_BAZ\_H

#define FOO\_BAR\_BAZ\_H

...

#endif //FOO\_BAR\_BAZ\_H

2、头文件以来

尽量减少.h中的#include的数量。

如：头文件总用到类File，但不需要访问File的声明，则头文件中只需声明class File即可。

3、函数内联

当函数小于等于10行时可将其定义为内联函数；

内联函数尽量不要包含循环或switch语句，得不偿失；

递归函数不要声明为内联函数；

4、-inl.h文件

可以存放内联函数的定义；

还可以存放函数模板的定义，增强可读性；

5、函数参数顺序

建议：输入参数在前，输出参数在后；

6、包含文件的名称及次序

建议次序：C库、C++库、其他库的.h、项目内的.h；

如：google-swesome-project/src/foo/internal/fooserver.cc的包含顺序：

#include "foo/public/fooserver.h"

#include <sys/types.h>

#include <unistd.h>

#include <hash\_map>

#include <vector>

#include "base/basictypes.h"

#include "foo/public/bar.h"

作用域

1、命名空间

谨慎使用命名空间，防止命名空间污染；

建议：.cc文件中使用不具名命名空间，以避免运行时的命名错误；

2、嵌套类

不要讲嵌套类定义为public；

3、非成员函数、静态成员函数和全局函数

使用命名空间中的非成员函数或静态成员函数，尽量不要使用全局函数；

4、局部变量

将函数变量尽可能置于最小作用域内，在声明变量时将其初始化；

5、全局变量

禁止class类型的全局变量；

C++类

1、构造函数的职责

构造函数只进行那些没有实际意义的初始化，尽量使用Init()方法几种初始化为有意义的数据；

2、默认构造函数

如果类中定义了成员变量，没有提供其他构造函数，你需要定义一个默认构造函数。

3、明确的构造函数

对单参数构造函数使用C++关键字explicit；

4、拷贝构造函数

尽在代码中需要拷贝一个类对象的时候使用拷贝构造函数；不需要拷贝时应使用DISALLOW\_COPY\_AND\_ASSIGN。

DISALLOW\_COPY\_AND\_ASSIGN的定义：

#define DISALLOW\_COPY\_AND\_ASSIGN(TypeName)\

TypeName(const TypeName&);\

void operator=(const TypeName&)

DISALLOW\_COPY\_AND\_ASSIGN的使用：

class Foo{

public:

Foo(int f);

~Foo();

private:

DISALLOW\_COPY\_AND\_ASSIGN(Foo);

}

5、结构体和类

仅当只有数据时使用struct，其他一概使用class；

6、继承

使用组合通常比使用继承更适宜，如果使用继承的话，只使用公共继承；

7、多重继承

真正需要多重继承的时候非常少，只有当最多一个基类中含有实现，其他基类都是以Interface为后缀的纯接口类时才使用多重继承；

8、接口

纯接口：

1）只有纯虚函数和静态函数（析构函数除外）；

2）没有非静态数据成员；

3）没有定义任何构造函数；

4）如果是子类，也只能继承满足上述条件并以Interface为后缀的类；

9、操作符重载

一般不要重载操作符，尤其是赋值操作（operator=）比较阴险，应避免重载

10、存取控制

将数据成员私有化，并提供相关存取函数；

11、声明次序

public在private之前，成员函数在数据成员前；

一般次序：

1）typedefs和enums；

2）常量；

3）构造函数；

4）析构函数；

5）成员函数，含静态成员函数；

6）数据成员，含静态数据成员；

宏DISALLOW\_COPY\_AND\_ASSIGN置于private块之后，作为类的最后部分；

12、编写短小函数

倾向于选择短小、凝练的函数，函数尽量不要超过40行；

智能指针和其他C++特性

1、智能指针

如果需要智能指针的话，scoped\_ptr完全可以胜任，在STL容器中，只使用std::tr1::shared\_ptr，任何情况下都不要使用auto\_ptr；

其他C++特性

1、引用参数

按引用传递的参数必须加上const；

2、函数重载

尽在输入参数类型不用、功能相同时使用重载函数，不要使用函数重载模仿缺省函数参数；

3、缺省参数

禁止使用缺省函数；

4、变长数组和alloca

禁止使用变长数组和alloca()；

5、友元

允许合理使用友元类及友元函数；

6、异常

不要使用C++异常；

7、运行时类型识别

禁止使用RTTI(运行时类型识别，Run-Time Type Information)；

8、类型转换

使用static\_cast<>()等C++的类型转换，不要使用int y = (int)x；

9、流

只在记录日志时使用流

10、前置自增和自减

对于迭代器和模板类型来说，要使用前置自增；

11、const的使用

建议在任何可以使用的情况下都要使用const；

12、整型

可以使用<stdint.h>中的精确宽度的整型，如int16\_t；

13、64位下的可移植性

14、预处理宏

使用宏要谨慎，尽量以内联函数、枚举和常量代之；

15、0和NULL

整数用0，实数用0.0，指针用NULL，字符串用'\0'；

16、sizeof(sizeof)

尽可能用sizeof(var)代替sizeof(type)；

17、Boost库

只使用Boost中被认可的库；

命名约定

1、通用命名规则

函数名、变量名、文件名应具有描述性，不要过度缩写；

2、文件命名

文件名要全部小写，可以包含下划线\_或短线-，按项目约定来；

3、类型命名

类型命名每个单词以答谢字母开头，不包含下划线；

类型包括——类、结构体、typedef、枚举；

4、变量命名

变量名一律小写，单词间以下划线相连，类的成员变量以下划线结尾；

全局变量以g\_为前缀；

5、常量命名

在名词前加k:kDayInAWeek；

6、函数命名

普通函数大小写混合，存取函数要求与变量名匹配：MyExcitingFunction()、set\_my\_exciting\_member\_variable();

7、命名空间

命名空间的名词是全小写，其命名基于项目名词和目录结构；

8、枚举命名

枚举值应全部大写，单词间以下划线相连；

9、宏命名

类似枚举命名一样全部大写、使用下划线；

代码注释

1、注释风格

使用//或/\* \*/，统一就好；

2、文件注释

在每一个文件开头加入版权公告，然后是文件内容描述；

次序：

1）版权：Copyright 2008 Google Inc;

2）许可版本：Apache 2.0;

3）作者：标识文件的原始作者；

4）文件内容；

3、类注释

每个类的定义要附着描述类的功能和用法的注释；

4、函数注释

函数声明处注释描述函数功能，定义处描述函数实现；

5、变量注释

通常变量名本身足以很好说明变量用途；

6、实现注释

对于实现代码中巧妙的、晦涩的、有趣的、重要的地方加以注释；

7、标点、拼写和语法

留意标点、拼写和语法，写的好的注释比差的要易读的多；

8、TODO注释

对那些临时的、短期的解决方案，或已经够好但并不完美的代码使用TODO注释；

格式

1、行长度

每一行代码字符数不超过80；

2、非ASCII字符

尽量不适用非ASCII字符，使用时必须使用UTF-8格式；

3、空格还是制表位

只使用空格，每次缩进2个空格；

使用空格进行缩进，不要在代码中使用tabs，设定编辑器将tab转为空格；

4、函数声明与定义

返回类型和函数名在同一行，合适的话，参数也放在同一行；

5、函数调用

尽量放在同一行，否则，将实参封装在圆括号中；

6、条件语句

提倡不在圆括号中添加空格，关键字else另起一行；

7、循环和开关选择语句

switch语句可以使用大括号分块；空循环应使用{}或continue；

8、指针和引用表达式

句点.或箭头->前后不要有空格，指针/地址操作符\*、&后不要有空格；

9、布尔表达式

如果一个布尔表达式超过标准行宽80字符，如果断行要统一一下；

10、函数返回值

return表达式中不要使用圆括号；

11、变量及数组初始化

选择=还是()；

12、预处理指令

预处理指令不要缩进，从行首开始；

13、类格式

声明属性次序是public、protect、private，每次缩进1个字符；

14、初始化列表

构造函数初始化列表放在同一行或四格缩进并排几行；

15、命名空间格式化

命名空间内容不缩进；

16、水平空白

水平空白的使用因地制宜。不要在行尾添加无谓的空白；

17、垂直空白

垂直空白越少越好；