百度 C++编码规范

命名

定义：蛇形命名法

用下划线分隔全小写英文单词的命名规则，例如：a_foo_function(int user_age)

定义：大写蛇形命名法

用下划线分隔全大写英文单词的命名规则，例如：A_FOO_CONSTANT

定义：驼峰命名法

首字母大写英文单词，连续单词之间无分隔，例如：User，UserName

实体	命名规则	示例
命名空间（name space）	蛇形命名法	namespace_name
函数名	蛇形命名法	function_name
类名	驼峰命名法	ClassName
结构体		TypeName
枚举		EnumerationName
全局变量	蛇形命名法，加前缀 g_	g_global_variable
实例变量（instance variable）	蛇形命名法，加前缀 m_	m_instance_variable
局部变量	蛇形命名法	local_varialbe
const 常量	大写蛇形命名法	CONSTANT_NAME
枚举常量		ENUMERATION_CONSTANT
宏		MACRO_NAME

原则：名字缩写

所有命名推荐使用完整单词，不使用缩写，除非该缩写已成为惯用法，例如：

. 蛇形命名时：使用 product 不允许 prod

. 驼峰命名时：使用 DepartmentManager 不允许 DptMgr

可以使用的惯用缩写：

. 产品线、项目组、功能名称，例如：fc, im, fcr, imbs, qs, bmm（高级短语）

. 语言中惯用缩写，例如：pointer 缩写为 ptr, variable 缩写为 var，value 缩写为 val

格式

定义：小括号

英文输入法，数字 9 和 0 的上档键：() —— 分别称为“左小括号”和“右小括号”

定义：大括号

英文输入法，符号 [和] 的上档键：{} —— 分别称为“左大括号”和“右大括号”

定义：单语句

需要用英文分号（;）结尾的表达式。

原则：左大括号位置

任何声明中，左大括号一律放在声明语句行尾，之前用一个空格间隔。

. 每一条单语句独立成行。
. 每行宽度不超过 80 个字符。
. 使用空格缩进，不使用制表符。
. 以 4 个空格为单位缩进。

. 注释：

```
// Copyright (c) ...
// @author ...
// @brief ...

// 函数职责说明
// 参数含义说明
void foo_function(int value) {
    //...
}
```

. 所有注释都使用行注释 //，不使用块注释 /**/。
. 行注释符（//）和注释内容之间用一个空格间隔。

```
int index = 0;  // 注释内容
```

. 代码行尾的注释，和代码之间至少用两个空格间隔。

```
for (int index = 0; index < 100; ++index) {
    a_funcation();       // 相关行注释
    another_function();  // 推荐缩进对齐
}
```

. 逻辑相关语句行尾的注释，推荐缩进对齐。

原则：大括号的使用

推荐语句块都用大括号包围。当且仅当语句块中只有一行单语句时，可以省略大括号。

. 命名空间（name space）声明：

```
namespace foo_name_space {
namespace bar_name_space {
class MyClass {
    //...
};
}  // namespace bar_name_space

namespace buz_name_space {
// ...
}  // namespace buz_name_space
}  // namespace foo_name_space
```

. 嵌套的命名空间声明无缩进。
. 命名空间内容无缩进
. 命名空间右括号另起一行。
. 命名空间右括号右边空格后注释：// namespace <命名空间名称>

. 赋值语句：

```
int user_age = 33;
```

. 等号左右各用一个空格间隔。

原则：操作符左右的间隔

除了递增（++）和递减（--），所有操作符左右各用一个空格间隔。

. if 语句：

```
if (age > 100) {
    // 语句块
}
// 用一行空行间隔
if (abs(value) < EPSILON) {
    // 语句块
}

if (!is_finished) {
    // 语句块
}

if (value == 0) {
    // 语句块
} else if (value == 1) {
    // 语句块
} else {
    // 语句块`
}
```

. 关键字 if、else、else if 和条件表达式之间用一个空格间隔。
. 条件表达式中运算符左右各用一个空格间隔
. 语句块用大括号包围。
. 语句块右大括号另起一行，缩进和关键字 if 对齐。
. 多个连续 if 语句之间用一行空行间隔。
. 关键字 else、else if 和上一语句块右大括号在同一行，之间用一个空格间隔。
. 连续 if-else-else if 之间无空行分隔。

```
if (val > 0)
    return true;

if (val > 0)
    do_something();
else
    do_other_thing();
```

. 语句块中只有一行单语句时，可以省略大括号。
. 语句块中单语句回行后，缩进一层。

```
if ( a == b && c == d
        || e == f) {  // 如果换行则缩进两层
    // 函数体
}
```

. 条件表达式如果换行，则缩进两层。

```
if (!is_true(arg)) {
    // 函数体
}

if (function_one(arg_1, arg_2, arg_3),
        function_two(arg_1, arg_2, arg_3), other_arguments) {
    // 函数体
}
```

. 条件表达式内函数调用当作（if 语句的）小括号内内容。
. 如果表达式内容换行，则缩进两层。

原则：小括号内语句的换行

推荐小括号内语句尽量简短并放在一行。如果小括号内语句换行，则缩进两层。

. 循环语句 for：

```
for (int index = 0; index < 100; ++index) {
  // 循环体
}
```

. 关键字 for 和条件表达式之间用一个空格间隔
. 循环体用大括号包围。
. 循环体右大括号另起一行，缩进和关键字 for 对齐。
. 循环条件语句的三段式之间用一个空格间隔。
. 循环体三段式内，操作符左右各用一个空格间隔。
. 循环体三段式中递增操作符（++）和变量之间无空格。

```
for (int x = 0, y = 1, z = 2;
        x < 100, y < 200, z < 300;  // 换行后缩进两层
        ++x, ++y, ++z) {
    // 循环体
}
```

. 循环条件语句如果换行，则换行后缩进两层（8 个空格）。
. 之后的换行和上一行对齐。

. 循环语句 while：

```
while (foo_value > 100) {
    // 循环体
}
```

. 关键字 while 和条件表达式之间用一个空格间隔。
. 条件表达式内，操作符左右各用一个空格间隔。
. 循环体用大括号包围。

. 循环体右大括号另起一行，缩进和关键字 while 对齐。

. 循环语句 do-while:

```
do {
    // 循环体
} while (bar_value > 100)
```

. 循环体用大括号包围。
. 循环体右大括号另起一行，缩进和关键字 do 对齐。
. 关键字 while 和循环体右大括号在同一行，之间用一个空格间隔。
. 关键字 while 和条件表达式之间用一个空格间隔。
. 条件表达式内，操作符左右各用一个空格间隔。

. switch 语句：

```
switch (user_age) {
    // 分支语句块
    case 20: {
        // 分支块
        break;
    }
    case 30: {
        //...
        break;
    }
    default: {
        //...
        return result;
    }
}
```

. 关键字 switch 和条件表达式之间用一个空格分隔。
. 整体 switch 语句块用大括号包围。
. 每一条 case 分支块用大括号包围。
. 每一条 case 分支块右大括号另起一行，缩进和关键字 case 对齐。

```
switch (condition) {
    case 0:
        return val_0;
    case 1:
        return val_1;
    default:
        return default_val;
}
```

. 如果 case 语句块只包含一行单语句，可以省略大括号。

. 函数声明：

```
int bar_function() {
    // 函数体
}
```

. 返回值和函数名在同一行。
. 函数名和参数列表左小括号之间无空格。
. 参数列表右小括号和函数左大括号在同一行。
. 参数列表右小括号和函数左大括号之间用一个空格间隔。
. 多个函数声明之间用一行空行间隔。

```
int foo_function(int foo_age, double bar_arg) {
    // 函数体
}
```

. 参数列表如果在一行，则：
  . 参数声明之间用一个空格间隔。

```
92  void foo(
93          int arg_1,  // 参数换行后两层缩进
94          int arg_2) {
95      // 函数体一层缩进
96  }
97
98  void a_function_has_a_very_long_and_complex_name(
99          int arg_1,
100         int arg_2) {
101     // 函数体
102 }
103
104 void a_function_has_a_long_arg_list(
105         int arg_1,
106         int arg_2,
107         int arg_3,
108         int arg_4,
109         int arg_5) {
110     // 函数体
111 }
```

. 参数如果换行，则：
  . 从第一个开始换行。
  . 换行后缩进两层（8 个空格）
  . 之后的参数和第一个参数的缩进对齐。
  . 每行有且只有一个参数。

原则：函数参数的位置

函数参数在位置上保持整体，即：要么都在一行，要么从第一个开始换行。

. 预处理指令

```
251 void function() {
252     if (lopsided_score) {
253 #if DISASTER_PENDING      // 预处理指令无缩进
254         DropEverything();
255 #if NOTIFY
256         NotifyClient();
257 #endif
258 #endif
259         BackToNormal();
260     }
261 }
```

  . 预处理指令无缩进。

. 类声明：

```
89  class MyClass {
90      // 类体
91  };
```

. 关键字 class、类名、左大括号在同一行。
. 右大括号另起一行，缩进和关键字 class 对齐。
. 访问控制符 public/protected/private 声明：
. 访问控制符声明：

```
146 class MyClass {
147 public:
148     void foo() {
149         // ...
150     }
151
152 protected:
153     void bar() {
154         // ...
155     }
156
157 private:
158     void buz() {
159         // ...
160     }
161 };
```

. 访问控制符声明顺序为：public, protected, private
. 访问控制符声明无缩进，和关键字 class 对齐。
. 访问控制符声明行前用一行空格间隔。

. 结构体声明：

```
93 typedef struct {
94     int x;
95     int y;
96 } Point;
```

. 使用 typedef 定义结构体。
. 右大括号另起一行，缩进和关键字 class 对齐。
. 右大括号和结构体名之间用一个空格间隔。

枚举声明：

```
 98 enum Day { SUNDAY = 0, MONDAY, TUESDAY, /*..*/ }
 99
100 enum LightSpectrum {
101     INFRARED_RAY,
102     VISIBLE_LIGHT,
103     ULTRAVIOLET_RAY,
104     X_RAY,
105     //...
106 }
```

. 一行定义时，各个常量之间用一个空格间隔。
. 常量需要赋值时，等号左右各有一个空格间隔。

文件和目录

. 所有代码文件以 UTF-8 为编码格式。
. 头文件命名：<文件名>.h
. 源文件命名：<文件名>.cpp
. 模板实现文件命名：<文件名>.hpp
. 单元测试文件命名：<文件名>_test.cpp

. 文件名都用蛇形命名法。

. 源文件目录结构和 namespace 结构一致，即：<产品线>/<模块>/<文件>

```
namespace im {
namespace bs {
class FooClass {
public:
    void function(){
        // ...
    }
};
}  // namespace bs
}  // namespace im
```

. 在<src_root>/im/bs/目录下。

文件布局

. 文件注释：版权、作者、文件内容说明：

```
// Copyright (c) 2011 Baidu.com, Inc. All Rights Reserved
// @author xiaoming(xiaoming@baidu.com)
//         xiaowang(xuaiwabg@baidu.com)
// @brief 用中文说明本文件的职责和作用
```

. 每个文件都有文件注释。

. 有多个作者时，作者名缩进对齐。

. 类注释：

. 每个类都有类注释。

. 说明类的职责，而非实现方法和细节。

. 函数注释：

. 说明函数的职责，而非实现的方法和细节。

. 如果有参数，说明参数的含义。

. 如果在类的上下文中函数的职责明显且无歧义，可以省略函数注释。

原则：注释的位置

声明注释（文件注释、类注释、函数注释）推荐放在.h 文件中，当且仅当没有.h 文件时才允许放在.cpp 文件中。实现注释（函数体内算法、逻辑说明注释）放在.cpp 文件中。

一份注释不允许重复出现在多个位置或多个文件中。

. 头文件：

. 包含保护（#define guard）：

```
#ifndef IM_BS_FOO_H
#define IM_BS_FOO_H
...
#endif  // IM_BS_FOO_H
```

. 头文件必须声明包含保护。
. 格式为 <产品线>_<模块>_<文件>_H

. 头文件依赖：
. 能使用前置声明时，尽量不使用 #include
. 内联函数（inline function）：
. 当且仅当函数长度在 10 行以内时，才允许定义为内联函数。
. 函数参数声明：
. 不建议使用 output parameter。
. 如果使用了 output parameter，则参数顺序为先 inputs 后 outputs。
. 包含（includes）的顺序和组织：

```
#include <stdio.h>

#include <iostream>

#include "third_party_lib.h"

#include "im/bs/foo.h"
```

. 顺序：C 标准库，C++标准库，第三方类库，其他产品或模块头文件，本项目头文件。
. 各组 include 之间用一行空行间隔。

语法

定义：语法规则等级

语法规则分为“必须”、“建议”、“允许”、“不建议”和“禁止”五个等级。凡未被“禁止”者理论上皆可使用；凡标为“不建议”者使用时需慎重考虑。

. 关键字 using 的使用：
. 头文件中禁止使用 using
. 源文件中禁止使用 using <namespace>，允许使用 using <Class>
. 单元测试文件中允许使用 using <namespace>

. 嵌入类（nested class）：
. 除非嵌入类是接口的一部分，否则禁止将其声明为 public

. 全局函数：
. 不建议使用全局函数。
. 建议使用命名空间内的非成员函数或静态函数。

. 全局和静态变量：
  . 禁止使用 Class 类型的全局或静态变量。
  . 允许使用 Class 类型的全局或静态指针。

. 局部变量（local variable）声明：

```
232 int index = initialize_index();
```

  . 建议在尽可能小的作用域内声明局部变量，即：局部变量声明尽可能接近其使用。
  . 必须在声明时进行初始化。

. 构造函数：
  . 职责：建议仅负责成员变量初始值的设置，任何复杂的初始化逻辑需放在 init()函数中。
  . 缺省构造函数：如果类包含成员变量且没有其他构造函数，则必须定义缺省构造函数。
  . 单参数构造函数：单参数构造函数必须使用 explicit 关键字修饰。
  . 拷贝构造函数（copy constructor）：
    . 除非确有必要，否则禁止定义拷贝构造函数和赋值运算符（assignment operator）。

```
234 #define DISALLOW_COPY_AND_ASSIGN(TypeName) \
235     TypeName(const TypeName&);             \
236     void operator=(const TypeName&)
237
238 class Foo {
239 public:
240     Foo(int f);
241     ~Foo();
242
243 private:
244     DISALLOW_COPY_AND_ASSIGN(Foo);
245 };
```

    . 不定义时用 DISALLOW_COPY_AND_ASSIGN 宏将其禁止。

. 结构体（struct）
  . 创建自定义类型时，建议使用 Class。
  . 仅当自定义类型只包含数据时，才允许使用结构体。

. 继承
  . 建议优先考虑使用组合（composition）。
  . 确实需要继承时（继承类 is-a 被继承类），必须定义为 public 继承。
  . 禁止使用 private 继承。

. 多重继承
  . 不建议使用多重继承。
  . 确实需要多继承时，必须满足：最多只有一个非抽象基类，其他基类都是纯接口类。

. 运算符重载
  . 除非确实必要，否则禁止重载操作符，例如：
    . 允许 operator << (ostream&, const T&))
    . 允许对象在 STL 容器中作键值时，重载 operator==或 operator<

. 成员变量和存取函数（reader/writter）

. 所有成员变量必须声明为 private
. 如果定义存取函数，必须遵守命名约定：m_foo 的取值函数为 foo()，赋值函数为 set_foo()

. 函数职责和长度
. 建议每个函数有且只有唯一职责。
. 每个函数应尽可能简短，长度建议不超过 40 行。

. 引用参数（reference argument）
. 按引用传递的参数必须用 const 修饰。

. 缺省参数
. 禁止定义函数缺省参数。

. 变长数组和 alloca()
. 禁止使用变长数组和 alloca()

. 友元
. 允许友元函数和友元类。
. 禁止友元成员变量。
. 建议将友元定义在同一个文件内。

. 异常（Exception）
. 禁止使用异常机制。

. 运行时类型识别（RTTI）
. 除单元测试外，禁止使用运行时类型识别。

. 类型转换
. 允许使用 C++的类型转换，如 static_cast<>()
. 禁止使用 C 的类型转换，如 int y = (int)x 或 int y = int(x)

. 自增和自减定义
. 对迭代器（iterator）和模板类型，建议使用前置自增或自减，即：++index 或—index

. const 的使用
. 建议在任何可能的情况下都使用 const

. 整型（integer）的使用
. 内建（built-in）类型中，必须仅使用 int
. 如果需要不同大小的变量，允许使用<stdint.h>中长度精确的整形，例如 int16_t

. 预处理宏
. 不建议使用宏，建议用内联函数、枚举和常量代替之。

. 0 和 NULL 的表示
  . 整数必须使用 0
  . 浮点必须使用 0.0
  . 指针必须使用 NULL
  . 字符（串）必须使用'\0'

. sizeof 的使用
  . 建议使用 sizeof(<变量>)，不建议使用 sizeof(<类型>)

. 流（stream）
  . 禁止使用流。

. 对 64 位友好
  . 代码必须是 64 位友好的。
  . 对于确定是 32 或 64 位的需求，建议使用 int64_t，int32_t 之类的表达。

. 第三方类库
  . 允许使用 STL 类库
  . 其他常用类库（例如 Boost），允许产品线内部在统一版本的情况下使用。

遗留代码

1. 遗留代码中存在和本规范不一致部分，在不影响当前工作情况下，可以不修改。
2. 当前工作中涉及到的遗留代码，建议逐步修改到和本规范一致。