[研发部 C++编码风格指南](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658)

[背景](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E8%83%8C%E6%99%AF)

C++ 语言是公司项目使用的主力开发语言之一。每个C++ 程序员都能感受到该语言的强大能力。但这些能力也带来了复杂性，反过来使代码更容易出错，更难以阅读和维护。

本规范的目标是通过详细描述编写代码的注意事项来管理这种复杂性。存在这些规则是为了使代码库易于管理，同时仍允许编码人员高效地使用语言功能。

代码风格，也称为可读性，就是我们称之为C++ 编码规范的约定。

请注意，本指南不是C++ 语言教程：我们假设读者熟悉该语言。

[C++ 语言版本](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#c-%E8%AF%AD%E8%A8%80%E7%89%88%E6%9C%AC)

默认情况下，项目交付代码使用**C**++03标准，以获得更广泛的编译环境兼容性。**C**++03是**C**++98标准的修订版，两者完全兼容。

项目交付之外的C++ 代码，如内部使用的开发、调试、部署工具，最高使用**C**++17标准。勿使用**C**++2x特性。

在项目中使用非标准扩展前，优先考虑多平台的可移植性。

[头文件](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A4%B4%E6%96%87%E4%BB%B6)

通常，每个.cpp文件都应该有一个关联的.h文件。有一些常见的例外，例如单元测试和.cpp仅包含**main**()函数的小文件 。

正确使用头文件会对代码的可读性，大小和性能产生巨大影响。

以下规则将指导您使用头文件的各种麻烦。

[**头文件保护**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A4%B4%E6%96%87%E4%BB%B6%E4%BF%9D%E6%8A%A4)

所有头文件都**必须**有多重包含保护。有#**ifndef** #**define** #**endif**及#**pragma** once两种方式，选用第一种。

第一种方式中符号名称的格式应为

\_<**PROJECT**>\_<**PATH**>\_<**FILE**>\_H\_

为了保证唯一性，它们应该基于项目源树中的完整路径。例如，foo/src/bar/baz.h项目中的文件foo应具有以下保护：

#**ifndef** \_FOO\_BAR\_BAZ\_H\_

#**define** \_FOO\_BAR\_BAZ\_H\_

// ...

#**endif** // \_FOO\_BAR\_BAZ\_H\_

[**头文件依赖**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A4%B4%E6%96%87%E4%BB%B6%E4%BE%9D%E8%B5%96)

头文件中不使用前置声明，直接#**include**所需的其他头文件。

**注意：** 只包含且必须包含必要的头文件，避免包含不需要的头文件。

**例外：** SDK接口的**Type**\*句柄类型，可使用头文件中前置声明的方式，隐藏实现细节。

[**函数参数顺序**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E5%8F%82%E6%95%B0%E9%A1%BA%E5%BA%8F)

定义参数时，参数顺序为：输入参数在前，输出参数在后。 输入参数为传值或常数引用；输出参数为非常数指针，并添加小写字母o作为前缀。

本条应尽量遵守，类或结构体中的输入输出两用参数作为中间参数。

[**函数多返回值**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E5%A4%9A%E8%BF%94%E5%9B%9E%E5%80%BC)

函数需要有多个返回值时，一般只在函数返回值中直接返回执行结果或错误码，其他数据通过输出参数返回。

如果返回值的拷贝代价较小，所有返回值在32字节以内（包含），也允许将返回值全部定义在结构体中，函数返回此结构体实例。如：

**struct** **GetResult**

{

**int** errorCode;

**int** dataSize;

// data中包含很多数据，但浅拷贝只涉及指针的拷贝

**void** \*data;

};

GetResult **Get**(**int** param)

{

GetResult getResult;

getResult.errorCode = 0;

getResult.dataSize = GetDataSize(param);

getResult.data = GetData(param);

/\* C++11标准以前没有返回值优化，此处会引发拷贝构造

\* 默认的拷贝构造函数会浅拷贝每个数据成员

\*/

**return** getResult;

}

如果项目允许使用C++11及以上标准：由于编译器有返回值优化，以上方式为建议多返回值方式；也允许使用std::tuple元组同时返回多个结果。

[**包含头文件的名词及次序**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%8C%85%E5%90%AB%E5%A4%B4%E6%96%87%E4%BB%B6%E7%9A%84%E5%90%8D%E8%AF%8D%E5%8F%8A%E6%AC%A1%E5%BA%8F)

包含次序标准化可增强可读性，避免依赖。次序为：C库、C++库、其他库的.h、项目内的.h。

即：

C系统文件

C++系统文件

其他库头文件

本项目内头文件

项目内头文件按照项目源码的目录树结构排序，避免使用UNIX路径.和..

**总结：**

(1) 避免多重包含

(2) 标准化函数参数顺序可提高可读性和易维护性

(3) 包含文件使用.和..虽然方便，却易混乱，使用完整项目路径更清晰、美观。

[作用域](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BD%9C%E7%94%A8%E5%9F%9F)

[**命名空间**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%91%BD%E5%90%8D%E7%A9%BA%E9%97%B4)

在.cpp文件中，提倡使用不具名命名空间。如果使用具名命名空间，基于项目或路径名称而不是**using**指示符。命名空间内不用缩进。

**定义：** 命名空间将全局作用域细分为不同的具名作用局，可有效防止全局作用局的命名冲突。

**优点：** 命名空间的命名轴线可嵌套。父级命名不同时即使当前命名重复也不会冲突。

**缺点：** 不具名命名空间容易违背C++中唯一定义的原则。

**结论：** 合理使用

[**嵌套类**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%B5%8C%E5%A5%97%E7%B1%BB)

不要将嵌套类定义为**public**，除非是接口的一部分。

[**非成员函数、静态成员函数、全局函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%9D%9E%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0%E9%9D%99%E6%80%81%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0%E5%85%A8%E5%B1%80%E5%87%BD%E6%95%B0)

使用命名空间内的非成员函数和静态成员函数，尽量不使用全局函数。

[**局部变量**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F)

将函数变量尽可能至于最小作用域内，在声明变量时将其初始化。

C++允许在函数的任何位置声明变量，提倡在尽可能小的作用域中声明变量，离第一次使用越近越好。这样代码易于阅读且易于定位变量的声明位置、变量类型和初始值。

*特别是，应使用初始化代替声明再赋值的方式。*

[**全局变量**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F)

禁止**class**类型的全局变量，多线程代码中禁止非常数全局变量，禁止使用函数返回值初始化全局变量。

**class**类型的全局变量内部构造、析构等初始化操作每次生成有可能变化，从而导致难以发现的缺陷。因此，禁止使用**class**类型的全局变量（包括std::string、std::vector等）。

全局的字符串常量，使用C风格的字符串，不要使用std::string。

如果必须使用**class**或标准库容器类型的全局变量，则应定义为**class**的原始指针类型，并做Lazy Init处理。

**注意：** 静态成员变量视为全局变量，也不能是**class**类型。

**总结：**

(1) .cpp中的不具名命名空间可避免命名冲突且限定作用域，避免使用**using**提示符污染命名空间；

(2) 嵌套类符合局部使用原则，可惜不能再其他头文件前置声明，尽量不要**public**；

(3) 尽量不用全局函数和全局变量，考虑作用域和命名空间限制，尽量单独形成编译单元；

(4) 多线程中的全局变量不要使用**class**类型（含标准库容器），避免编译器未定义行为（UB）引发的缺陷；

作用域的使用，除了考虑名称污染、可读性外，主要是为了降低耦合度，提高编译、执行效率。

[C++类](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#c%E7%B1%BB)

类是C++中基本的代码单元，被广泛使用。

[**构造函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)

构造函数只初始化没有实际意义的成员变量。需要初始化的数据在Init()函数中集中初始化。

**定义：** 在构造函数中执行初始化操作

**优点：** 排版方便，无需担心类是否初始化

**缺点：** 在构造函数中执行初始化操作容易引起以下问题：

(1) 构造函数中不易报告错误，不能使用异常

(2) 操作失败会造成对象初始化失败，引起不确定状态

(3) 构造函数内调用虚函数，调用不会派发到子类实现中，即使当前不需要子类实现，但依然是隐患

(4) 如果有人创建该类的全局变量（这样做违背禁止类作为全局变量的准则），构造函数在**main**()函数执行前调用，有可能影响代码中使用flag的逻辑。

**结论：**

如果对象需要有意义的初始化，使用Init()函数合并需要初始化的内容，并增加一个成员变量标记该对象是否初始化成功。

[**默认构造函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%BB%98%E8%AE%A4%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)

如果一个类定义了若干个成员变量又没有其他构造函数，需要定义一个默认构造函数，否则编译器将自动生成默认构造函数。

**定义：** 新建一个没有参数的对象时，默认构造函数被调用，当使用**new**时，默认构造函数会被调用。

**优点：** 默认将结构体初始化为"不可能的"值，使调试更加容易

**缺点：** 对代码编写者来说，这是多余的工作。

**结论：** 如果类中定义了成员变量，没有提供其他构造函数，则需要自定义一个默认构造函数（没有参数）。默认构造函数用于初始化对象，使对象内部状态一致、有效。

提供默认构造函数的原因：编译器自动生成的构造函数不会对对象进行初始化。

如果定义的类继承自现有类，且没有新增新的成员变量，则不需要为新类定义默认构造函数。

[**明确的构造函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%98%8E%E7%A1%AE%E7%9A%84%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)

单参数构造函数使用C++关键字**explicit**

**定义：** 通常，只有一个参数的构造函数会发生隐式转换。避免此问题，可声明为**explicit**

**优点：** 避免不合时宜的变换。

**缺点：** 无

**结论：**

所有单参数构造函数必须是明确的，在类定义中加**explicit**

拷贝构造函数不需特意声明为**explicit**

[**拷贝构造函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%8B%B7%E8%B4%9D%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)

仅在代码中需要拷贝一个类对象的时候使用拷贝构造函数；不需要拷贝时使用

**定义：** 通过拷贝新建对象时可使用拷贝构造函数（特别是对象的传值时）。

**优点：** 拷贝构造函数使得拷贝对象更加容易，标准库容器要求所有内容可拷贝、可赋值。

**缺点：** C++中对象的隐式拷贝是导致很多性能问题和缺陷的根源。拷贝构造函数降低了代码的可读性，相比按引用传递，跟踪按值传递的对象更加困难，对象修改的地方变的难以捉摸。

**结论：** 禁止使用拷贝构造函数，容器中的元素使用类的指针指针类型。

[**结构体和类**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%BB%93%E6%9E%84%E4%BD%93%E5%92%8C%E7%B1%BB)

仅当只有数据时使用**struct**，其他一概使用**class**。

**注意：** 类和结构体的成员变量使用不同的命名规则

[**接口（Interface）**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%8E%A5%E5%8F%A3interface)

接口是指满足特性条件的类，这些类以Interface为后缀（命名约定）。

**定义：** 当一个类满足以下要求时，称之为纯接口：

(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，或定义不含参数的protected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

接口类不能被直接实例化，因为他声明了纯虚函数。为确保接口类所有实现可被正确销毁，必须为之申明虚析构函数。

**优点：** 代码维护人员看到Interface就知道不能为该接口类实现函数或非静态数据成员

**缺点：** 增加类名长度带来阅读不便。同时，接口细节不应暴露给用户。

**结论：** 满足上述需求的类才使用Interface结尾，但满足也可以不使用。

[**继承**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%BB%A7%E6%89%BF)

尽量使用组合而不是继承。如果使用继承，只用**public**继承。

**定义：** 接口继承：继承自纯接口的抽象类。实现继承：继承自一个非纯接口类。

允许接口继承，尽量不使用实现继承。

[**多重继承**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A4%9A%E9%87%8D%E7%BB%A7%E6%89%BF)

多重继承的使用场景：仅允许最多一个基类中包含实现，其他基类都是以Interface为后缀的纯接口。

[**操作符重载**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%93%8D%E4%BD%9C%E7%AC%A6%E9%87%8D%E8%BD%BD)

除少数特定环境外（例外），不要重载操作符。

**定义：** 一个类合一定义诸如+、/等操作符，使其可以像内建类型一样直接使用。

**优点：** 提高代码可读性，比如Equals()用==代替

**缺点：**

(1) 混淆直觉，让开发者误以为一些耗时的操作像内建操作那样轻巧；

(2) 查找重载操作符的调用处更加困难，查找Equals()比==调用容易的多。

(3) 有的操作符可以操作指针，容易导致缺陷

(4) 副作用，如重载操作符&不能被前置声明

**结论：**

禁止重载操作符，尤其是赋值操作符严格禁止。

**例外：**

类型需要作为Key放入std::map，或类型需要在容器中排序。则应当遵从标准库要求，提供相应的操作符重载。

重载原则：单目操作符（如**operator**!）定义为类成员函数；双目操作符（如**operator**<, operator==）定义为全局函数，并声明为类友元。

[**getter/setter**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#gettersetter)

将数据成员私有化，并提供相关的getter/setter

[**声明次序**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A3%B0%E6%98%8E%E6%AC%A1%E5%BA%8F)

类中声明次序：**public**、**protected**、**private**，如果某一块没有，直接忽略。

每一块中，声明次序如下：

(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；

.cpp文件中的函数定义应尽可能声明次序一致。

禁止将大型函数内联到类的定义中。

[**编写短小函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%BC%96%E5%86%99%E7%9F%AD%E5%B0%8F%E5%87%BD%E6%95%B0)

函数长度尽量不超过40行，倾向选择短小、凝练的函数。

函数尽量短小、简单，便于他人阅读和修改代码。处理代码时如发现长函数，考虑分割为若干短小、易于管理的若干函数。

**总结：**

(1) 不在构造函数中做太多逻辑相关的初始化；

(2) 编译器默认构造函数不会初始化变量，如果定义了其他构造函数，编译器不再提供，需要编码这自行提供默认构造函数；

(3) 为避免隐式转换，需将单参数构造函数声明为**explicit**；

(4) 为避免拷贝构造函数、赋值操作的滥用和编译器自动生成，可声明为**private**且无需实现；

(5) 仅在作为数据集合时使用**struct**；

(6) 组合>实现继承>接口继承>私有继承，子类重载的虚函数也要声明**virtual**关键字；

(7) 避免使用多重继承，唯一的使用场景是最多一个基类含实现，其他基类纯接口；

(8) 接口类类名以Interface为后缀，全部使用纯虚函数；

(9) 禁止重载操作符

(10) 存取函数一般内联在头文件中；

(11) 声明次序: public -> protected -> private；

(12) 函数体尽量短小、紧凑，功能单一。

[其他C++特性](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%85%B6%E4%BB%96c%E7%89%B9%E6%80%A7)

[**智能指针**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%99%BA%E8%83%BD%E6%8C%87%E9%92%88)

禁止使用std::auto\_ptr。

[**const**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#const)

引用形参用const，否则使用指针形参

[**函数重载的使用要清晰、易读**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E9%87%8D%E8%BD%BD%E7%9A%84%E4%BD%BF%E7%94%A8%E8%A6%81%E6%B8%85%E6%99%B0%E6%98%93%E8%AF%BB)

[**禁止使用缺省函数参数（测试框架例外）**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%A6%81%E6%AD%A2%E4%BD%BF%E7%94%A8%E7%BC%BA%E7%9C%81%E5%87%BD%E6%95%B0%E5%8F%82%E6%95%B0%E6%B5%8B%E8%AF%95%E6%A1%86%E6%9E%B6%E4%BE%8B%E5%A4%96)

[**禁止使用变长数组**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%A6%81%E6%AD%A2%E4%BD%BF%E7%94%A8%E5%8F%98%E9%95%BF%E6%95%B0%E7%BB%84)

用std::vector代替

[**合理使用友元**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%90%88%E7%90%86%E4%BD%BF%E7%94%A8%E5%8F%8B%E5%85%83)

[**限制使用C++异常，测试框架例外**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%99%90%E5%88%B6%E4%BD%BF%E7%94%A8c%E5%BC%82%E5%B8%B8%E6%B5%8B%E8%AF%95%E6%A1%86%E6%9E%B6%E4%BE%8B%E5%A4%96)

底层库允许使用异常，并将异常限定在模块内部，不要抛出到外部。

构造、析构函数中不要抛出异常。

并尽力避免在构造、析构函数中执行会引发异常的操作。如难以避免，也应捕获并处理所有异常，而不能抛出。

[**禁止使用RTTI**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%A6%81%E6%AD%A2%E4%BD%BF%E7%94%A8rtti)

[**类型转换**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%B1%BB%E5%9E%8B%E8%BD%AC%E6%8D%A2)

使用**static\_cast**<Type>(var)等C++的类型转换，禁止使用(**Type**)**var**强制类型转换。

[**禁止使用流**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%A6%81%E6%AD%A2%E4%BD%BF%E7%94%A8%E6%B5%81)

除日志、序列化需要，其他场景禁止使用std::stream，使用printf代替。

[**const的使用**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#const%E7%9A%84%E4%BD%BF%E7%94%A8)

任何可以使用的场景都要使用**const**，且**const**在前。

[**使用确定大小的整型**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BD%BF%E7%94%A8%E7%A1%AE%E5%AE%9A%E5%A4%A7%E5%B0%8F%E7%9A%84%E6%95%B4%E5%9E%8B)

使用确定大小的整型，除表示数组位数外的场景禁止使用无符号型

[**可移植性**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%8F%AF%E7%A7%BB%E6%A4%8D%E6%80%A7)

格式化输出及结构对齐时，注意32位和64位系统差异

[**预处理宏**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%A2%84%E5%A4%84%E7%90%86%E5%AE%8F)

除字符串化和连接的场景外，一律禁止使用宏。要求使用内联函数、枚举和常量代替宏。

[**0和NULL**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#0%E5%92%8Cnull)

整数用0，实数用0.0，指针用nullptr，字符（串）用'\0'

[**sizeof**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#sizeof)

用sizeof(**variableName**)代替sizeof(**TypeName**)

[**Boost库**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#boost%E5%BA%93)

暂不使用Boost

[命名约定](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%91%BD%E5%90%8D%E7%BA%A6%E5%AE%9A)

一致性命名规则，包括类型、变量、函数、常量、宏等等。

[**通用命名规则**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%80%9A%E7%94%A8%E5%91%BD%E5%90%8D%E8%A7%84%E5%88%99)

函数、变量、文件命名应具有描述性，不要过分缩写。

类型和变量应该是名词，函数名用"命令性质"的动词。

命名方法：

尽可能给出描述性名词，提高代码可读性，目标是让别人快速理解。

禁止使用模糊的缩写或随意的字符。

类型和变量名一般为名词，如numErrors。禁止使用泛称，尽量用完整的单词描述清楚作用和意图，不要怕字多。例如**struct** **ProcData**就不是好的命名。

函数名通常是指令性的，如OpenFile()、SetNumErrors()，访问函数需要描述的更细致，要与其访问的变量相吻合。

缩写：

除非是放在项目外也非常明了的缩写，如IOT、DNS、IP等

除linux命令常用缩写（如 ls、pwd、usr等）外，不要用省略字母的缩写。如count不要写成cnt

[**文件命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%96%87%E4%BB%B6%E5%91%BD%E5%90%8D)

文件名同类名一致，包括大小写。已有命名约定要求的项目遵循项目要求。

不要使用已经存在于系统中的文件名，如Linux系统不要使用/usr/**include**下的文件名，如**db**.h。

[**类型命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%B1%BB%E5%9E%8B%E5%91%BD%E5%90%8D)

类型命名每个单词以大写字母开头，不包含下划线，如：MyExcitingClass、MyExcitingEnum。

所有类型命名：类、结构体、类型定义（typedef）、枚举，使用相同约定，例如：

// class and structs

**class** **UrlTable** {...};

**class** **UrlTableTester** {...};

**struct** **UrlTableProperties** {...};

// typedefs

**typedef** hash\_map<UrlTableProperties \*, string> PropertiesMap;

// enums

**enum** UrlTableErrors{...};

[**变量命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%8F%98%E9%87%8F%E5%91%BD%E5%90%8D)

变量名使用驼峰命名法，第一个单词以小写开头，其他单词首字母大小，单词之间不使用分隔符。类的私有成员添加下划线结尾。示例如下：

**class** **SampleClass**

{

**public**:

// 函数参数使用驼峰命名，无特殊前缀或后缀

**void** **SampleFunction**(**int** parameterVariable)

{

// 临时变量一般使用驼峰命名，无特殊前缀或后缀

**int** localVariable = 0;

// 无特殊含义的临时变量，可以使用全小写，同重要变量有所区分

**for** (**int** i = 0, ncount = 50; i < ncount; ++i)

{

**int** retval = OtherFunctionCall();

**if** (retval != 0)

{

**int** errcode = write(...);

}

}

}

**private**:

// 类数据成员，使用下划线结尾

**int** dataMember\_;

};

**struct** **Message**

{

// 结构体成员默认为public，不使用下划线结尾

**int** index;

string msg;

};

[**全局变量命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F%E5%91%BD%E5%90%8D)

不建议使用全局变量。如果一定要用，以g\_为前缀以便与局部变量区分。

[**静态变量命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%9D%99%E6%80%81%E5%8F%98%E9%87%8F%E5%91%BD%E5%90%8D)

不建议使用静态变量。如果一定要用，以s\_为前缀以便与非静态变量区分。静态全局变量仅需添加s\_前缀。

[**常量命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%B8%B8%E9%87%8F%E5%91%BD%E5%90%8D)

常量命名全部大写，单词间以下划线分隔，如：

**const** **int** DAYS\_IN\_A\_WEEK = 7;

[**函数命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E5%91%BD%E5%90%8D)

普通函数，函数名大写字母开头，每个单词首字母大写，没有下划线：

AddTableEntry()

getter/setter函数，函数名要与get/set的变量名匹配。

**class** **MyClass**

{

**public**:

// ...

**int** **GetNumEntries**() **const** { **return** numEntries\_; }

**void** **SetNumEntries**(**int** numEntries)

{

numEntries\_ = numEntries;

}

**private**:

**int** numEntries\_;

};

[**名空间命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%90%8D%E7%A9%BA%E9%97%B4%E5%91%BD%E5%90%8D)

全部小写，命名基于项目名称和目录结构

**namespace** pengze

{

}

[**枚举命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%9E%9A%E4%B8%BE%E5%91%BD%E5%90%8D)

枚举值全部大写，单词之间用下划线分隔 MY\_EXCITING\_ENUM\_VALUE。

枚举名称属于类型，因此大小写混合。

**enum** UrlTableErrors {

OK = 0,

ERROR\_OUT\_OF\_MEMORY = 1,

ERROR\_MALFORMED\_INPUT = 2,

};

[**宏命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%AE%8F%E5%91%BD%E5%90%8D)

尽量使用枚举、常量、内联替代宏。

如果必须要用，宏的名称全部大写，单词之间用下划线分隔 MY\_MACRO\_THAT\_SCARES\_SMALL\_CHILDREN\_AND\_ADULTS\_ALIKE

[**命名规则例外**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%91%BD%E5%90%8D%E8%A7%84%E5%88%99%E4%BE%8B%E5%A4%96)

参考现有或相近命名约定

[代码注释](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BB%A3%E7%A0%81%E6%B3%A8%E9%87%8A)

[**注释风格**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%B3%A8%E9%87%8A%E9%A3%8E%E6%A0%BC)

说明代码功能的关键注释统一使用/\* \*/。

面向外部的头文件中，统一使用Doxygen注释风格。

[**文件注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%96%87%E4%BB%B6%E6%B3%A8%E9%87%8A)

文件注释需说明当前文件在项目中的核心作用。并使用Doxygen注释风格：

/\*\*

\* @file Sample.h

\* @author Author Name (username@ghostcloud.cn)

\* @brief Doxygen 注释风格示例

\* @version 0.1

\* @date 2021-03-11

\*

\* @copyright Copyright (c) 2021

\*

\*/

#**ifndef** \_GHOSTCLOUD\_PENGZE\_SAMPLE\_H\_

#**define** \_GHOSTCLOUD\_PENGZE\_SAMPLE\_H\_

#**endif** /\* !\_GHOSTCLOUD\_PENGZE\_SAMPLE\_H\_ \*/

[**类注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%B1%BB%E6%B3%A8%E9%87%8A)

类的头文件中，对类的成员函数、成员变量做说明。

[**函数注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E6%B3%A8%E9%87%8A)

函数申明处注释的内容：

(1) 函数作用：当前函数需要达成的目的

(2) 函数参数：说明是输入还是输出，及参数含义

(3) 返回值：含义和类型

面向外部的接口函数注释使用Doxygen风格：

/\*\*

\* @brief 创建一个示例对象

\*

\* @details 详细说明（可选）

\*

\* @note 注意事项说明（可选）

\*

\* @param arg0 参数0，参数0说明

\* @param arg1 参数1，参数1说明

\* @param arg2 参数2，参数2说明

\*

\* @return int ErrorCode

\* @retval 0 成功

\* @retval -1 失败

\*

\*/

**int** **CreateSample**(**int** arg0, **int** arg1, **int** arg2);

[**变量注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%8F%98%E9%87%8F%E6%B3%A8%E9%87%8A)

变量在上一行使用 // 注释

// a variable

**int** a;

[**实现注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%AE%9E%E7%8E%B0%E6%B3%A8%E9%87%8A)

复杂或重要的逻辑实现，需要说明实现的意图、目的、主要思想。

**注意：** 注释是对逻辑的解释，而非代码的翻译。要假定代码阅读者比你更熟悉C++语言本身，但他不明白为什么要这样实现。

[**标点、拼写和语法**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%A0%87%E7%82%B9%E6%8B%BC%E5%86%99%E5%92%8C%E8%AF%AD%E6%B3%95)

注释要易读

[**TODO**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#todo)

临时、短期的解决方案，需要使用TODO注释。临时注释使用 //

TODO大写，需要备注作者名，备查找： // TODO(Luffy): 需要优化逻辑

[代码格式](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BB%A3%E7%A0%81%E6%A0%BC%E5%BC%8F)

团队的C++开发规范约定，1个Tab=4个空格

[**行长度**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E8%A1%8C%E9%95%BF%E5%BA%A6)

每一行字符数不超过80

这一条存在例外情况：

(1) 如果一行注释包含了超过80字符的命令或URL，出于复制粘贴方便可以超过80字符；

(2) 包含长路径的可以超过80列，尽量避免

(3) 头文件保护可以不遵从该原则

[**括号空格与换行**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%8B%AC%E5%8F%B7%E7%A9%BA%E6%A0%BC%E4%B8%8E%E6%8D%A2%E8%A1%8C)

**void** **MyFunctionName**()

{

// 操作符之间空格隔开

**int** a = 50 \* 2 + 1;

// 关键字与括号之间留一个空格

**if** (a == 50)

{

}

**else** **if** (a > 60)

{

}

**else**

{

}

**while** (a == 70)

{

}

// do while的while与花括号在同一行

**do**

{

} **while** (a < 90);

// for循环分号后面加空格

**for** (**int** i = 0; i < 70; ++i)

{

}

}

[**非ASCII字符**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%9D%9Eascii%E5%AD%97%E7%AC%A6)

尽量不使用ASCII字符，使用时必须使用UTF-8格式

[**缩进空格 or 制表符Tab**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%BC%A9%E8%BF%9B%E7%A9%BA%E6%A0%BC-or-%E5%88%B6%E8%A1%A8%E7%AC%A6tab)

使用空格缩进，每级缩进4个空格。

[**函数声明与定义**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E5%A3%B0%E6%98%8E%E4%B8%8E%E5%AE%9A%E4%B9%89)

返回类型和函数名在同一行，参数也在同一行。

如果一行文本较多写不下所有参数，则参数换行空格对齐：

ReturnType LongClassName::ReallyReallyReallyLongFunctionName(

Type parName1,

Type parName2,

Type parName3)

{

DoSomething(); // 2 space indent

...

}

**注意：**

(1) 返回值和函数在同一行

(2) 左圆括号(总是和函数名在同一行

(3) 函数名和左圆括号之间没有空格

(4) 圆括号与参数没有空格

(5) 函数的左花括号另起一行，单独置于一行

(6) 函数的右花括号在函数最后，单独置于一行

(7) 右圆括号和左花括号间换行

(8) 函数声明和实现处的所有形参名称必须保持一致

(9) 所有形参尽可能对齐

(10) 默认一级缩进为4个空格

(11) 独立封装的参数保持4个空格缩进

const函数，关键字**const**和最后一个参数置于同一行

ReturnType LongClassName::ReallyReallyReallyLongFunctionName(

Type parName1,

Type parName2,

Type parName3) **const**

{

DoSomething(); // 2 space indent

...

}

暂不使用的参数需在函数的注释中说明

[**函数调用**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E8%B0%83%E7%94%A8)

尽量放在同一行，否则将括号内的参数换行对齐

[**条件语句**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%9D%A1%E4%BB%B6%E8%AF%AD%E5%8F%A5)

条件语句的左花括号放在条件语句的行位，格式上与函数区分。

注意：条件内哪怕只有一行语句，也要写全花括号，方便后续维护。

有些条件语句写在一行可以增强可读性，仅当语句简单且没有**else**子句时可以使用：

**if** (x == kFoo) **return** **new** Foo();

**if** (x == kBar) **return** **new** Bar();

注意：如果有else子句，则禁止不带花括号。

[**循环和开关选择语句（loop and switch）**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%BE%AA%E7%8E%AF%E5%92%8C%E5%BC%80%E5%85%B3%E9%80%89%E6%8B%A9%E8%AF%AD%E5%8F%A5loop-and-switch)

switch 语句尽量使用大括号分块

switch 语句必须要有default存在，如正常不会执行则以异常日志或控制台信息的方式处理。

空循环使用{}或者continue，禁止循环只用一个分号

[**指针和引用表达式**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%8C%87%E9%92%88%E5%92%8C%E5%BC%95%E7%94%A8%E8%A1%A8%E8%BE%BE%E5%BC%8F)

句号.、箭头->前后不能有空格。

\*、&作为解引用、取值操作符使用时，要紧贴其变量；作为指针和引用声明时，建议\*紧贴变量名，&紧贴类型名。

value = \*test;

value = &test;

value = c.test;

value = c->test;

**char** \*cinput;

**const** string& strName;

[**布尔表达式**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%B8%83%E5%B0%94%E8%A1%A8%E8%BE%BE%E5%BC%8F)

如果布尔表达式超过标准行宽（80字符），则断行且要求逻辑操作符置于行尾。

**if** (thisOneThing > thisOtherThing &&

aThirdThing == aFourthThing &&

yetAnother & lastOne)

{

...

}

有多个逻辑操作符时，考虑使用圆括号()提示运算优先级

[**返回值**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E8%BF%94%E5%9B%9E%E5%80%BC)

函数返回（即：**return** 表达式）时不要使用圆括号。

[**初始化操作**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%88%9D%E5%A7%8B%E5%8C%96%E6%93%8D%E4%BD%9C)

使用等号=完成

[**预处理**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%A2%84%E5%A4%84%E7%90%86)

不要缩进，从行首开始。及时在缩进代码块中，也要从行首开始。

[**类格式**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%B1%BB%E6%A0%BC%E5%BC%8F)

类格式参考本规范声明次序章节。

[**初始化列表**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%88%9D%E5%A7%8B%E5%8C%96%E5%88%97%E8%A1%A8)

初始化列表置于同一行，或换行空格补齐

[**命名空间格式化**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%91%BD%E5%90%8D%E7%A9%BA%E9%97%B4%E6%A0%BC%E5%BC%8F%E5%8C%96)

命名空间内容不缩进。

**namespace** pengze

{

**class** **MyClass**

{

};

}

[**水平空白**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%B0%B4%E5%B9%B3%E7%A9%BA%E7%99%BD)

行尾不要增加无意义的空白。当行尾有注释需要空格补齐时可以适当留空。

[**垂直空白**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%9E%82%E7%9B%B4%E7%A9%BA%E7%99%BD)

垂直空白尽量少。函数之间、模块之间留一样空白

[**规则之例外**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E8%A7%84%E5%88%99%E4%B9%8B%E4%BE%8B%E5%A4%96)

前文提到的编码习惯基本是强制性的，但优秀的规则都允许例外。

[**Windows代码**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#windows%E4%BB%A3%E7%A0%81)

在Windows平台开发，除系统API不能更改外，自主开发内容需遵循本规则。

[**团队合作**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%9B%A2%E9%98%9F%E5%90%88%E4%BD%9C)

团队风格一致性，代码高可读性。

[C++编码风格指南](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#c%E7%BC%96%E7%A0%81%E9%A3%8E%E6%A0%BC%E6%8C%87%E5%8D%97)

[背景](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E8%83%8C%E6%99%AF)

C++ 语言是公司项目使用的主力开发语言之一。每个C++ 程序员都能感受到该语言的强大能力。但这些能力也带来了复杂性，反过来使代码更容易出错，更难以阅读和维护。

本规范的目标是通过详细描述编写代码的注意事项来管理这种复杂性。存在这些规则是为了使代码库易于管理，同时仍允许编码人员高效地使用语言功能。

代码风格，也称为可读性，就是我们称之为C++ 编码规范的约定。

请注意，本指南不是C++ 语言教程：我们假设读者熟悉该语言。

[C++ 语言版本](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#c-%E8%AF%AD%E8%A8%80%E7%89%88%E6%9C%AC)

默认情况下，项目交付代码使用**C**++03标准，以获得更广泛的编译环境兼容性。**C**++03是**C**++98标准的修订版，两者完全兼容。

项目交付之外的C++ 代码，如内部使用的开发、调试、部署工具，最高使用**C**++17标准。勿使用**C**++2x特性。

在项目中使用非标准扩展前，优先考虑多平台的可移植性。

[头文件](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A4%B4%E6%96%87%E4%BB%B6)

通常，每个.cpp文件都应该有一个关联的.h文件。有一些常见的例外，例如单元测试和.cpp仅包含**main**()函数的小文件 。

正确使用头文件会对代码的可读性，大小和性能产生巨大影响。

以下规则将指导您使用头文件的各种麻烦。

[**头文件保护**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A4%B4%E6%96%87%E4%BB%B6%E4%BF%9D%E6%8A%A4)

所有头文件都**必须**有多重包含保护。有#**ifndef** #**define** #**endif**及#**pragma** once两种方式，选用第一种。

第一种方式中符号名称的格式应为

\_<**PROJECT**>\_<**PATH**>\_<**FILE**>\_H\_

为了保证唯一性，它们应该基于项目源树中的完整路径。例如，foo/src/bar/baz.h项目中的文件foo应具有以下保护：

#**ifndef** \_FOO\_BAR\_BAZ\_H\_

#**define** \_FOO\_BAR\_BAZ\_H\_

// ...

#**endif** // \_FOO\_BAR\_BAZ\_H\_

[**头文件依赖**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A4%B4%E6%96%87%E4%BB%B6%E4%BE%9D%E8%B5%96)

头文件中不使用前置声明，直接#**include**所需的其他头文件。

**注意：** 只包含且必须包含必要的头文件，避免包含不需要的头文件。

**例外：** SDK接口的**Type**\*句柄类型，可使用头文件中前置声明的方式，隐藏实现细节。

[**函数参数顺序**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E5%8F%82%E6%95%B0%E9%A1%BA%E5%BA%8F)

定义参数时，参数顺序为：输入参数在前，输出参数在后。 输入参数为传值或常数引用；输出参数为非常数指针，并添加小写字母o作为前缀。

本条应尽量遵守，类或结构体中的输入输出两用参数作为中间参数。

[**函数多返回值**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E5%A4%9A%E8%BF%94%E5%9B%9E%E5%80%BC)

函数需要有多个返回值时，一般只在函数返回值中直接返回执行结果或错误码，其他数据通过输出参数返回。

如果返回值的拷贝代价较小，所有返回值在32字节以内（包含），也允许将返回值全部定义在结构体中，函数返回此结构体实例。如：

**struct** **GetResult**

{

**int** errorCode;

**int** dataSize;

// data中包含很多数据，但浅拷贝只涉及指针的拷贝

**void** \*data;

};

GetResult **Get**(**int** param)

{

GetResult getResult;

getResult.errorCode = 0;

getResult.dataSize = GetDataSize(param);

getResult.data = GetData(param);

/\* C++11标准以前没有返回值优化，此处会引发拷贝构造

\* 默认的拷贝构造函数会浅拷贝每个数据成员

\*/

**return** getResult;

}

如果项目允许使用C++11及以上标准：由于编译器有返回值优化，以上方式为建议多返回值方式；也允许使用std::tuple元组同时返回多个结果。

[**包含头文件的名词及次序**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%8C%85%E5%90%AB%E5%A4%B4%E6%96%87%E4%BB%B6%E7%9A%84%E5%90%8D%E8%AF%8D%E5%8F%8A%E6%AC%A1%E5%BA%8F)

包含次序标准化可增强可读性，避免依赖。次序为：C库、C++库、其他库的.h、项目内的.h。

即：

C系统文件

C++系统文件

其他库头文件

本项目内头文件

项目内头文件按照项目源码的目录树结构排序，避免使用UNIX路径.和..

**总结：**

(1) 避免多重包含

(2) 标准化函数参数顺序可提高可读性和易维护性

(3) 包含文件使用.和..虽然方便，却易混乱，使用完整项目路径更清晰、美观。

[作用域](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BD%9C%E7%94%A8%E5%9F%9F)

[**命名空间**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%91%BD%E5%90%8D%E7%A9%BA%E9%97%B4)

在.cpp文件中，提倡使用不具名命名空间。如果使用具名命名空间，基于项目或路径名称而不是**using**指示符。命名空间内不用缩进。

**定义：** 命名空间将全局作用域细分为不同的具名作用局，可有效防止全局作用局的命名冲突。

**优点：** 命名空间的命名轴线可嵌套。父级命名不同时即使当前命名重复也不会冲突。

**缺点：** 不具名命名空间容易违背C++中唯一定义的原则。

**结论：** 合理使用

[**嵌套类**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%B5%8C%E5%A5%97%E7%B1%BB)

不要将嵌套类定义为**public**，除非是接口的一部分。

[**非成员函数、静态成员函数、全局函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%9D%9E%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0%E9%9D%99%E6%80%81%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0%E5%85%A8%E5%B1%80%E5%87%BD%E6%95%B0)

使用命名空间内的非成员函数和静态成员函数，尽量不使用全局函数。

[**局部变量**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F)

将函数变量尽可能至于最小作用域内，在声明变量时将其初始化。

C++允许在函数的任何位置声明变量，提倡在尽可能小的作用域中声明变量，离第一次使用越近越好。这样代码易于阅读且易于定位变量的声明位置、变量类型和初始值。

*特别是，应使用初始化代替声明再赋值的方式。*

[**全局变量**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F)

禁止**class**类型的全局变量，多线程代码中禁止非常数全局变量，禁止使用函数返回值初始化全局变量。

**class**类型的全局变量内部构造、析构等初始化操作每次生成有可能变化，从而导致难以发现的缺陷。因此，禁止使用**class**类型的全局变量（包括std::string、std::vector等）。

全局的字符串常量，使用C风格的字符串，不要使用std::string。

如果必须使用**class**或标准库容器类型的全局变量，则应定义为**class**的原始指针类型，并做Lazy Init处理。

**注意：** 静态成员变量视为全局变量，也不能是**class**类型。

**总结：**

(1) .cpp中的不具名命名空间可避免命名冲突且限定作用域，避免使用**using**提示符污染命名空间；

(2) 嵌套类符合局部使用原则，可惜不能再其他头文件前置声明，尽量不要**public**；

(3) 尽量不用全局函数和全局变量，考虑作用域和命名空间限制，尽量单独形成编译单元；

(4) 多线程中的全局变量不要使用**class**类型（含标准库容器），避免编译器未定义行为（UB）引发的缺陷；

作用域的使用，除了考虑名称污染、可读性外，主要是为了降低耦合度，提高编译、执行效率。

[C++类](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#c%E7%B1%BB)

类是C++中基本的代码单元，被广泛使用。

[**构造函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)

构造函数只初始化没有实际意义的成员变量。需要初始化的数据在Init()函数中集中初始化。

**定义：** 在构造函数中执行初始化操作

**优点：** 排版方便，无需担心类是否初始化

**缺点：** 在构造函数中执行初始化操作容易引起以下问题：

(1) 构造函数中不易报告错误，不能使用异常

(2) 操作失败会造成对象初始化失败，引起不确定状态

(3) 构造函数内调用虚函数，调用不会派发到子类实现中，即使当前不需要子类实现，但依然是隐患

(4) 如果有人创建该类的全局变量（这样做违背禁止类作为全局变量的准则），构造函数在**main**()函数执行前调用，有可能影响代码中使用flag的逻辑。

**结论：**

如果对象需要有意义的初始化，使用Init()函数合并需要初始化的内容，并增加一个成员变量标记该对象是否初始化成功。

[**默认构造函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%BB%98%E8%AE%A4%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)

如果一个类定义了若干个成员变量又没有其他构造函数，需要定义一个默认构造函数，否则编译器将自动生成默认构造函数。

**定义：** 新建一个没有参数的对象时，默认构造函数被调用，当使用**new**时，默认构造函数会被调用。

**优点：** 默认将结构体初始化为"不可能的"值，使调试更加容易

**缺点：** 对代码编写者来说，这是多余的工作。

**结论：** 如果类中定义了成员变量，没有提供其他构造函数，则需要自定义一个默认构造函数（没有参数）。默认构造函数用于初始化对象，使对象内部状态一致、有效。

提供默认构造函数的原因：编译器自动生成的构造函数不会对对象进行初始化。

如果定义的类继承自现有类，且没有新增新的成员变量，则不需要为新类定义默认构造函数。

[**明确的构造函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%98%8E%E7%A1%AE%E7%9A%84%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)

单参数构造函数使用C++关键字**explicit**

**定义：** 通常，只有一个参数的构造函数会发生隐式转换。避免此问题，可声明为**explicit**

**优点：** 避免不合时宜的变换。

**缺点：** 无

**结论：**

所有单参数构造函数必须是明确的，在类定义中加**explicit**

拷贝构造函数不需特意声明为**explicit**

[**拷贝构造函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%8B%B7%E8%B4%9D%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)

仅在代码中需要拷贝一个类对象的时候使用拷贝构造函数；不需要拷贝时使用

**定义：** 通过拷贝新建对象时可使用拷贝构造函数（特别是对象的传值时）。

**优点：** 拷贝构造函数使得拷贝对象更加容易，标准库容器要求所有内容可拷贝、可赋值。

**缺点：** C++中对象的隐式拷贝是导致很多性能问题和缺陷的根源。拷贝构造函数降低了代码的可读性，相比按引用传递，跟踪按值传递的对象更加困难，对象修改的地方变的难以捉摸。

**结论：** 禁止使用拷贝构造函数，容器中的元素使用类的指针指针类型。

[**结构体和类**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%BB%93%E6%9E%84%E4%BD%93%E5%92%8C%E7%B1%BB)

仅当只有数据时使用**struct**，其他一概使用**class**。

**注意：** 类和结构体的成员变量使用不同的命名规则

[**接口（Interface）**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%8E%A5%E5%8F%A3interface)

接口是指满足特性条件的类，这些类以Interface为后缀（命名约定）。

**定义：** 当一个类满足以下要求时，称之为纯接口：

(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，或定义不含参数的protected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

接口类不能被直接实例化，因为他声明了纯虚函数。为确保接口类所有实现可被正确销毁，必须为之申明虚析构函数。

**优点：** 代码维护人员看到Interface就知道不能为该接口类实现函数或非静态数据成员

**缺点：** 增加类名长度带来阅读不便。同时，接口细节不应暴露给用户。

**结论：** 满足上述需求的类才使用Interface结尾，但满足也可以不使用。

[**继承**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%BB%A7%E6%89%BF)

尽量使用组合而不是继承。如果使用继承，只用**public**继承。

**定义：** 接口继承：继承自纯接口的抽象类。实现继承：继承自一个非纯接口类。

允许接口继承，尽量不使用实现继承。

[**多重继承**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A4%9A%E9%87%8D%E7%BB%A7%E6%89%BF)

多重继承的使用场景：仅允许最多一个基类中包含实现，其他基类都是以Interface为后缀的纯接口。

[**操作符重载**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%93%8D%E4%BD%9C%E7%AC%A6%E9%87%8D%E8%BD%BD)

除少数特定环境外（例外），不要重载操作符。

**定义：** 一个类合一定义诸如+、/等操作符，使其可以像内建类型一样直接使用。

**优点：** 提高代码可读性，比如Equals()用==代替

**缺点：**

(1) 混淆直觉，让开发者误以为一些耗时的操作像内建操作那样轻巧；

(2) 查找重载操作符的调用处更加困难，查找Equals()比==调用容易的多。

(3) 有的操作符可以操作指针，容易导致缺陷

(4) 副作用，如重载操作符&不能被前置声明

**结论：**

禁止重载操作符，尤其是赋值操作符严格禁止。

**例外：**

类型需要作为Key放入std::map，或类型需要在容器中排序。则应当遵从标准库要求，提供相应的操作符重载。

重载原则：单目操作符（如**operator**!）定义为类成员函数；双目操作符（如**operator**<, operator==）定义为全局函数，并声明为类友元。

[**getter/setter**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#gettersetter)

将数据成员私有化，并提供相关的getter/setter

[**声明次序**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%A3%B0%E6%98%8E%E6%AC%A1%E5%BA%8F)

类中声明次序：**public**、**protected**、**private**，如果某一块没有，直接忽略。

每一块中，声明次序如下：

(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；

.cpp文件中的函数定义应尽可能声明次序一致。

禁止将大型函数内联到类的定义中。

[**编写短小函数**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%BC%96%E5%86%99%E7%9F%AD%E5%B0%8F%E5%87%BD%E6%95%B0)

函数长度尽量不超过40行，倾向选择短小、凝练的函数。

函数尽量短小、简单，便于他人阅读和修改代码。处理代码时如发现长函数，考虑分割为若干短小、易于管理的若干函数。

**总结：**

(1) 不在构造函数中做太多逻辑相关的初始化；

(2) 编译器默认构造函数不会初始化变量，如果定义了其他构造函数，编译器不再提供，需要编码这自行提供默认构造函数；

(3) 为避免隐式转换，需将单参数构造函数声明为**explicit**；

(4) 为避免拷贝构造函数、赋值操作的滥用和编译器自动生成，可声明为**private**且无需实现；

(5) 仅在作为数据集合时使用**struct**；

(6) 组合>实现继承>接口继承>私有继承，子类重载的虚函数也要声明**virtual**关键字；

(7) 避免使用多重继承，唯一的使用场景是最多一个基类含实现，其他基类纯接口；

(8) 接口类类名以Interface为后缀，全部使用纯虚函数；

(9) 禁止重载操作符

(10) 存取函数一般内联在头文件中；

(11) 声明次序: public -> protected -> private；

(12) 函数体尽量短小、紧凑，功能单一。

[其他C++特性](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%85%B6%E4%BB%96c%E7%89%B9%E6%80%A7)

[**智能指针**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%99%BA%E8%83%BD%E6%8C%87%E9%92%88)

禁止使用std::auto\_ptr。

[**const**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#const)

引用形参用const，否则使用指针形参

[**函数重载的使用要清晰、易读**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E9%87%8D%E8%BD%BD%E7%9A%84%E4%BD%BF%E7%94%A8%E8%A6%81%E6%B8%85%E6%99%B0%E6%98%93%E8%AF%BB)

[**禁止使用缺省函数参数（测试框架例外）**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%A6%81%E6%AD%A2%E4%BD%BF%E7%94%A8%E7%BC%BA%E7%9C%81%E5%87%BD%E6%95%B0%E5%8F%82%E6%95%B0%E6%B5%8B%E8%AF%95%E6%A1%86%E6%9E%B6%E4%BE%8B%E5%A4%96)

[**禁止使用变长数组**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%A6%81%E6%AD%A2%E4%BD%BF%E7%94%A8%E5%8F%98%E9%95%BF%E6%95%B0%E7%BB%84)

用std::vector代替

[**合理使用友元**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%90%88%E7%90%86%E4%BD%BF%E7%94%A8%E5%8F%8B%E5%85%83)

[**限制使用C++异常，测试框架例外**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%99%90%E5%88%B6%E4%BD%BF%E7%94%A8c%E5%BC%82%E5%B8%B8%E6%B5%8B%E8%AF%95%E6%A1%86%E6%9E%B6%E4%BE%8B%E5%A4%96)

底层库允许使用异常，并将异常限定在模块内部，不要抛出到外部。

构造、析构函数中不要抛出异常。

并尽力避免在构造、析构函数中执行会引发异常的操作。如难以避免，也应捕获并处理所有异常，而不能抛出。

[**禁止使用RTTI**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%A6%81%E6%AD%A2%E4%BD%BF%E7%94%A8rtti)

[**类型转换**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%B1%BB%E5%9E%8B%E8%BD%AC%E6%8D%A2)

使用**static\_cast**<Type>(var)等C++的类型转换，禁止使用(**Type**)**var**强制类型转换。

[**禁止使用流**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%A6%81%E6%AD%A2%E4%BD%BF%E7%94%A8%E6%B5%81)

除日志、序列化需要，其他场景禁止使用std::stream，使用printf代替。

[**const的使用**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#const%E7%9A%84%E4%BD%BF%E7%94%A8)

任何可以使用的场景都要使用**const**，且**const**在前。

[**使用确定大小的整型**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BD%BF%E7%94%A8%E7%A1%AE%E5%AE%9A%E5%A4%A7%E5%B0%8F%E7%9A%84%E6%95%B4%E5%9E%8B)

使用确定大小的整型，除表示数组位数外的场景禁止使用无符号型

[**可移植性**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%8F%AF%E7%A7%BB%E6%A4%8D%E6%80%A7)

格式化输出及结构对齐时，注意32位和64位系统差异

[**预处理宏**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%A2%84%E5%A4%84%E7%90%86%E5%AE%8F)

除字符串化和连接的场景外，一律禁止使用宏。要求使用内联函数、枚举和常量代替宏。

[**0和NULL**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#0%E5%92%8Cnull)

整数用0，实数用0.0，指针用nullptr，字符（串）用'\0'

[**sizeof**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#sizeof)

用sizeof(**variableName**)代替sizeof(**TypeName**)

[**Boost库**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#boost%E5%BA%93)

暂不使用Boost

[命名约定](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%91%BD%E5%90%8D%E7%BA%A6%E5%AE%9A)

一致性命名规则，包括类型、变量、函数、常量、宏等等。

[**通用命名规则**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%80%9A%E7%94%A8%E5%91%BD%E5%90%8D%E8%A7%84%E5%88%99)

函数、变量、文件命名应具有描述性，不要过分缩写。

类型和变量应该是名词，函数名用"命令性质"的动词。

命名方法：

尽可能给出描述性名词，提高代码可读性，目标是让别人快速理解。

禁止使用模糊的缩写或随意的字符。

类型和变量名一般为名词，如numErrors。禁止使用泛称，尽量用完整的单词描述清楚作用和意图，不要怕字多。例如**struct** **ProcData**就不是好的命名。

函数名通常是指令性的，如OpenFile()、SetNumErrors()，访问函数需要描述的更细致，要与其访问的变量相吻合。

缩写：

除非是放在项目外也非常明了的缩写，如IOT、DNS、IP等

除linux命令常用缩写（如 ls、pwd、usr等）外，不要用省略字母的缩写。如count不要写成cnt

[**文件命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%96%87%E4%BB%B6%E5%91%BD%E5%90%8D)

文件名同类名一致，包括大小写。已有命名约定要求的项目遵循项目要求。

不要使用已经存在于系统中的文件名，如Linux系统不要使用/usr/**include**下的文件名，如**db**.h。

[**类型命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%B1%BB%E5%9E%8B%E5%91%BD%E5%90%8D)

类型命名每个单词以大写字母开头，不包含下划线，如：MyExcitingClass、MyExcitingEnum。

所有类型命名：类、结构体、类型定义（typedef）、枚举，使用相同约定，例如：

// class and structs

**class** **UrlTable** {...};

**class** **UrlTableTester** {...};

**struct** **UrlTableProperties** {...};

// typedefs

**typedef** hash\_map<UrlTableProperties \*, string> PropertiesMap;

// enums

**enum** UrlTableErrors{...};

[**变量命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%8F%98%E9%87%8F%E5%91%BD%E5%90%8D)

变量名使用驼峰命名法，第一个单词以小写开头，其他单词首字母大小，单词之间不使用分隔符。类的私有成员添加下划线结尾。示例如下：

**class** **SampleClass**

{

**public**:

// 函数参数使用驼峰命名，无特殊前缀或后缀

**void** **SampleFunction**(**int** parameterVariable)

{

// 临时变量一般使用驼峰命名，无特殊前缀或后缀

**int** localVariable = 0;

// 无特殊含义的临时变量，可以使用全小写，同重要变量有所区分

**for** (**int** i = 0, ncount = 50; i < ncount; ++i)

{

**int** retval = OtherFunctionCall();

**if** (retval != 0)

{

**int** errcode = write(...);

}

}

}

**private**:

// 类数据成员，使用下划线结尾

**int** dataMember\_;

};

**struct** **Message**

{

// 结构体成员默认为public，不使用下划线结尾

**int** index;

string msg;

};

[**全局变量命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F%E5%91%BD%E5%90%8D)

不建议使用全局变量。如果一定要用，以g\_为前缀以便与局部变量区分。

[**静态变量命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%9D%99%E6%80%81%E5%8F%98%E9%87%8F%E5%91%BD%E5%90%8D)

不建议使用静态变量。如果一定要用，以s\_为前缀以便与非静态变量区分。静态全局变量仅需添加s\_前缀。

[**常量命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%B8%B8%E9%87%8F%E5%91%BD%E5%90%8D)

常量命名全部大写，单词间以下划线分隔，如：

**const** **int** DAYS\_IN\_A\_WEEK = 7;

[**函数命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E5%91%BD%E5%90%8D)

普通函数，函数名大写字母开头，每个单词首字母大写，没有下划线：

AddTableEntry()

getter/setter函数，函数名要与get/set的变量名匹配。

**class** **MyClass**

{

**public**:

// ...

**int** **GetNumEntries**() **const** { **return** numEntries\_; }

**void** **SetNumEntries**(**int** numEntries)

{

numEntries\_ = numEntries;

}

**private**:

**int** numEntries\_;

};

[**名空间命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%90%8D%E7%A9%BA%E9%97%B4%E5%91%BD%E5%90%8D)

全部小写，命名基于项目名称和目录结构

**namespace** pengze

{

}

[**枚举命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%9E%9A%E4%B8%BE%E5%91%BD%E5%90%8D)

枚举值全部大写，单词之间用下划线分隔 MY\_EXCITING\_ENUM\_VALUE。

枚举名称属于类型，因此大小写混合。

**enum** UrlTableErrors {

OK = 0,

ERROR\_OUT\_OF\_MEMORY = 1,

ERROR\_MALFORMED\_INPUT = 2,

};

[**宏命名**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%AE%8F%E5%91%BD%E5%90%8D)

尽量使用枚举、常量、内联替代宏。

如果必须要用，宏的名称全部大写，单词之间用下划线分隔 MY\_MACRO\_THAT\_SCARES\_SMALL\_CHILDREN\_AND\_ADULTS\_ALIKE

[**命名规则例外**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%91%BD%E5%90%8D%E8%A7%84%E5%88%99%E4%BE%8B%E5%A4%96)

参考现有或相近命名约定

[代码注释](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BB%A3%E7%A0%81%E6%B3%A8%E9%87%8A)

[**注释风格**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%B3%A8%E9%87%8A%E9%A3%8E%E6%A0%BC)

说明代码功能的关键注释统一使用/\* \*/。

面向外部的头文件中，统一使用Doxygen注释风格。

[**文件注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%96%87%E4%BB%B6%E6%B3%A8%E9%87%8A)

文件注释需说明当前文件在项目中的核心作用。并使用Doxygen注释风格：

/\*\*

\* @file Sample.h

\* @author Author Name (username@ghostcloud.cn)

\* @brief Doxygen 注释风格示例

\* @version 0.1

\* @date 2021-03-11

\*

\* @copyright Copyright (c) 2021

\*

\*/

#**ifndef** \_GHOSTCLOUD\_PENGZE\_SAMPLE\_H\_

#**define** \_GHOSTCLOUD\_PENGZE\_SAMPLE\_H\_

#**endif** /\* !\_GHOSTCLOUD\_PENGZE\_SAMPLE\_H\_ \*/

[**类注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%B1%BB%E6%B3%A8%E9%87%8A)

类的头文件中，对类的成员函数、成员变量做说明。

[**函数注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E6%B3%A8%E9%87%8A)

函数申明处注释的内容：

(1) 函数作用：当前函数需要达成的目的

(2) 函数参数：说明是输入还是输出，及参数含义

(3) 返回值：含义和类型

面向外部的接口函数注释使用Doxygen风格：

/\*\*

\* @brief 创建一个示例对象

\*

\* @details 详细说明（可选）

\*

\* @note 注意事项说明（可选）

\*

\* @param arg0 参数0，参数0说明

\* @param arg1 参数1，参数1说明

\* @param arg2 参数2，参数2说明

\*

\* @return int ErrorCode

\* @retval 0 成功

\* @retval -1 失败

\*

\*/

**int** **CreateSample**(**int** arg0, **int** arg1, **int** arg2);

[**变量注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%8F%98%E9%87%8F%E6%B3%A8%E9%87%8A)

变量在上一行使用 // 注释

// a variable

**int** a;

[**实现注释**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%AE%9E%E7%8E%B0%E6%B3%A8%E9%87%8A)

复杂或重要的逻辑实现，需要说明实现的意图、目的、主要思想。

**注意：** 注释是对逻辑的解释，而非代码的翻译。要假定代码阅读者比你更熟悉C++语言本身，但他不明白为什么要这样实现。

[**标点、拼写和语法**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%A0%87%E7%82%B9%E6%8B%BC%E5%86%99%E5%92%8C%E8%AF%AD%E6%B3%95)

注释要易读

[**TODO**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#todo)

临时、短期的解决方案，需要使用TODO注释。临时注释使用 //

TODO大写，需要备注作者名，备查找： // TODO(Luffy): 需要优化逻辑

[代码格式](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BB%A3%E7%A0%81%E6%A0%BC%E5%BC%8F)

团队的C++开发规范约定，1个Tab=4个空格

[**行长度**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E8%A1%8C%E9%95%BF%E5%BA%A6)

每一行字符数不超过80

这一条存在例外情况：

(1) 如果一行注释包含了超过80字符的命令或URL，出于复制粘贴方便可以超过80字符；

(2) 包含长路径的可以超过80列，尽量避免

(3) 头文件保护可以不遵从该原则

[**括号空格与换行**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%8B%AC%E5%8F%B7%E7%A9%BA%E6%A0%BC%E4%B8%8E%E6%8D%A2%E8%A1%8C)

**void** **MyFunctionName**()

{

// 操作符之间空格隔开

**int** a = 50 \* 2 + 1;

// 关键字与括号之间留一个空格

**if** (a == 50)

{

}

**else** **if** (a > 60)

{

}

**else**

{

}

**while** (a == 70)

{

}

// do while的while与花括号在同一行

**do**

{

} **while** (a < 90);

// for循环分号后面加空格

**for** (**int** i = 0; i < 70; ++i)

{

}

}

[**非ASCII字符**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%9D%9Eascii%E5%AD%97%E7%AC%A6)

尽量不使用ASCII字符，使用时必须使用UTF-8格式

[**缩进空格 or 制表符Tab**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%BC%A9%E8%BF%9B%E7%A9%BA%E6%A0%BC-or-%E5%88%B6%E8%A1%A8%E7%AC%A6tab)

使用空格缩进，每级缩进4个空格。

[**函数声明与定义**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E5%A3%B0%E6%98%8E%E4%B8%8E%E5%AE%9A%E4%B9%89)

返回类型和函数名在同一行，参数也在同一行。

如果一行文本较多写不下所有参数，则参数换行空格对齐：

ReturnType LongClassName::ReallyReallyReallyLongFunctionName(

Type parName1,

Type parName2,

Type parName3)

{

DoSomething(); // 2 space indent

...

}

**注意：**

(1) 返回值和函数在同一行

(2) 左圆括号(总是和函数名在同一行

(3) 函数名和左圆括号之间没有空格

(4) 圆括号与参数没有空格

(5) 函数的左花括号另起一行，单独置于一行

(6) 函数的右花括号在函数最后，单独置于一行

(7) 右圆括号和左花括号间换行

(8) 函数声明和实现处的所有形参名称必须保持一致

(9) 所有形参尽可能对齐

(10) 默认一级缩进为4个空格

(11) 独立封装的参数保持4个空格缩进

const函数，关键字**const**和最后一个参数置于同一行

ReturnType LongClassName::ReallyReallyReallyLongFunctionName(

Type parName1,

Type parName2,

Type parName3) **const**

{

DoSomething(); // 2 space indent

...

}

暂不使用的参数需在函数的注释中说明

[**函数调用**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%87%BD%E6%95%B0%E8%B0%83%E7%94%A8)

尽量放在同一行，否则将括号内的参数换行对齐

[**条件语句**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%9D%A1%E4%BB%B6%E8%AF%AD%E5%8F%A5)

条件语句的左花括号放在条件语句的行位，格式上与函数区分。

注意：条件内哪怕只有一行语句，也要写全花括号，方便后续维护。

有些条件语句写在一行可以增强可读性，仅当语句简单且没有**else**子句时可以使用：

**if** (x == kFoo) **return** **new** Foo();

**if** (x == kBar) **return** **new** Bar();

注意：如果有else子句，则禁止不带花括号。

[**循环和开关选择语句（loop and switch）**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%BE%AA%E7%8E%AF%E5%92%8C%E5%BC%80%E5%85%B3%E9%80%89%E6%8B%A9%E8%AF%AD%E5%8F%A5loop-and-switch)

switch 语句尽量使用大括号分块

switch 语句必须要有default存在，如正常不会执行则以异常日志或控制台信息的方式处理。

空循环使用{}或者continue，禁止循环只用一个分号

[**指针和引用表达式**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%8C%87%E9%92%88%E5%92%8C%E5%BC%95%E7%94%A8%E8%A1%A8%E8%BE%BE%E5%BC%8F)

句号.、箭头->前后不能有空格。

\*、&作为解引用、取值操作符使用时，要紧贴其变量；作为指针和引用声明时，建议\*紧贴变量名，&紧贴类型名。

value = \*test;

value = &test;

value = c.test;

value = c->test;

**char** \*cinput;

**const** string& strName;

[**布尔表达式**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%B8%83%E5%B0%94%E8%A1%A8%E8%BE%BE%E5%BC%8F)

如果布尔表达式超过标准行宽（80字符），则断行且要求逻辑操作符置于行尾。

**if** (thisOneThing > thisOtherThing &&

aThirdThing == aFourthThing &&

yetAnother & lastOne)

{

...

}

有多个逻辑操作符时，考虑使用圆括号()提示运算优先级

[**返回值**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E8%BF%94%E5%9B%9E%E5%80%BC)

函数返回（即：**return** 表达式）时不要使用圆括号。

[**初始化操作**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%88%9D%E5%A7%8B%E5%8C%96%E6%93%8D%E4%BD%9C)

使用等号=完成

[**预处理**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E9%A2%84%E5%A4%84%E7%90%86)

不要缩进，从行首开始。及时在缩进代码块中，也要从行首开始。

[**类格式**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E7%B1%BB%E6%A0%BC%E5%BC%8F)

类格式参考本规范声明次序章节。

[**初始化列表**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%88%9D%E5%A7%8B%E5%8C%96%E5%88%97%E8%A1%A8)

初始化列表置于同一行，或换行空格补齐

[**命名空间格式化**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%91%BD%E5%90%8D%E7%A9%BA%E9%97%B4%E6%A0%BC%E5%BC%8F%E5%8C%96)

命名空间内容不缩进。

**namespace** pengze

{

**class** **MyClass**

{

};

}

[**水平空白**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E6%B0%B4%E5%B9%B3%E7%A9%BA%E7%99%BD)

行尾不要增加无意义的空白。当行尾有注释需要空格补齐时可以适当留空。

[**垂直空白**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%9E%82%E7%9B%B4%E7%A9%BA%E7%99%BD)

垂直空白尽量少。函数之间、模块之间留一样空白

[**规则之例外**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E8%A7%84%E5%88%99%E4%B9%8B%E4%BE%8B%E5%A4%96)

前文提到的编码习惯基本是强制性的，但优秀的规则都允许例外。

[**Windows代码**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#windows%E4%BB%A3%E7%A0%81)

在Windows平台开发，除系统API不能更改外，自主开发内容需遵循本规则。

[**团队合作**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E5%9B%A2%E9%98%9F%E5%90%88%E4%BD%9C)

团队风格一致性，代码高可读性。

[C++ 标准库](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#c-%E6%A0%87%E5%87%86%E5%BA%93)

C++ 标准制订了标准库的接口规范，现代C++ 编译器一般都内置一个特定的标准库实现。

早期的C++ 标准库仅包含一套基于模板的泛型库，这套库被成为标准模板库STL，因此STL一度成为C++ 标准库的代称。**C**++11之后，标准库做了很多扩充，支持更多通用特性，因此STL仅仅是标准库的一个子集。

C++ 标准库使用的一般原则，可参考\*Effective C++\*一书，这里摘抄一些必须遵循的原则：

[**使用 .empty() 代替 .size() != 0 判断容器是否为空**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#%E4%BD%BF%E7%94%A8-empty-%E4%BB%A3%E6%9B%BF-size--0-%E5%88%A4%E6%96%AD%E5%AE%B9%E5%99%A8%E6%98%AF%E5%90%A6%E4%B8%BA%E7%A9%BA)

早期版本的标准库，某些容器的size函数实现是遍历所有元素计数，时间复杂度为O(N)，但empty函数一定是O(1)的。

因此如果仅仅想检查容器是否为空，一定使用empty函数。这样语义也更清晰。

std::list<**int**> myList;

// 错误做法，可能是O(N)

**if** (myList.size() != 0) { ... }

// 正确做法，一定是O(1)

**if** (myList.empty()) { ... }

[**for 循环中 erase 的标准写法**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#for-%E5%BE%AA%E7%8E%AF%E4%B8%AD-erase-%E7%9A%84%E6%A0%87%E5%87%86%E5%86%99%E6%B3%95)

#**include** <string>

#**include** <list>

#**include** <unordered\_map>

std::list<std::string> myList = { "a", "b", "c", "d" };

**for** (**auto** it = myList.begin(); it != myList.end(); ) {

**if** (condition) {

// erarse会返回擦除之后

it = myList.erase(it);

}

**else** {

++it;

}

}

std::unordered\_map<std::string, std::string> myMap = {

{ "a", "1" }, { "b", "2" }, { "c", "3" }

};

**for** (**auto** it = myMap.begin(); it != myMap.end(); ) {

**if** (condition) {

it = myMap.erase(it);

}

**else** {

++it;

}

}

[C++11 新特性使用](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#c11-%E6%96%B0%E7%89%B9%E6%80%A7%E4%BD%BF%E7%94%A8)

[**Lambda 捕获列表必须完整填写**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#lambda-%E6%8D%95%E8%8E%B7%E5%88%97%E8%A1%A8%E5%BF%85%E9%A1%BB%E5%AE%8C%E6%95%B4%E5%A1%AB%E5%86%99)

**int** i = 100

**double** d = 3.14159;

std::string s = "pi";

**auto** func = [i, d, s]() {

...

}

[**Lambda 捕获列表，写[&]前请三思**](http://newben.ghostcloud.cn:30038/pages/viewpage.action?pageId=15073658#lambda-%E6%8D%95%E8%8E%B7%E5%88%97%E8%A1%A8%E5%86%99%E5%89%8D%E8%AF%B7%E4%B8%89%E6%80%9D)

std::function<**void**()> Test() {

std::string str = "my string";

// 非法操作！将一个局部对象str的引用传递到函数作用域之外。

// 函数对象func捕获了str的引用，而将func作为返回值返回到函数之外，str的引用也随着func出了函数作用域

// 外部调用func就会访问局部对象str的引用，出现内存异常。

**auto** func = [&str]() {

...

};

**return** func;

}

**void** **Test2**(MyClass& c) {

std::string str = "my string";

// 非法操作！将一个局部对象str的引用传递到函数作用域之外。

// 函数对象func捕获了str的引用，而将func设置到外部对象上，str的引用也随着func出了函数作用域

// 外部调用func就会访问局部对象str的引用，出现内存异常。

**auto** func = [&str]() {

...

};

c.SetCallback(func);

}

**void** **Test3**() {

// [&]引用捕获唯一合法也是推荐的用法，算法回调后函数对象不会再被使用

// 因此传递局部对象的引用是安全的。

std::string s;

std::find\_if(arr.begin(), arr.end(), [&s](**const** **auto**& x) {

**if** (...) **return** true;

});

}