C++代码规范

目录

1命名约定 3

1.1通用命名约定 3

1.2特殊命名约定 5

2.文件 8

2.1源代码文件 8

2.2nclude文件与Include语句 8

3.语句 10

3.1类型 10

3.2变量 10

3.3循环 11

3.4条件 12

3.5杂项 14

4.布局与注释 14

4.1布局 14

4.2空白符 18

[4.3注释 2](#_Toc18385202)0



**1命名约定**

* 1. **通用命名约定**

**类型名必须混合大小写且以大写开始。**

*Line, SavingsAccount*

C++开发团体的通用习惯。

**变量名必须混合大小写且以小写开始。**

*line, savingsAccount*

C++开发团体的通用习惯。使得容易分辨变量与类型，并且有效解决了如申明Line line时潜在的名字冲突。

**常量（含枚举值）的命名必须全部为大写，以下划线分隔单词。**

*MAX\_ITERATIONS, COLOR\_RED, PI*

C++开发团体的通用习惯。一般情况下，少用这些常量。在很多实现中，用方法替代值是较好的选择：

*int getMaxIterations() // NOT: MAX\_ITERATIONS = 25*

*{*

*return 25;*

*}*

这种形式容易阅读，并且确保了使用统一的接口访问类值。

**方法与函数的命名必须是动作性的，采用大小写混合的写法以小写开始。**

*getName(), computeTotalWidth()*

C++开发团体的通用习惯。这同样适用于变量名，但函数在C++中已经通过特定的形式与变量相区分。

**名字空间的命名表应该全部小写。**

*analyzer, iomanager, mainwindow*

C++开发团体的通用习惯。

**模板类型名应该用单个大写字母表示。**

*template<class T> ...*

*template<class C, class D> ...*

C++开发团体的通用习惯。这使得模板名明显不同于使用的其他名字。

**缩写与首写字母缩写作为名字不要全部大写。**

*exportHtmlSource(); // NOT: exportHTMLSource();*

*openDvdPlayer(); // NOT: openDVDPlayer();*

对基本的名字使用全部大写会与上面给出的命名约定冲突。这种类型的变量将有诸如dVD；hTML等名字。很明显这样可读性太差。上面的例子中，另外一个问题是，当名字与另一个连接时，可读性严重降低；缩写之后的单词不再是它应该代表的意思。

**全局变量应该通过:: 操作符使用。**

*::mainWindow.open(), ::applicationContext.getName()*

通常，避免使用全局变量。考虑用singleton对象代替。

**全局变量的命名加g\_前缀，类得成员变量加m\_前缀，静态变量加s\_前缀。**

除名字与类型，变量的范围也是非常重要的。通过标识类范围的变量会很容易将它们与局部临时变量区分。类的变量比方法内的变量更有意义，编程人员需要更细心对待。

**类的私有变量应该带有下划线作后缀。（Option）**

除名字与类型，变量的范围也是非常重要的。通过下划线标识类范围的变量会很容易将它们与局部临时变量区分。类的变量比方法内的变量更有意义，编程人员需要更细心对待。

*class SomeClass {*

*private:*

*int m\_length\_;*

}

*void setDepth (int depth)*

*{*

*depth\_ = depth;*

*}*

**普通变量应该与它们的类型使用同样的名字。**

*void setTopic (Topic topic) // NOT: void setTopic (Topic value)*

*// NOT: void setTopic (Topic aTopic)*

*// NOT: void setTopic (Topic x)*

*void connect (Database database) // NOT: void connect (Database db)*

*// NOT: void connect (Database oracleDB)*

通过减少术语与名字的使用数目减少复杂性。同时，可仅仅通过变量名推导类型。

假如由于某种原因这种约定看起来不合适，这很可能意味这选择了一个不好的类型名。

有些变量有相应的角色，对这些变量常常结合角色与类型来命名：

*Point startingPoint, centerPoint;*

*Name loginName;*

**所有的名字应该使用英语的写法。**

*fileName; // NOT: filNavn*

对国际开发，英语是优选语言。

**大范围内的变量应该使用长的名字，小范围内的变量一般使用短的名字。**

临时变量的名字最好保持短一些。

**对象的名字是隐含的，应该避免在方法名中出现。**

*line.getLength(); // NOT: line.getLineLength();*

* 1. **特殊命名约定**

**对属性的直接访问必须使用get/set术语。**

*employee.getName(); matrix.getElement (2, 4);*

*employee.setName (name); matrix.setElement (2, 4, value);*

一致的术语增强可读性。

**当一些东西被计算时，一般在方法中使用术语compute。**

*valueSet->computeAverage(); matrix->computeInverse()*

提示可能有时间的消耗，如重复使用，使用者可能会考虑缓存。一致的术语增强可读性。

**当一些东西被查找时，一般在方法中使用术语find。**

*vertex.findNearestVertex(); matrix.findMinElement();*

提示简单的查找，较小的计算消耗。一致的术语增强可读性。

**当一些东西对象或概念被确定时，一般在方法中使用术语initialize。**

*printer.initializeFontSet();*

避免init缩写。

**表示GUI构件的变量应该将构件的类型名作为后缀。**

*mainWindow, propertiesDialog, widthScale, loginText, leftScrollbar, mainForm, fileMenu, minLabel, exitButton, yesToggle etc.*

增强可读性。

**当名字表示一个对象链表时，应该加后缀List。**

*vertex (one vertex), vertexList (a list of vertices)*

增强可读性，名字提示使用者关于变量类型和对象提供可能的操作的线索。

一个list是复合数据类型，能够被正向、反向遍历等（典型的STL vector）。一个普通的数组很简单。将

Array后缀用于数组对象。

**表示对象的数量的变量，应该加前缀n。**

*nPoints, nLines*

作为表示对象数目的约定。

**表示实体数量的变量，应该加后缀No。**

*tableNo, employeeNo*

另一个方法是使用i前缀：iTable，iEmployee。这样更正式，但通用性小。

**用于迭代的变量应该使用i，j，k等。**

*for (int i = 0; i < nTables); i++) {*

*:*

*}*

*vector<MyClass>::iterator i;*

*for (i = list.begin(); i != list.end(); i++) {*

*Element element = \*i;*

*...*

*}*

作为表示循环的约定。

**布尔变量与方法应该使用is前缀。**

*isSet, isVisible, isFinished, isFound, isOpen*

使用is前缀解决了选择用如status或flag作不好的boolean名的问题，简单使用isStatus 或 isFlag是不合适的，应该使用更有意义的名字。

在某些情况下使用其他的前缀可能更合适。如：

*bool hasLicense();*

*bool canEvaluate();*

*bool shouldSort();*

**对相反的操作必须使用相反的名字。**

*get/set, add/remove, create/destroy, start/stop, insert/delete, increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide, suspend/resume, etc.*

通过对称降低复杂性。

**应该避免名字的缩写。**

*computeAverage(); // NOT: compAvg();*

考虑两种单词。一是在语言字典中的一般单词，必须不使用缩写。不用如下的缩写：

*cmd 代替 command*

*cp 代替 copy*

*pt 代替 point*

*comp 代替 compute*

*init 代替 initialize*

*etc.*

其次是领域专用缩写，这些领域专用缩写或术语非常明确，应该使用。不用如下写法：

*HypertextMarkupLanguage 代替 html*

*CentralProcessingUnit 代替 cpu*

*PriceEarningRatio 代替 pe*

*etc.*

**应该避免命名指针。**

*Line \*line; // NOT: Line \*pLine; or Line \*linePtr; etc.*

**必须避免否定的布尔变量名。**

*bool isError; // NOT: isNoError*

*bool isFound; // NOT: isNotFound*

因为当变量名作逻辑否定操作时会否定之否定，不好直接理解。如!isNotFound的意图不直接。

**枚举常数一般用一个通用类型名作前缀。**

*enum Color {*

*COLOR\_RED,*

*COLOR\_GREEN,*

*COLOR\_BLUE*

*};*

提供额外的信息如帮助查找声明、组织信息与表示的概念。

一种替代的方式是通过它们的公共类型引用常量，如：Color::RED, Airline::AIR\_FRANCE等。

**Exception类应该使用Exception作后缀。**

*class AccessException*

*{*

*:*

*}*

Exception类不是真正的程序主要设计部分，因此通过这种命名将它们与其他类区分开来。

**函数(返回一些东西的方法)命名应该遵循它们的返回；过程(void方法) 命名应该遵循它们的处理。**

增强可读性。这使得代码做什么很明确，而不是假设做所有事情。这同样使得代码保持清晰并减少副作用。

1. **文件**
   1. **源代码文件**

**文件的换行格式应该采用UNIX格式，而不采用windows格式；**

**unix：只有换行；windows：换行+回车**

**在(UltraEdit->高级->配置->常规)中可以设置**

**C++头文件应该有.h扩展。源文件一般有.cc(推荐)，.C，.c++或.cpp扩展。**

MyClass.cc, MyClass.h

这些均为C++标准接受的文件扩展名。

**类应该在头文件中声明，在源文件中定义，文件名与类名相匹配。**

MyClass.h, MyClass.cc

容易发现与给定类相关的文件。此约定在Java 中是强制的，很成功。当头文件中有多个类说明时，应该将主类放在前面，并且文件名和主类名一致！

**所有的定义应该在源文件中。**

*class MyClass*

*{*

*public:*

*int getValue () {return value\_;} // NO!*

*...*

*private:*

*int value\_;*

*}*

应该在头文件中定义接口，在源文件中实现它。当需要查找实现时，程序员总是知道在源文件中去找。明显的例外是inline函数必须在头文件中定义。

**文件的内容必须保持在80列内。**

80 列是一个通用的维数，能适应多种编辑器、终端、打印机与调试器。避免被断行而破坏可读性。

**特殊的字符如TAB与分页必须避免，代码缩进时使用4个空格代替TAB。**

这些字符在不同程序员和不同环境的编辑器、终端、打印机与调试器中表现不同。

**不完整而分几行的必须是明显的，最后一个分隔符不应该换行放在下一行的前面，应该放在最后。**

*totalSum = a + b + c +*

*d + e;*

*function (param1, param2,*

*param3);*

*setText ("Long line split"*

*"into two parts.");*

*for (tableNo = 0; tableNo < nTables;*

*tableNo += tableStep)*

超过80列的限制后分行。很难给出分行的绝对规则，上面的例子供参考。

一般的：

在逗号后断行；

在操作符后断行；

新行与上一行的表达式对齐。

* 1. **Include文件与Include语句**

**头文件必须有防止包含多次的构造。作为惯例一般使用模块名，文件名和.h后缀的全部大写。**

*#ifndef MOD\_FILENAME\_H*

*#define MOD\_FILENAME\_H*

*:*

*#endif*

**include 语句应该有序并分组。顺序按照在系统中的层次位置，底层的在前。在include语句分组之间使用空行。**

*#include <fstream>*

*#include <iomanip>*

*#include <Xm/Xm.h>*

*#include <Xm/ToggleB.h>*

*#include "ui/PropertiesDialog.h"*

*#include "ui/MainWindow.h"*

**在cpp，cc等源文件中应该只包含一个和自己类名相同的头文件，其他用到的头文件应该在头文件中包含。**

***#include “MyClass.h”***

***//#include <iomanip> //这两行应该写在MyClass.h中***

***//#include "ui/MainWindow.h"***

**Include语句必须只在文件的头部。**

1. **语句**
   1. **类型**

**只针对一个文件的局部类型一般在相应文件内部声明。**

强制的信息隐藏。

**类必须按照public，protected和private的顺序分几部分。每部分必须被显式的标识。没有用到的部分应该预留。**

按照 "most public first"的原则，仅仅使用类的人可以不阅读 protected/private部分。

**类型转换必须显式进行。不要依赖隐式的类型转换。**

*floatValue = static\_cast<float> (intValue); // YES!*

*floatValue = intValue; // NO!*

这样，程序员明确表示他知道进行了类型转换。

* 1. **变量**

**变量应该在它们声明的地方初始化。**

这确保变量在任何时间是合法的。有时变量不能在声明时被初始化为合法的值，如：

*int x, y, z;*

*getCenter (&x, &y, &z);*

这种情况下，保持变量未被初始化胜过给一个假冒的值。

**变量不要有双重含义。**

通过确保所有概念被唯一表示增强可读性，同时减少负作用错误。

**应该尽量少使用全局变量。**

在C++ 中全局量完全可以使用类进行封装，以保证避免名字空间的冲突问题。这同样适用于全局函数与静态量。

**类的变量应该不要声明为public。**

public变量与信息隐藏相矛盾，使用private变量与访问函数代替。一个例外是当类的本质是一个数据结构，没有行为（等效于C的结构）。在这种情况下，类的实例变量为public。

注意在C++中保留struct仅仅为兼容C，避免使用它们通过减少使用构造的数目来增加代码的可读性。使用类来代替。

**同一类型的相关的变量一般在一条语句中声明。**

不相关的变量不要在同一语句中声明。

float x, y, z;

float revenueJanuary, revenueFebruary, revenueMarch;

通过分组变量增强可读性。

**C++ 指针和引用的符号应该靠近变量名而非类型名。**

*float \*x; // NOT: float\* x;*

*int &y; // NOT: int& y;*

对于指针是一个变量的指针类型(float\* x)还是指针指向一个给定的变量类型(float \*x)是有争议的。重要的是第一种推荐方式事实上不能在以行语句中声明多个指针。如：float\* x, y, z; 等效于float \*x; float y; float z;

**const 关键字应该在类型名之前。**

*void f1 (const Widget \*v) // NOT: void f1 (Widget const \*v)*

没有好坏，这样更普遍。

**布尔型变量和指针之外，应该不用隐式的0值测试。**

*if (nLines != 0) // NOT: if (nLines)*

*if (value != 0.0) // NOT: if (value)*

编译器没有必须定义int和float 0的实现是二进制0。同样，使用显式的测试语句能够清楚的表明被测试的类型。同样，指针测试不要用隐式的0测试，如使用 if (line)代替if (line == 0)。

**变量应该在尽可能小的范围内声明。**

保持在一个小的范围内操作对象，这样容易控制变量的影响与边界影响。

* 1. **循环**

**必须只有控制循环的语句包含在for()的构造中。**

*sum = 0; // NOT: for (i = 0, sum = 0; i < 100; i++)*

*for (i = 0; i < 100; i++) // sum += value[i];*

*sum += value[i];*

增加可维护性与可读性。清晰循环的控制与循环体。

**循环变量应该在循环体前被初始化。**

*isDone = false; // NOT: bool isDone = false;*

*while (!isDone) { // :*

*: // while (!isDone) {*

*} // :*

*}*

**一般避免do-while循环。**

do-while 循环因为条件在底部，没有while 循环与for 循环可读性好。读者必须扫描整个循环体才明白循环范围。

另外do-while 循环不是必须的，任何do-while循环均可以用while 循环或for循环重写。减少这种机制可以增强可读性。

**在循环内应该避免使用break和continue。**

这些构造可以与goto 比较，仅仅用在必须的地方。

**应该使用while(true)表示无限循环，。**

*while (true) {*

*:*

*}*

*for (;;) { // NO!*

*:*

*}*

*while (1) { // NO!*

*:*

*}*

测试1既无必要也无意义。for (;;) 可读性不好。

* 1. **条件**

**必须避免非常复杂的条件表达式。通过引入临时的布尔变量代替。**

*if ((elementNo < 0) || (elementNo > maxElement)||*

*elementNo == lastElement) {*

*:*

*}*

应该替代如下：

*isFinished = (elementNo < 0) || (elementNo > maxElement);*

*isRepeatedEntry = elementNo == lastElement;*

*if (isFinished || isRepeatedEntry) {*

*:*

*}*

通过将表达式赋值给boolean 变量，程序能做到自文档化。这种方式易于阅读与调试。

**对于if语句，正常的分支应该放在if部分，异常部分放在else部分。**

*isError = readFile (fileName);*

*if (!isError) {*

*:*

*}*

*else {*

*:*

*}*

确保例外不要出现在正常执行路径。这对于可读性与性能很重要。

**条件应该放在单独的一行。**

*if (isDone) // NOT: if (isDone) doCleanup();*

*doCleanup();*

这用于调试目的，写成一行，难查看测试的结果为真为假。

**必须避免在条件部分包含执行语句。**

*// Bad!*

*if (!(fileHandle = open (fileName, "w"))) {*

*:*

*}*

*// Better!*

*fileHandle = open (fileName, "w");*

*if (!fileHandle) {*

*:*

*}*

在条件部分包含执行语句很难阅读，对C/C++ 新手尤其如此。

* 1. **杂项**

**应该在代码中避免出现神秘的数。除0和1以外的任何数均为神秘数，应该用命名常量代替。**

**建议通过方法来访问常量。**

**函数必须总是显式列出返回值。**

*int getValue() // NOT: getValue()*

*{*

*:*

*}*

在函数没有显式返回时，C++却省返回int，但程序员不应该依赖这个特征。

**检查函数所有参数输入的有效性；并对所调用函数的错误返回码要仔细、全面地处理。**

**这样能保证函数的稳定性，健壮性。**

**应该不用goto。**

goto 语句与结构化代码相矛盾。仅仅在证明必须使用时才使用。

**在不包含注释和空行的情况下，函数的规模尽量限制在200行以内。**

**减少函数本身或函数间的递归调用；**

**应该用”0”替代”NULL”。**

NULL是标准C库的一部分，但在C++中是过时的。

1. **布局与注释**
   1. **布局**

**基本的缩排为4。**

*for (i = 0; i < nElements; i++)*

*a[i] = 0;*

缩排为1不明显；缩排大于4当嵌套深时，代码南阅读且一行代码被分割的机会加大。2、3与4是较好的选择。

**块的缩排应该使用下面的例1(java默认编排方式)或例2的布局，但不可以使用例3。函数和类的块必须使用例2的布局。建议使用例2。**

*while (!done) {*

*doSomething();*

*done = moreToDo();*

*}*

*while (!done)*

*{*

*doSomething();*

*done = moreToDo();*

*}*

*while (!done)*

*{*

*doSomething();*

*done = moreToDo();*

*}*

例3额外的缩排并没有强调程序的逻辑结构，例1与例2很清晰。

**类的申明应该使用如下的形式：**

*class SomeClass : public BaseClass*

*{*

*public:*

*...*

*protected:*

*...*

*private:*

*...*

*}*

遵循块的通用规则。

**函数或过程的开始、结构的定义及循环，判断等语句中的代码都要采用缩进风格。**

**函数的申明应该使用如下的形式：**

*void someMethod()*

*{*

*...*

*}*

遵循块的通用规则。

**各类if-else语句应该使用如下的形式：**

*if (condition)*

*{*

*statements;*

*}*

*if (condition)*

*{*

*statements;*

*}*

*else*

*{*

*statements;*

*}*

*if (condition)*

*{*

*statements;*

*}*

*else if (condition)*

*{*

*statements;*

*}*

*else*

*{*

*statements;*

*}*

下面部分遵循块的通用规则，但是else语句与前一个if或else语句的右括号在同一行：

*if (condition)*

*{*

*statements;*

*}*

*else*

*{*

*statements;*

*}*

这等效于Sun的推荐。这种方法的选择考虑是更好的方式将每部分if-else写在文件的多行内。这使得操纵语句更容易，如移动else周围的内容。

**for 语句应该使用如下的形式：**

*for (initialization; condition; update)*

*{*

*statements;*

*}*

遵循块的通用规则。

**空的for语句应该使用如下的形式：**

*for (initialization; condition; update)*

*;*

强调for语句是空的，显式表明意图。然而，空循环应避免。

**while 语句应该使用如下的形式：**

*while (condition)*

*{*

*statements;*

*}*

遵循块的通用规则。

**do-while 语句应该使用如下的形式：**

*do*

*{*

*statements;*

*} while (condition);*

遵循块的通用规则。

**switch语句应该使用如下的形式：**

*switch (condition)*

*{*

*case ABC :*

*statements;*

*// Fallthrough*

*case DEF :*

*statements;*

*break;*

*case XYZ :*

*statements;*

*break;*

*default :*

*statements;*

*break;*

*}*

每个case关键字的缩进与switch语句相关作为一个整体，这样突出显示switch语句。注意在每个：之前使用格外的空白符。当对应的case语句没有break语句时，需要显式的注释说明，防止因为无意漏加。

**try-catch语句应该使用如下的形式：**

*try*

*{*

*statements;*

*}*

*catch (Exception exception)*

*{*

*statements;*

}

部分遵循通用块的规则。对于if-else语句使用的闭括号规则同样适用于try-catch语句。

**单一的if-else，for或while语句一般可以不用大扩号。**

*if (condition)*

*statement;*

*while (condition)*

*statement;*

*for (initialization; condition; update)*

*statement;*

通用的推荐(包括Sun Java推荐在内) 这种情况总是使用大括号。但是，大括号作为语言机制是用于分组多条语句的，对单一语句使用是多余的。

* 1. **空白符**

**一般操作的前后加一个空白符。**

**C++的保留字后需跟随一个空白符。**

**逗号后应该跟随一个空白符。**

**冒号的前后加一个空白符**

**在for语句中的分号后加一个空白符。**

*a = (b + c) \* d; // NOT: a=(b+c)\*d*

*while (true) { // NOT: while(true) ...*

*doSomething (a, b, c, d); // NOT: doSomething (a,b,c,d);*

*case 100 : // NOT: case 100:*

*for (i = 0; i < 10; i++) { // NOT: for (i=0;i<10;i++){*

突出显示语句的成份，增强可读性。很难给出在C++代码中使用空白符的完整列表。上面的例子给出这种努力的通用想法。

**方法名后如有其他名字，应该紧随一个空白符。**

*doSomething (currentFile); // NOT: doSomething(currentFile);*

突出显示个体的名字，增强可读性。

一种替代的方法是在括号后跟随一个空白符，如doSomething( currentFile )；这突出了名字的意图，但是，后面的括号是多余的，如去掉，空白语句不对称(doSomething( currentFile);)。

**相对独立的程序块之间，变量说明之后必须加空白行分隔。**

**一个块内的逻辑单元应该使用1个空白行分隔。**

在块内的逻辑单元间使用空白将增强可读性。

**方法应该用3个空白行分隔。**

通过使用比方法内更多的空白，使得在文件中方法更加突出。

**变量在声明时应该左对齐。**

*AsciiFile \*file;*

*int nPoints;*

*float x, y;*

增强可读性。通过对齐很明显认出变量与类型。

**通过对齐增强可读性。**

*if (a == lowValue)*

*compueSomething();*

*else if (a == mediumValue)*

*computeSomethingElse();*

*else if (a == highValue)*

*computeSomethingElseYet();*

*value = (potential \* oilDensity) / constant1 +*

*(depth \* waterDensity) / constant2 +*

*(zCoordinateValue \* gasDensity) / constant3;*

*minPosition = computeDistance (min, x, y, z);*

*averagePosition = computeDistance (average, x, y, z);*

*switch (value)*

*{*

*case PHASE\_OIL : strcpy (string, "Oil"); break;*

*case PHASE\_WATER : strcpy (string, "Water"); break;*

*case PHASE\_GAS : strcpy (string, "Gas"); break;*

*}*

通过对齐可以增强程序的可读性，虽然有时会破坏通用的规则。很难给出统一的对齐标准，上面的例子可供参考。

* 1. **注释**

**不好的代码不应该试图通过注释解决，应该重写代码！**

一般的，代码应该通过适当的名字选择与显式的逻辑结构做到自身文档化，应该少使用注释来弥补差的代码。

对编码层文档的主要贡献不是注释，而是好的程序风格。风格包含好的程序结构，直接使用和易于理解的方法、好的变量名、好的子程序名、命名常量、清晰的布局和最小的控制流及灵活的数据结构。

**所有的注释应该用英文写。由于目前的用户主要是国内的用户，故建议使用中文。**

**源文件头部应进行注释，列出：版权说明、版本号、生成日期、作者、模块目的功能，主要函数及其功能、修改日志等。**

**对所有需要注释的使用//注释符，包括多行注释。**

*// Comment spanning*

*// more than one line.*

既然C注释不支持多层，使用//注释确保当为了调试等原因时总能使用/\* \*/在文件中注释一整块。

在“//“与实际注释间适用一个空白符，注释以大写字母开始以句号结束。

**注释的位置应该与代码相关。注释的缩进应该和代码同步。**

*while (true) // NOT: while (true) {*

*{*

*// Do something // // Do something*

*something(); // something();*

*} // }*

这样避免注释破坏了程序的逻辑结构。