网络聊天室项目开发规范

文档历史

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 描述 | 作者 | 日期 |
| V0.1 | 格式建立初稿完成 | 冯致远 | 2018.06.04 |
|  |  |  |  |
|  |  |  |  |

网络聊天室开发规范

Contents

[1 文档编写目的 4](#_Toc515894234)

[2 命名规则 4](#_Toc515894235)

[2.1 工程命名 4](#_Toc515894236)

[2.2 类的命名 4](#_Toc515894237)

[2.3 函数命名 5](#_Toc515894238)

[2.4 变量的命名 5](#_Toc515894239)

[3 注释规则 7](#_Toc515894240)

[4 代码格式化 9](#_Toc515894241)

[4.1 类（class）的书写顺序 9](#_Toc515894242)

[4.2 对齐方式 10](#_Toc515894243)

[4.3 空行空格使用 10](#_Toc515894244)

[5 组织结构 11](#_Toc515894245)

[5.1 组织通则 11](#_Toc515894246)

[5.2 头文件 12](#_Toc515894247)

[5.3 源文件 12](#_Toc515894248)

[5.4 类 13](#_Toc515894249)

# 1 文档编写目的

为了保证应用程序的结构和代码风格标准化，使网络聊天项目其他成员可以协同开发或共享开发成果，特制定《网络聊天室C++开发规范》。本规范注重于代码的物理结构和外观，而不是代码的逻辑结构，使得代码更加容易阅读、维护、管理以及修订。

本规范于2018年6月由冯致远起草制定，经团队讨论后，于同月实施。从实施之日起，所有团队中新增及修订的C++语言源代码都必须遵从本规范进行书写。需要对本规范进行修订增删时，须由团队进行集体讨论，先达成共识，再进行修改。

# 2 命名规则

通则：

1. 所有命名都应使用标准的英文单词或缩写，不得使用拼音或拼音缩写，除非该名字描述的是中文特有的内容，如半角、全角，声母、韵母等。

2. 所有命名都应遵循达意原则，即名称应含义清晰、明确。

3. 所有命名都不宜过长，应控制在规定的最大长度以内。

4. 命名中，每个单词的第一个字母应该大写，单词与单词之间直接连接，用大写字母加以区别。

5. 所有命名都应尽量使用全称，如果使用缩写，则应该使用《通用缩写表》(https://blog.csdn.net/z\_xyin/article/details/46603631)中的缩写。原则上不推荐使用《通用缩写表》以外的缩写，如果使用，则必须对其进行注释和说明。

6. 命名的长度应当符合“min-length && max-information”原则。一般来说，长名字能够更好地表达含义。单字符的名字也是有用的，常见如i、j、k、n、x、y、z等，它们通常可用作函数内的局部变量。

## 2.1 工程命名

工程项目的意义名称根据团队讨论决定，在此工程意义名称的前面添加大写的“TG”（Tarena Group）作为此工程项目的工程命名。如，工程项目的意义名称为Model，工程名称为TGModel。

## 2.2 类的命名

类的定义以大写’CD’开头，例如 ‘CDStudentInfo’；类的对象以大写’O’开头，例如

‘OStudentInfo’

类在作为函数参数传递时，以小写’c’开头，例如 CStudentInfo &cStudentInfo；全局类的对象全部大写，例如’OINPUT’

## 2.3 函数命名

函数的命名必须符合：动词 [+ 名词] 的原则，类的成员函数也可以只使用“动词”，被省略掉的名词就是对象本身。如：（驼峰命名）

void GetWidth(double \*Width)

## 2.4 变量的命名

变量限定词

|  |  |  |
| --- | --- | --- |
| 限定词 | 说明 | 例子 |
| 无 | 局部变量 |  |
| m\_ | 类私有成员变量 | int m\_Width |
| p\_ | 类公有成员变量 | int p\_Name |
| s\_ | 静态变量 | static int s\_Num |
| g\_ | 全局变量 | int g\_HowManyStu |
| sg\_ | 静态全局变量 |  |

变量类型前缀

|  |  |
| --- | --- |
| 前缀 | 类型 |
| b | Bool |
| ch | char |
| s | string |
| v | vector |
| n | int |
| u | unsigned |
| l | Long |
| ll | Long long |
| ul | Unsigned long |
| d | double |
| f | float |
| p | point |
| fp | \*File |
| e | enum |
| st | struct |
| set | set |
| uni | union |
| w | WORD |
| dw | DWORD |

- 禁止使用单字节命名变量，但允许定义i，j，k作为局部循环变量

- 使用名词或者形容词＋名词方式命名变量

- 一个变量只有一个功能，不能把一个变量用作多种用途

- 防止局部变量与全局变量同名

- 严禁使用未经初始化的变量作为右值

- 在首次使用前初始化变量，初始化的地方离使用的地方越近越好，类的成员变量可以在构造函数中初始化的一定要初始化

- 变量的命名应该遵循即：[限定词+’\_’ +]类型前缀+意义名词

- 最终的变量名总长不得超过32个英文字符

宏和常量

~~~~~~~~~

- 不允许直接使用魔鬼数字（即意义不明的数字），应为此类数字添加宏定义或是使用常量，例如：

#define PI 3.1415926

const double ELEC\_CHARGE = 1.602e-19

- 对于数值或者字符串等等常量或宏的定义，建议采用全大写字母，单词之间加下划线 `\_` 的方式命名（枚举同样建议使用此方式定义）

- 除了头文件或编译开关等特殊标识定义，宏定义不能使用下划线 `\_` 开头和结尾

局部变量

~~~~~~~~

- 局部变量不必要加限定词，即：类型前缀+意义名词。例如: int nImageNumber，其中 ‘n’表示此变量int型，’ImageNumber’表示此变量的意义。

类的成员变量

~~~~~~~~~~~~

- 类中的成员变量命名的限定词为字母’m’和’p’，所以类中的成员变量命名必须以小写字母’m\_’或’p\_’开始。例如：int m\_nImageNumber中，’m’表示类中私有变量，‘n’表示此变量为int型， ‘ImageNumber’表示此变量的意义。

静态变量

~~~~~~~~

- 对于类中的成员静态变量命名必须以小写字母’ms\_’开始。例如：

int ms\_nImageNumber中，’ms’表示类中成员静态变量，’n’表示此变量为int型，’ImageNumber’表示此变量的意义。

- 对于局部的静态变量命名必须以小写字母 ‘s\_’开始。例如 int s\_nImageNumber，其中’s’表示静态变量，’n’表示此变量为int型， ‘ImageNumber’表示此变量的意义。

参数

~~~~

- 参数的命名和局部变量的命名相同，即：类型缩写+意义名词。例如：

GetImageNumber(int &nImageNumber) 之中，’n’表示参数为int型， ‘ImageNumber’表示参数的意义。

枚举

~~~~

- 枚举类型的命名中，单词与单词之间直接连接，用大写字母加以区别，并且加 `e` 作为前缀。

- 定义枚举时，每个元素单独占一行，且按从小到大定义，元素和赋值等号分别对齐

- 枚举元素所有字符大写，单词间用下划线隔开.

enum eClockDirecter{CLOCKWISE = 1, ANTICLOCKWISE = -1};

联合

~~~~

- 联合类型的命名中，单词与单词之间直接连接，用大写字母加以区别，并且加`uni` 作为前缀。联合包含的类型数据的命名必须遵循局部变量的命名规则。

结构体

~~~~~~

- 结构类型的命名中，单词与单词之间直接连接，用大写字母加以区别，并且加`st` 作为前缀。结构包含的类型数据的命名必须遵循局部变量的命名规则。

struct stPerson // Declare struct type

{

int nAge; // Declare member types

float fWeight;

}

# 3 注释规则

使用代码注释的目的主要有：

1. 用文字说明代码的作用（即为什么编写该代码，而不是如何编写）；

2. 明确指出该代码的编写思路和逻辑方法；

3. 使人们注意到代码中的重要转折点；

4. 使代码的阅读者不必在他们的头脑中仿真运行代码的执行过程。需要注意的是，空行和空白字符也是一种特殊的注释。注释可以与语句在同一行，也可以在上行，禁止在下面注释。规定在所有的注释中都以“//”开始，“/\*”和“\*/”之间的代码仅表示此段代码暂时不用。在注释中所有的标示符都必须用窄字符。（此处可以讨论一下）

--------

注释的关键思想是：注释的目的是尽可能的帮助读者了解的信息和作者一样多。

---------

什么不需要注释

~~~~~~~~~~~~~~~~~~~~

- 不要为那些从代码本身就能快速推断的事实写注释

- 不要为了注释而注释

- 不给不好的名字加注释，先修改名字

- 好代码 > 坏代码 + 好注释

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

什么需要注释-记录你的思想

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

一个好的方法帮助你添加好的注释是，站在读者的角度上：

- 意料之中的提问

- 公布可能的陷阱

- 全局观注释：类之间如何交互，复杂算法的参考资料，算法实现思路

- 总结性注释

~~~~~~~~~~~~~~~~~~

言简意赅的注释

~~~~~~~~~~~~~~~~~~

- 格式紧凑

- 避免不明确的代词

- 润色粗糙的句子

- 精确描述函数行为

- 用输入/输出的例子来说明特别的情况

- 注释高层次的代码，而非明显的细节

- 嵌入式注释可以很好的注释函数的参数

注释的两种方式如下：

\* 简单的说明，必须与代码同一行或上行，用“//”开始，例如：

if(fValue > 0) // if value greater than 0. execute ...

\* 详细的说明，必须在代码上方，用“//@Begin Description:”标示开始进行说明，用“//@End Description”标示结束。例如：

//@Begin Description:

//if value is greater than 0. execute ...

//@End Description

if(value > 0)

{

......

}

(注释规则也可以再讨论确定最后形式)

~~~~~~~~~~~~~~~~~~

函数的注释

~~~~~~~~~~~~~~~~~~~

函数的简单说明应该放在头文件中，详细的说明应放在CPP文件中。在头文件(H文件)中的注释模板为：

//@Begin Description:

//Get the bits depth of the tray image. the default value

//is 8;

@End Description

size\_t GetBits( ) const;

或者为：

size\_t GetBits( ) const; // Get the bits depth

在CPP文件中，函数的注释模板为(放在文件头)：

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//@Function Name:

//@Parameter [in] : int a—

//@Parameter [out] : int b—

//@Parameter [in,out] : int c—

//@Description:

//@Return Value:

// 为1，表示：

// 为0，表示：

//@Example: Demo about the function

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 4 代码格式化

基本要求如下：

\* 程序结构清晰，简单易懂，单个函数的程序函数推荐在60行以内。

\* 循环、分支层次不要超过五层。

\* 用 IF 语句来强调只执行两组语句中的一组，禁止 ELSE GOTO 和 ELSERETURN。

\* 用 CASE 实现多路分支。

\* 函数只有一个出口。

## 4.1 类（class）的书写顺序

规定类(CLASS)中内容的书写顺序如下：

Public:

1. Type Define 类型定义

Public:

2. Constructor Function

3. Destroy Function

Public:

4. Property or Data Member

Public:

5. SetParalmeter Method

6. Execute Method

7. GetPalameter Mehod

Protected:

8. Type Define 类型定义

Protected:

9. Data Member

Protected:

10. Method

Privated:

11. Type Define 类型定义

Privated:

12. Data Member

Privated:

13. Method

## 4.2 对齐方式

* 在软件中，所有的对齐都用TAB进行对齐（虽然TAB=4个空格，但是为了保证一致，不采用空格）
* 所有的条件编译宏都必须靠最左边。
* 程序的分界符“{”、“}”应独占一行并且位于同一列，同时与引用它们的语句对齐。{ }之内的代码块在“{”右边一个TAB处左对齐。
* 循环体必须另起一行，for、while、do等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加{ }。
* if语句独占一行，执行语句不得紧跟其后。不论执行语句有多少都需要加{}，并且if要和其后程序体的“{”、“}”和“else”在同一列。

## 4.3 空行空格使用

程序中，空行起着分隔程序段落的作用。空行得体将使程序的布局更加清晰。

* 首先，在每个类声明、每个函数定义结束之后都要加空行
* 在一个函数体内，逻辑上密切相关的语句之间不加空行，其他地方应加空行分隔。
* 函数名之后不要留空格，紧跟左括号“(”，以与关键字区别。
* “(”向后紧跟，“)”、“,”、“;”向前紧跟，紧跟处不留空格。
* “,”之后要留空格，例如 Function(x, y, z)。如果“;”不是一行的结束符号，其后要留空格，例如 for (initialisation; condition; update)。
* 赋值操作符、比较操作符、算术操作符、逻辑或位等二元操作符的前后都应当加空格。
* 一元操作符与操作数之间不加空格
* 指针或引用 `\*` 和 & 应紧跟变量名称或数据类型（仅一侧有空格），如

x = \*p;

p = &x;

char \*c;

char\* d;

const string& str;

for (int i = 1; i < N; ++i)

{

Compute(A, &B, C);

}

Example：

void Func1(int x, int y, int z); // 良好的风格

void Func1 (int x,int y,int z); // 不良的风格

if (year >= 2000) // 良好的风格

if(year>=2000) // 不良的风格

if ((a >= b) && (c <= d)) // 良好的风格

if(a>=b&&c<=d) //不良的风格

switch (Value) // 良好的风格

switch(Value) //不良的风格

while (Value) // 良好的风格

while(Value) //不良的风格

for (i = 0; i < 10; i++) // 良好的风格

for(i=0;i<10;i++) // 不良的风格

x = a < b ? a : b; // 良好的风格

x=a<b?a:b; // 不良的风格

int \*x = &y; // 良好的风格

int \* x = & y; // 不良的风格

# 5 组织结构

## 5.1 组织通则

- 1个.h文件定义1个类，或者功能相近的1组类（组合形式被一个类所使用）

- 1个.cpp文件中定义1个函数，或者功能相近的1组函数

- 尽量避免头文件互相调用（环状结构）及前置声明

## 5.2 头文件

头文件按照如下顺序组织，如果没有对应项则可以省略。

- 文件头

- 防止重复引用的设置

- #include部分

+ 引用标准库的头文件

+ 引用当前工程中的头文件

- 宏定义

- 常量声明

- 枚举声明

- 结构声明

- 类声明

- 内联函数定义

- 模板函数定义

- 全局变量声明

- 全局函数声明

- 本地变量声明

- 本地函数声明

## 5.3 源文件

------------

源文件按照如下顺序组织，如果没有对应项则可以省略。

- 文件头

- #include部分

+ 引用相关的头文件

+ 引用仅用于实现的头文件

- 宏定义

- 常量定义

- 类定义（C++）

- 外部变量引用

- 外部函数引用

- 全局变量定义

- 全局函数定义

- 本地变量定义

- 本地函数定义

## 5.4 类

--------

类的定义按照如下顺序组织，如果没有对应项则可以省略。

- 类型定义

- 构造、析构、初始化

- 虚函数

- 公用方法

- 公用静态方法

- 公有变量

- 私有方法

- 私有静态方法

- 私有变量