XEngine网络通信框架

代码规范标准文档

目录

[一 概述 4](#_Toc16146)

[1.1 编写目的 4](#_Toc20523)

[1.2 观看对象 4](#_Toc20304)

[1.2 联系方式 4](#_Toc16645)

[1.2.1 BUG提交 4](#_Toc19243)

[1.2.2 意见建议 4](#_Toc18648)

[1.2.3 商业合作 4](#_Toc3493)

[1.2.4 联系我们 4](#_Toc3890)

[1.3 阅读准备 4](#_Toc15963)

[1.4 开发模式 5](#_Toc27232)

[二 项目格式 5](#_Toc31492)

[2.1 目录格式 5](#_Toc1900)

[2.2 文件格式 5](#_Toc29164)

[2.3 创建项目 6](#_Toc25398)

[三 开发要求 6](#_Toc7466)

[3.1 编码要求 6](#_Toc9546)

[3.1.1 定义与实现 6](#_Toc3235)

[3.1.2 导出与导入 6](#_Toc12890)

[3.1.3 条件与判断 7](#_Toc23839)

[3.1.3 变量与函数 7](#_Toc15234)

[3.2 代码风格 9](#_Toc11507)

[3.3 内存申请 9](#_Toc30917)

[3.4 错误定义 9](#_Toc8974)

[3.5 文件编码 10](#_Toc28774)

[3.6 标准要求 10](#_Toc881)

[四 编译要求 10](#_Toc31239)

[4.1 Makefile 10](#_Toc25684)

[4.2 Visual Studio 11](#_Toc25519)

[五 调试要求 11](#_Toc29064)

[六 跨平台 11](#_Toc4143)

[附录 11](#_Toc22487)

[附录1 错误定义 11](#_Toc24954)

[1.XEngine\_BaseLib 11](#_Toc110)

[2.XEngine\_Core 12](#_Toc5867)

[3.XEngine\_Client 12](#_Toc17561)

[4.XEngine\_SystemSdk 12](#_Toc18064)

[5.XEngine\_HelpComponents 12](#_Toc17494)

[6.XEngine\_NetHelp 12](#_Toc4592)

[7.XEngine\_RfcComponents 13](#_Toc7821)

[8.XEngine\_AVCodec 13](#_Toc31210)

[9.XEngine\_StreamMedia 13](#_Toc32744)

|  |  |  |
| --- | --- | --- |
| 文件状态：  [ ] 草稿  [√] 正式发布 | 文件名称： | XEngine 代码规范标准指导文档 |
| 当前版本： | V9.10.0.1001 |
| 完成日期： | 2025-01-10 |
| 拟 制： qyt | | |

# 一 概述

## 1.1 编写目的

提供对XEngine开发人员使用,让开发人员能够了解本引擎所使用的开源库和环境,搭建一个可以二次开发的系统环境,方便开发人员快速上手。

开发人员必须严格按照此文档编写XEngine模块代码.

## 1.2 观看对象

开发人员

开发主管

代码质量检查相关人员

## 1.2 联系方式

### 1.2.1 BUG提交

你可以通过发送电子邮件的方式或者论坛提交给我们反驳BUG。Email：486179@qq.com 或者论坛 bbs.xyry.org,感谢大家的支持。

### 1.2.2 意见建议

如果有任何意见和建议都可以通过EMAIL方式发送给我们，电子邮件地址可以使用上面的地址提交即可。

当然你也可以直接与我们交流引擎方面的意见和好的想法。QQ:486179

### 1.2.3 商业合作

如果有公司或者个人想进行商业合作的，可以联系我们

### 1.2.4 联系我们

1. 网站:http://www.xyry.org
2. 论坛:http://bbs.xyry.org
3. QQ:486179
4. 邮件:486179@qq.com

## 1.3 阅读准备

阅读此文档之前,你需要先阅读开发必读文档与环境搭建文档.

## 1.4 开发模式

WINDOWS:使用VS2022开发,编译,调试

LINUX:使用MAKEFILE脚本编译,GDB调试,VS编写代码

MacOS:使用Makefile脚本编译,lldb调试,VS编写代码

Android:使用makefile编译,vs编写代码,需要Ndk环境

IOS:使用Makefile脚本编译,vs编写代码,需要mac自身的开发工具

# 二 项目格式

## 2.1 目录格式

注意:目录名称为大小写敏感..

如果要创建一个新的模块,必须在相应的组件目录下.如果这个组件目录不存在,那么你需要新建一个组件目录

比如,你要创建的模块是操作系统驱动级.那么可以新建一个XEngine\_Drive的目录.然后根据你的模块功能来创建一个模块项目.比如,你的模块提供的是操作网络驱动的功能,那么名称可以命名为:XEngine\_NetDrive

## 2.2 文件格式

每个项目都需要提供固定的文件格式.比如导出定义文件\*\_Define.h和错误定义文件\*\_Error.h

为了让代码跨平台.你还需要提供一个公用头文件pch.h这个头文件里面包含了各种需要加载的头文件.LINUX和MACOS只需要加载这个头文件和自身代码定义头文件即可.而WINDOWS的VS这个头文件是预编译头文件.

为了方便以后维护,每个功能模块的功能都需要分类.比如.一个网络模块可能提供各种网络服务.如TCP,UDP,那么需要为他们创建各自的文件夹.并且还可以区分TCP SELECT或者UDP SELECT,那么还需要在他们各自文件夹下面创建XCore\_TCPSelect.h和XCore\_TCPSelect.cpp文件.这个代表的是tcpselect服务器,如果是客户端,可以使用XClient\_TCPSelect.\*来表示.

## 2.3 创建项目

一个项目的创建需要用VS来创建,创建的项目文件需要添加到GIT中提交.他拥有以下公用必须的文件

* \*\_Define.h : 导出定义
* \*\_Error.h : 导出错误
* pch.h : 公用头文件
* pch.cpp : 导出实现
* \*\_BTorrent.def : 模块定义文件
* Makefile: 编译脚本

以及其他项目文件

# 三 开发要求

## 3.1 编码要求

### 3.1.1 定义与实现

每个功能的实现,都需要使用.h和.cpp来实现.而要实现这个功能,需要在.h里面申请函数.并且在CPP中实现它.

注意:所有定义都采用类包装的方式来实现.

需要公开或者导出的函数采用public

不需要公函的函数使用protect

不需要公开的变量使用private

为每一种类型都添加一次权限定义,比如内部函数和内部线程可以使用两次protect.也比如结构体变量和一般的变量(int string)等分别使用private 隔开.

### 3.1.2 导出与导入

实现了功能类,我们在外层\*\_Define.h中使用c模式导出函数.导出的函数名称必须采用C语言风格

模块导出的函数不能使用c++的任何特性和模式.比如导出类.STL等都不允许.

具体可以参考我们以前的代码.你必须按照规定的格式写代码,不然你的代码会被拒绝合并.

### 3.1.3 条件与判断

判断条件,需要完整语句,要求常量定义在前,可变类型定义在后,比如.判断语句需要换行和空格对其

if (1 == n)

{

Todo...

}

Switch (enum)

{

Case 1:

Break;

Case 2:

break

}

### 3.1.3 变量与函数

#### 3.1.3.1 变量

变量的使用和定义必须初始化.初始化都在类的构造函数里面,通过析构函数释放.如果需要的话.你也可以使用新的C++特性在类的声明中初始化,比如

class

{

int a = 0

}

变量的定义名称为固定格式.采用统一的风格,比如成员变量使用m\_开头,根据变量类型定义后面的字符,比如整数型m\_n 字符m\_c.字符串 m\_Str.根据提供的功能定义最后的名称,比如XNETHANDLE m\_xhThreads.表示这是一个内部使用的线程池句柄.

这个是内部定义的变量,如果是函数输入输出那么直接xhThreads即可,不需要加m\_了.如果是输出,那么需要指定pxhThreads;如果是导出函数指针的指针,那么使用ppxhThreads.或者整数型pInt\_和ppInt\_.字符串ptsz psz等.

#### 3.1.3.2 结构体与容器

结构体的参数一般都使用指针的方式作为输入和输出.比如pSt\_UserInfo.一个结构体的前面必须是pSt\_开头.如果不是指针就是用st\_开头,比如st\_UserInfo

容器虽然不能作为导出函数的参数使用,但是可以在内部使用.也就是你可以在内部函数中作为参数使用,容器参数使用方式同结构体,指针使用pStl\_开头,跟上容器类型,比如list,使用 pStl\_ListUser,映射类型pStl\_MapUser.非指针使用stl\_ListUser,stl\_MapUser.

请注意:框架内部所有结构提和类成员不允许有重复的名称出现,无论是否是在同一个模块或者项目中.

#### 3.1.3.3 函数

函数的使用和定义格式需要按照统一风格,特别是public的函数,比如我们的一个提供tcpselect服务的模块,在XEngine\_Core目录下.那么我们可以使用下面的方式定义函数名称.

XCore\_TCPSelect\_Start 表示启动一个tcpselect服务器.

函数名称命名规则需要根据实际情况定义,一般情况下,函数名称必须有三段.

第一段:表示模块或者组件名称,当模块内部区分多的时候可以直接使用模块名称,如果不足够,使用组件名称代替.

第二段:模块实现的功能分类,比如TCPSelect,也就是模块下创建的分类目录.

第三段:这个函数实现的功能.

#### 3.1.3.4 类型

每个可跨平台的自定义定义类型应该使用typedef 或者 define定义.比如跨平台函数定义XEngine\_Types.h,你需要按照次文件中的类型定义来处理.我们一般以VS中的跨平台代码为主,添加自己的类型字符比如x,位置也需要按照定义来添加.

变量类型自定义可以参考XEngine\_CommHdr.h,根据简化与可读性要求.有语言自身定义的跨平台变量类型不在重复定义.比如,bool,true和false等.

#### 3.1.3.5 参数

输入参数和输出参数.当一个函数拥有输入和输出参数的时候,需要遵守规则进行参数的定义.当输入参数是固定的时候,那么输入参数在前,如果输出参数是固定的时候,那么输出参数在前

参数的输入输出位置根据当前功能类来确定,比如一个解析器,需要解析缓冲区的参数,并且导出相对应的数据,那么输入参数在前,输出在后.再比如一个封装器,需要对我们的自定义数据进行封装,那么输出参数在前,因为输出的缓冲区和大小是固定的.而输入的封装数据是不固定的参数

## 3.2 代码风格

允许使用C/C++.尽量使用C.

由于存在跨平台需求,在编写代码的时候可以使用一些跨平台的函数与类.比如C++11-20中的线程函数std::thread 智能指针std::shared\_ptr 读写锁shared\_mutex等等.

禁止使用c++的特性,比如:重载,友元,继承,派生等等...

代码风格的规定统一目标是为了后期维护方便.请严格按照规定规定编写代码

## 3.3 内存申请

如果要申请内存,那么内存申请要求如下

注意:这个操作不包含内存池,内存池的时候按照自己的写法申请和释放内存.

如果需要申请的内存是变量类型.比如.int char等,那么使用malloc和free来申请和释放内存.

如果需要申请的内存是结构体或者类.那么使用new和delete来申请和释放内存.

## 3.4 错误定义

错误定义全部都在\*\_Error.h相关头文件下面,你必须为你模块提供一个函数来获取错误定义.比如 \*\_GetLastError();

错误定义开头必须为ERROR.然后按照以下格式填写你的错误.

ERROR\_XENGINE(此字段不是必须)\_组件名\_模块名\_功能分类名\_子功能分类名(如果有的话)\_函数功能\_错误类型

也可以简写为ERROR\_XENGINE(此字段不是必须)\_组件名\_模块名\_错误类型,这样一个模块可以通用所有错误码.

定义后你需要注意定义错误码的值.值是有严格区分的.具体可以参考附加信息说明

错误值最高定义为0xFFFFFFFF.前两位为组件,如果不足够扩展一位.比如0x110.后一位代表模块顺序比如0x110A(后面为B,C,D,E,F)如果不足像后扩展.比如0x111A.这里扩展方式有所改编,一个组件模块都需要扩展.下一位模块分类所属0x110A1,表示第一个功能分类.然后是类的分类,比如一个目录下面很很多相似但是却不关联的功能类文件.TimeSpan,TimeTrigger.等0x110A11表示分类顺序.左后两位是错误函数每个函数为10.20.30区分.

如果使用简写错误表,那么不需要区分功能类型,直接按照编号顺序编写下去即可.比如\*\*\*01-09后,顺序加1为 10-\*\*

## 3.5 文件编码

我们使用的代码格式都统一使用VS创建管理,所有文件代码格式必须是UTF8 WITH BOM的格式.其他格式不被支持,请勿所用.

也就是说,UTF8 WITH BOM的代码字符集格式将支持所有平台编译.

## 3.6 标准要求

C:按照VS2022的统一要求设定,VS的C语言标准与其他编译器不一致,比较落后,所以按照VS的标准做统一

C++:目前最高支持C++20版本

C:目前最高支持C17版本

# 四 编译要求

## 4.1 Makefile

非Windows系统(包括Linux,MacOS,Android.IOS),都使用Makefile来编译,Makefile应该支持非Windows下的所有平台编译.

如果你创建了一个新的模块,需要你自己提供编译命令,编译方式采用g++编译.需要自己提供makefile文件在你的目录下,并且编辑源代码主目录下面的Makefile来添加你的编译项目文件.

不提供cmake模式

如果需要调试,可以使用gdb来调试

LINUX下的MAKEFILE文件默认可以使用make来编译,默认编译为带-g 的调试编译,也可以带参数RELEASE=1 表示不带调试的编译.

编译目标平台:默认为x86,如果需要arm64,可以设置ARCH=arm

## 4.2 Visual Studio

Windows平台统一采用VS来编译.

我们要求每个项目在WINDOWS中都可以用VS打开编译和调试.你可以使用XEngine.sln来打开他,这个项目文件包含了WINDOWS下所有可用的模块与组件.

# 五 调试要求

WINDOWS采用VS调试,LINUX采用GDB调试.Mac采用lldb

# 六 跨平台

跨平台我们有宏定义配置,如果某些函数参数在某些平台有区别,那么需要通过宏定义来判断和处理.

VC(Windows)平台:\_MSC\_BUILD

LINUX平台:\_\_UBUNTU\_\_ 或者 \_\_CENTOS\_\_ 分别代表LINUX的两个发行系统

MACOS平台:\_\_MACOS\_\_

ANDROID平台:\_\_ANDROID\_\_

IOS平台:\_\_IOS\_\_

# 附录

## 附录1 错误定义

错误定义格式为0x010A0010最高8位,前两位0x01为所属组件,后一位0位模块补位,在后一位A开始为模块所属,A-F,如果超过补位为1后继续A-F,依次类推.后4位模块内的错误定义,00为分类错误定义,00,10,20最后两位为函数错误定义

组件所属010可以忽略最开始的0写成10.比如下面都是这种

### 1.XEngine\_BaseLib

0x10\*\*\*\*\*

XEngine\_Algorithm:0x10A\*\*\*\*

XEngine\_BaseLib:0x10B\*\*\*\*

XEngine\_BaseSafe:0x10C\*\*\*\*

### 2.XEngine\_Core

0x20\*\*\*\*\*

XEngine\_Core:0x20A\*\*\*\*

XEngine\_Manage:0x20B\*\*\*\*

XEngine\_Cryption:0x20D\*\*\*\*

XEngine\_Protocol:0x20E\*\*\*\*

### 3.XEngine\_Client

0x30\*\*\*\*\*

XClient\_Socket:0x30A\*\*\*\*

XClient\_Stream:0x30C\*\*\*\*

XClient\_APIHelp:0x30D\*\*\*\*

### 4.XEngine\_SystemSdk

0x40

XEngine\_SystemApi:0x40A\*\*\*\*

XEngine\_ProcFile:0x40B\*\*\*\*

XEngine\_SystemConfig:0x40C\*\*\*\*

### 5.XEngine\_HelpComponents

0x50\*\*\*\*\*

HelpComponents\_BINPack:0x50B\*\*\*\*

HelpComponents\_Compress:0x50C\*\*\*\*

HelpComponents\_DataBase:0x50D\*\*\*\*

HelpComponents\_Packets:0x50E\*\*\*\*

HelpComponents\_XLog:0x50F\*\*\*\*

HelpComponents\_WBlackList:0x50A

### 6.XEngine\_NetHelp

0x60\*\*\*\*\*

NetHelp\_APIFlow:0x60A\*\*\*\*

NetHelp\_APIHelp:0x60B\*\*\*\*

NetHelp\_XSocket:0x60C\*\*\*\*

NetHelp\_APIAddr:0x60D\*\*\*\*

### 7.XEngine\_RfcComponents

0x70\*\*\*\*\*

RfcComponents\_HttpProtocol:0x70A\*\*\*\*

RfcComponents\_MQTTProtocol:0x70B\*\*\*\*

RfcComponents\_NatProtocol:0x70C\*\*\*\*

RfcComponents\_ProxyProtocol:0x70D\*\*\*\*

RfcComponents\_NTPProtocol:0x70E\*\*\*\*

RfcComponents\_SIPPorotocol:0x70F\*\*\*\*

RfcComponents\_SnmpProtocol:0x71A\*\*\*\*

RfcComponents\_SSDPProtocol:0x71B\*\*\*\*

RfcComponents\_WSProtocol:0x71C\*\*\*\*

### 8.XEngine\_AVCodec

0x80\*\*\*\*\*

XEngine\_AudioCodec:0x80A\*\*\*\*

XEngine\_AVCollect:0x80B\*\*\*\*

XEngine\_AVHelp:0x80C\*\*\*\*

XEngine\_AVFormat:0x80D\*\*\*\*

XEngine\_AVFilter:0x80E\*\*\*\*

XEngine\_VideoCodec:0x80F\*\*\*\*

XEngine\_AVFrame:0x81A\*\*\*\*

### 9.XEngine\_StreamMedia

0x90\*\*\*\*

StreamMedia\_HLSProtocol:0x90A\*\*\*\*

StreamMedia\_RTCPProtocol:0x90B\*\*\*\*

StreamMedia\_RTPProtocol:0x90C\*\*\*\*

StreamMedia\_RTSPProtocol:0x90D\*\*\*\*

StreamMedia\_SDPProtocol:0x90E\*\*\*\*

StreamMedia\_RTMPProtocol:0x90F\*\*\*\*

StreamMedia\_FLVProtocol:0x91A\*\*\*\*

StreamMedia\_MP4Protocol:0x91B\*\*\*\*