XEngine网络通信引擎

开发文档

目录

[一 概述 4](#_Toc8484)

[1.1 编写目的 4](#_Toc9813)

[1.2 观看对象 4](#_Toc25753)

[1.2 联系方式 4](#_Toc4916)

[1.2.1 BUG提交 4](#_Toc4698)

[1.2.2 意见建议 4](#_Toc24177)

[1.2.3 商业合作 4](#_Toc7749)

[1.2.4 联系我们 4](#_Toc12405)

[1.3 阅读准备 4](#_Toc2834)

[1.4 开发模式 5](#_Toc29988)

[二 项目格式 5](#_Toc222)

[2.1 目录格式 5](#_Toc6348)

[2.2 文件格式 5](#_Toc5345)

[2.3 创建项目 5](#_Toc13688)

[三 开发要求 6](#_Toc13275)

[3.1 编码要求 6](#_Toc31786)

[3.1.1 定义与实现 6](#_Toc27913)

[3.1.2 导出与导入 6](#_Toc16509)

[3.1.3 条件与判断 7](#_Toc2459)

[3.1.3 变量与函数 7](#_Toc704)

[3.2 代码风格 8](#_Toc32005)

[3.1 内存申请 8](#_Toc29582)

[3.2 错误定义 8](#_Toc23708)

[四 编译要求 9](#_Toc1381)

[4.1 LINUX 9](#_Toc16737)

[4.2 MacOS 9](#_Toc981)

[4.3 WINDOWS 9](#_Toc19693)

[五 调试要求 10](#_Toc9528)

[六 跨平台 10](#_Toc4815)

[6.1 注意事项 10](#_Toc29458)

[附录 10](#_Toc29158)

[附录1 错误定义 10](#_Toc1671)

[1 XEngine组件 10](#_Toc1400)

[4 XEngine\_AVCoder组件 10](#_Toc16791)

[5 XEngine\_Client组件 11](#_Toc21526)

[6 XEngine\_HelpComponents组件 11](#_Toc13040)

[7 XEngine\_Lib组件 11](#_Toc21136)

[8 XEngine\_NetHelp组件 11](#_Toc407)

[9 XEngine\_RfcComponents组件 11](#_Toc17277)

[10 XEngine\_StreamMedia组件 12](#_Toc26310)

[11 XEngine\_SystemSdk组件 12](#_Toc45)

|  |  |  |
| --- | --- | --- |
| 文件状态：  [ ] 草稿  [√] 正式发布 | 文件名称： | 网络通信引擎开发需求文档 |
| 当前版本： | V7.40 |
| 完成日期： | 2022-09-23 |
| 拟 制： qyt | | |

# 一 概述

## 1.1 编写目的

提供对网络通信开发人员使用,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

# 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\*\_CommHdr.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在预编译头文件下添加这个头文件.

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.Linux: linux编译脚本
* Makefile.Darwin:MacOS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.

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.

#### 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.

## 3.2 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.

但是禁止使用c++的特性,比如:重载,友元,继承等等...

## 3.1 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.

## 3.2 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-\*\*

# 四 编译要求

## 4.1 LINUX

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.

## 4.2 MacOS

MacOS系统也使用makefile 来编写,通过makefile的后缀名区分.要求和LINUX一样.

## 4.3 WINDOWS

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.

同LINUX,如果你需要提供windows的编译,你只需要把VS项目配置好即可.

# 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

需要注意和重新开发的事项

## 6.1 注意事项

XEngine\_NetXApi:NetXApi\_Socket\_GetProtocolStatics,目前仅支持WINDOWS

NetXApi\_Socket\_ProcessNet目前仅支持WINDOWS

# 附录

## 附录1 错误定义

错误定义格式为0x120A0010最高8位,前二位0x12为所属组件,后一位0为模块补位,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为模块内的错误定义00为分类错误定义,00,10,20最后两位为函数错误定义

### 1 XEngine组件

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

XEngine\_Core:0x10A\*\*\*\*

XEngine\_Manage:0x10B\*\*\*\*

XEngine\_NetXApi:0x10C\*\*\*\*

XEngine\_OPenSsl:0x10D\*\*\*\*

XEngine\_Protocol:0x10E\*\*\*\*

XEngine\_WBlackList:0x10F\*\*\*\*

### 4 XEngine\_AVCoder组件

0x40\*\*\*\*\*

XEngine\_AudioCoder:0x40A\*\*\*\*

XEngine\_AVCollect:0x40B\*\*\*\*

XEngine\_AVHelp:0x40C\*\*\*\*

XEngine\_VideoCoder:0x40D\*\*\*\*

XEngine\_AVPlayer:0x40E\*\*\*\*

XEngine\_AVPacket:0x40F\*\*\*\*

### 5 XEngine\_Client组件

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

XClient\_Socket:0x50A\*\*\*\*

XClient\_OPenSsl:0x50B\*\*\*\*

### 6 XEngine\_HelpComponents组件

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

HelpComponents\_BINPack:0x70A\*\*\*\*

HelpComponents\_Compress:0x70B\*\*\*\*

HelpComponents\_DataBase:0x70C\*\*\*\*

HelpComponents\_Packets:0x70D\*\*\*\*

HelpComponents\_Authorize:0x70E\*\*\*\*

HelpComponents\_XLog:0x70F\*\*\*\*

### 7 XEngine\_Lib组件

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

XEngine\_Algorithm:0x80A\*\*\*\*

XEngine\_BaseLib:0x80B\*\*\*\*

### 8 XEngine\_NetHelp组件

0x90\*\*\*\*\*

NetHelp\_APIHelp:0x90A\*\*\*\*

NetHelp\_StressTest:0x90B\*\*\*\*

### 9 XEngine\_RfcComponents组件

0x120\*\*\*\*\*

RfcComponents\_EmailClient:0x120A\*\*\*\*

RfcComponents\_ProxyProtocol:0x120B\*\*\*\*

RfcComponents\_SIPPorotocol:0x120C\*\*\*\*

RfcComponents\_WSProtocol:0x120D\*\*\*\*

RfcComponents\_SnmpProtocol:0x120E\*\*\*\*

RfcComponents\_UPNPProtocol:0x120F\*\*\*\*

RfcComponents\_HttpServer:0x120A\*\*\*\*

RfcComponents\_SDPProtocol:0x121B\*\*\*\*

RfcComponents\_NatClient:0x121C\*\*\*\*

### 10 XEngine\_StreamMedia组件

0x140\*\*\*\*

StreamMedia\_XClient:0x140A\*\*\*\*

StreamMedia\_HLSProtocol:0x140B\*\*\*\*

StreamMedia\_RTPProtocol:0x140C\*\*\*\*

StreamMedia\_RTSPProtocol:0x140D\*\*\*\*

StreamMedia\_RTCPProtocol:0x140E\*\*\*\*

### 11 XEngine\_SystemSdk组件

0x150

XEngine\_SystemApi:0x150A\*\*\*\*

XEngine\_ProcFile:0x150B\*\*\*\*