XEngine网络通信引擎协议文档

目录

[XEngine网络通信引擎协议文档 1](#_Toc31510)

[前言 4](#_Toc5167)

[阅读者 4](#_Toc27962)

[概述 4](#_Toc13061)

[协议定义 4](#_Toc29968)

[相关模块 4](#_Toc8730)

[一 协议规范 4](#_Toc31020)

[1.1 协议头说明 5](#_Toc29266)

[1.2 协议头格式 5](#_Toc254)

[1.2.1 普通协议头 5](#_Toc18194)

[1.2.2 扩展协议头 5](#_Toc15678)

[1.3 协议头字段 6](#_Toc25970)

[1.3.1 普通协议头字段 6](#_Toc24800)

[1.3.2 扩展协议头字段 6](#_Toc5607)

[1.4 协议头表示 7](#_Toc13966)

[1.5 协议展示 7](#_Toc13398)

[1.5.1 普通协议 7](#_Toc32499)

[1.5.2 扩展协议 7](#_Toc27059)

[二 使用协议 8](#_Toc18012)

[2.1 主协议 8](#_Toc9108)

[2.2 子协议 8](#_Toc5250)

[2.3 协议分类 8](#_Toc16738)

[三 系统协议 9](#_Toc12150)

[3.1 日志协议 9](#_Toc27662)

[3.2 验证协议 10](#_Toc13055)

[3.3 流媒体协议 10](#_Toc29810)

[3.4 存储服务协议 10](#_Toc2035)

[3.5 后台服务协议 10](#_Toc13775)

[3.6 P2XP协议 10](#_Toc25426)

[3.7 消息队列 10](#_Toc9464)

[3.8 消息分发 10](#_Toc29476)

[3.9 UDX协议 10](#_Toc14901)

[3.9.1 数据传输协议 11](#_Toc20337)

[3.9.2 登录协议 11](#_Toc4836)

[3.9.3 数据重传 12](#_Toc14280)

[3.9.4 通知协议 13](#_Toc20821)

[3.10 RPC协议 14](#_Toc10801)

[3.11 心跳服务 14](#_Toc13445)

[3.11.1 心跳同步 14](#_Toc1663)

[3.12 分包协议 15](#_Toc28315)

[3.12.1 分包开始 15](#_Toc18780)

[3.12.2 分包结束 16](#_Toc14416)

[3.13 离开协议 16](#_Toc1154)

[四 用户协议 17](#_Toc20657)

[4.1 用户协议规范 17](#_Toc9731)

[五 特别说明 17](#_Toc21469)

[5.1 协议头 17](#_Toc28304)

[5.1.1 加密类型 17](#_Toc26786)

[5.1.2 负载类型 17](#_Toc14101)

[5.2 权限级别 18](#_Toc1074)

[附录 18](#_Toc26668)

[附录1 类型定义 18](#_Toc15424)

[附录2 协议定义 18](#_Toc23089)

[附录3 转换定义 18](#_Toc9028)

|  |  |  |  |
| --- | --- | --- | --- |
| 文件状态：  [ ] 草稿  [√] 正式发布 | 文件名称： | XEngine网络通信引擎协议文档 | |
| 适用版本： | V7.31 | |
| 发布日期： | 2022-03-24 | |
| 拟 制： qyt | | |

# 前言

## 阅读者

开发人员,测试人员,产品经理,QA

阅读此篇文档之前,你应该首先阅读SDK开发文档(Readme)

## **概述**

此文档包含XEngine网络通信引擎现有协议解释说明.用户在开发基于XEngine网络通信引擎(XEngine)SDK的时候可以参考这个文档使用我们的网络通信协议!

## 协议定义

协议头定义文件都在我们提供的SDK头文件的主目录下,文件名为:XEngine\_Protocol.h文件.

## 相关模块

使用我们的协议,我们已经提供了一个模块供大家使用,这个模块在帮助组件下,名字为HelpComponents\_Packets.此模块可以快速帮助用户完成一个协议的解析和组装.

# 一 协议规范

在所有网络通信过程中,我们的服务都有一个自己的包头,采用的协议格式是包头+包体(STRUCT OR JSON)数据格式来通信.

以下0x 开头都代表16进程,当然,你可以转换成10进程发送,道理是一样的.

部分协议说明可能没有表示出所有协议字段,这种情况下不代表可以不填写或者不带其他协议头的字段.无论什么情况下,协议头必须带上,并且他的开始和结束标志必须填充.其他值你可以根据需要填写.

对齐方式:所有协议按照1个字节内存对其

字节序:协议头按照小端模式发送,无需网络字节序,如果你有特殊需求,那么你需要自己封装协议头,否则我们的协议头封装函数都不会转字节序.

## 协议头说明

## 1.2 协议头格式

我们提供的协议头有两种,一个是普通的协议头,用于对于安全与网络通信要求不高的情况,另外一种是扩展协议头,里面包含了很多详细的协议字段,用于对网络通信要求高的环境.

我们下面演示的协议头是通过C/C++方式来观看的,如果你的开发语言是其他语言,那么只需要按照大小填充内容即可

注意:普通协议和扩展协议头无法一起使用!

### 1.2.1 普通协议头

普通协议头,固定30个字节,一般情况下只适用于TCP

typedef struct tag\_XEngine\_ProtocolHdr

{

*WORD* wHeader; //协议头头部 固定的赋值

XNETHANDLE xhToken; //唯一标识符

*UINT* unOperatorType; //操作类型

*UINT* unOperatorCode; //操作码

*UINT* unPacketSize; //数据包大小，后续包的大小

*BYTE* byVersion; //协议版本

*BYTE* byIsReply; //是否需要回复包 0 否，1是

*SHOT* wReserve : 12; //自定义数据位或者保留

SHOT wCrypto : 4; //加解密标志位

*WORD* wPacketSerial; //包序列号

*WORD* wTail; //协议头尾部 固定的赋值

}XENGINE\_PROTOCOLHDR,\*LPXENGINE\_PROTOCOLHDR;

### 1.2.2 扩展协议头

扩展协议头,扩展协议头适用于TCP,UDP,扩展字段功能更完善,更适用于对通信要求严格的服务.请注意字段的占位标识符.

typedef struct tag\_XEngine\_ProtocolHdrEx

{

*WORD* wHeader : 8; //协议头头部 固定的赋值

*WORD* wVersion : 4; //协议头版本号标志

*WORD* wPayload : 4; //后续数据包负载类型

XNETHANDLE xhToken; //唯一标识符

XNETHANDLE xhXTPTime; //时间戳

*UINT* unOperatorType : 16; //操作类型

*UINT* unOperatorCode : 16; //操作码

*UINT* unPacketCrypt : 1; //加密标志,0没有加密,其他值表示加密

*UINT* unPacketCount : 7; //0不分包,> 0 分包个数

*UINT* unPacketSerial : 8; //包序列号,只有分包的时候这个值才有效，其他时候请填充0

*UINT* unPacketSize : 16; //数据包大小，后续包的大小，不包括协议头和协议尾

*WORD* wReserve : 8; //自定义数据位或者保留

*WORD* wIsReply : 8; //是否需要回复包 0 否，1是

}XENGINE\_PROTOCOLHDREX, \*LPXENGINE\_PROTOCOLHDREX;

扩展协议头需要协议尾,协议的尾部需要加到数据的末尾一起发送.格式是:协议头+数据+协议尾

typedef struct tag\_XEngine\_ProtocolTailEx

{

*WORD* wCheckSum : 8; //数据校验码,数据区校验

*WORD* wTail : 8; //协议头尾部 固定的赋值

}XENGINE\_PROTOCOLTAILEX, \*LPXENGINE\_PROTOCOLTAILEX;

## 1.3 协议头字段

### 1.3.1 普通协议头字段

* wHeader:协议头的包头字段,两个字节(unsigned short类型),采用固定填充XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER(0x11)
* xhToken:8个字节(unsigned long long),一般创建流或者文件传输的时候用到
* unOperatorType:操作类型,4个字节(unsigned int),代表协议头的类型
* unOperatorCode:操作码,4个字节,代表协议头类型的操作码.
* unPacketSize:后续包长度,,4个字节,不包括此包头.用于组包
* byVersion:协议版本,(unsigned char 类型)1个字节,1为后续包为结构体,2为后续包为JSON,3为其他,除了特别说明的地方以外,默认只支持版本1
* byIsReply:是否需要回复包,(unsigned char类型)1个字节,除非下面协议说明写了支持回复包设置,否则将为FALSE
* wReserve:保留字段,((short类型)12BIT位,用户可自行使用.
* wCrypto:4个BIT位,加解密标志,参考特殊说明
* wPacketSerial:包序列号,(unsigned short类型)两个字节,可为0,暂时不起作用
* wTail:协议头的包尾,(unsigned short类型)两个字节,采用固定值XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL(0xff)

### 1.3.2 扩展协议头字段

* unPacketCrypt:占用1位,表示是否使用了加密,0表示数据包没有加密>0表示中间的数据包是经过加密的,1-9由用户指定加密协议类型
* unPacketCount:占用7位.分包标志,0不分包,大于0,需要分包,分包个数最大为127个.
* wCheckSum:占用一个字节,校验和,用于计算数据载体的校验.可以单独调用函数进行计算
* wPayload:后续数据负载类型
* xhXTPTime:XEngine Time Protocol,是由64位整数组成,高32位为当前UTC -TIME数值,低32位为微秒数值

## 1.4 协议头表示

协议头里面包括后面的结构体字段,都可以通过你指定的大小来设置,不一定要用此方式或者此类型,在计算机里面只要他的取值返回和内存大小相同,都可以通用的.

## 1.5 协议展示

这里展示了一个协议在数据内存中表示的情况,开发者可以根据这条展示信息只管的了解到我们的协议使用的方式.下面展示的协议包含了普通协议和扩展协议,它适用于所有协议类型.

以下展示结果为主机字节序,也就是小端模式,以后不在说明.

### 1.5.1 普通协议

30字节头+数据(如果没有数据,那么只有30字节头)

不带数据的心跳包数据展示,回复请求设置为0,后续数据类型设置为0

0x11 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0A 0x00 0x00 0x00 0x01 0x0A 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF 0x00

带数据的网络验证协议登录数据展示,回复请求设置为1,后续数据类型设置为1

0x11 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x05 0x20 0x00 0x00 0x40 0x01 0x00 0x00 0x01 0x01 0x00 0x00 0x00 0x00 0xFF 0x00 DATA(0x00 ....)

### 1.5.2 扩展协议

28字节协议头+数据+两个字节协议尾

不带数据心跳包展示,协议版本号设置为1,回复请求设置为0,后续数据类型设置为0

0x11 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0A 0x00 0x01 0x0A 0x00 0x00 0x00 0x00 0x00 0x00 DATA(0x00...) 0x00 0xFF

带数据包的网络验证协议,协议尾的CHECKSUM被设置有值,回复请求设置为真

0x11 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x05 0x20 0x00 0x00 0x40 0x01 0x00 0x01 DATA(0x00...) 0x7E 0xFF

# 二 使用协议

## 2.1 主协议

我们提供的协议分了类,一个主类,在协议头里面的unOperatorType表示,这个值表示一个主要的大类协议,还有子协议用于区别.

主协议里面的协议为系统默认协议,用户的自定义协议不应该与系统内部定义的协议相同.下面演示了主协议类型

typedef enum en\_NetEngine\_XComm\_Protocol

{

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_NORMAL = 0, //普通协议

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_XLOG = 1, //网络日志协议

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_AUTH = 2, //网络验证服务协议

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_SMS = 3, //流媒体协议

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_FILE = 4, //文件传输协议

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_BACKSERVICE = 5, //后台服务协议

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP = 6, //P2XP协议

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_XMQ = 7, //消息队列服务

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_XDDS = 8, //消息分发服务

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_HEARTBEAT = 0xA, //心跳协议

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_CHUNKED = 0xB, //CHUNKED包模式，使用此模式协议头的unPacketSize字段Post将无效

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_LEAVE = 0xF, //离开包，投递了这个包后后续包都将被抛弃

ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_USER = 100 //用户包,用户自定义包开始为101开始100以内为协议内部保留

}ENUM\_NETENGINE\_XCOMM\_PROTOCOL;

如果用户想要使用我们的系统并且自定义协议,那么需要定义的主协议的值需要定义到100以上才是最好的,否则有可能会引发冲突.

## 2.2 子协议

子协议的字段是协议头的unOperatorCode表示,表示一个主协议下面有多少个子协议.在子协议最开始部分,我们还定义了通用子协议.这些协议可能适用于所有类型的协议.

#define XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_ISOK 0x0001 //通用成功

#define XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_ISFAILED 0x0002 //通用失败

#define XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_LEAVE 0x000A //离开包

## 2.3 协议分类

我们的协议分为主动请求类型和通知类型

主动请求类型的协议用户必须请求后服务器才会回复,这个表示一请求一应答模式.

通知类型的协议是用户启用后服务器会自动推送的协议.用户只需要一直接受即可

# 三 系统协议

系统协议是指系统内部定义好的可以直接使用的协议.下面展示的通用协议说明都是我们内部模块已经定义好的操作.如果你想要自己实现这些协议,可以不用关心.但是,你想要使用我们的组包拆包模块,你必须按照规定使用我们的协议头,协议体可以自己定义.

我们的协议,客户端到服务器的请求CODE字段都有\*\*\*\*REQ\*\*的表示方法,服务器回复的CODE字段都是REP的表示方法,他们是一一对应的.

服务器回复的处理结果,一般都采用JSON表示,除了CODE表示服务器处理的协议意外,我们的JSON用来表示处理结果,Code 字段,0一般表示成功,其他值表示失败,小于0表示服务器处理错误,大于0客户端请求的数据有问题.当然,在某些时候会有CodeMsg字段显示错误的意思,一般都只有在错误的时候才会有这个字段.

## 3.1 日志协议

日志协议是属于通知协议,不需要用户请求,在用户启用日志后通过参数设置开启网络日志后即可使用.

日志协议格式是协议头+协议体+日志内容,如下所示:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_XLOG

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_XLOG\_PUSH

unPacketSize = sizeof(XENGINE\_XLOG\_PROTOCOL)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_XLog\_Protocol

{

*CHAR* tszFuncName[64]; //函数名称

*CHAR* tszLogTimer[64]; //日志时间

int nLogLine; //代码行数

int nLogLeave; //日志级别

int nLogLen; //打印的日志长度

}XENGINE\_XLOG\_PROTOCOL, \*LPXENGINE\_XLOG\_PROTOCOL;

## 3.2 验证协议

参考网络验证服务文档

## 3.3 流媒体协议

参考流媒体服务器文档

## 3.4 存储服务协议

参考存储服务说明文档

## 3.5 后台服务协议

后台服务协议参考后台服务说明文档

## 3.6 P2XP协议

P2XP协议参考存储服务说明文档

## 3.7 消息队列

消息队列协议参考消息服务中的文档.

## 3.8 消息分发

消息分发协议参考消息服务文档

## 3.9 UDX协议

UDX传输协议基于UDP协议开发而来,如果你对传输性能要求高,但是对传输完整性要求低,可以使用此协议.此协议只能在UDP上面进行数据传输!

每个UDX包不能超过MTU大小,如果超过,需要分包处理.每次分包处理的数据必须带上协议头.

协议头序列号是必须的,每次一个包就加1,最大值为65535,超过此值需要重置为0

注意:UDX底层不能使用HelpComponents\_Packets模块

### 3.9.1 数据传输协议

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_UDX

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_UDX\_TRANSMISSION

unPacketSize = DATALEN

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:后续数据

### 3.9.2 登录协议

登录协议,只有协议头,请求的登录协议unPacketSzie表示自身可以使用的滑动窗口大小,单位:BYTE,请求端的值表示客户端滑动窗口大小,服务器返回的值表示服务器可使用滑动窗口大小.

#### 3.9.2.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_UDX

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_AUTH\_REQLOGIN

unPacketSize = WINDOWSIZE

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.9.2.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_UDX

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_AUTH\_REPLOGIN

unPacketSize = WINDOWSIZE

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

### 3.9.3 数据重传

wPacketSerial 表示丢包的序列号,只能在当前序列中进行.如果序列号已经第二次轮训,那么将无法重传.

#### 3.9.3.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_UDX

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_UDX\_RETREQ

unPacketSize = 0

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.9.3.2 回复

如果对端找到了重传包,那么没有特定的回复,而是由对端发送接受到的请求重传的数据包.

如果对端没有找到需要重传的包,会立即返回一个协议告知对端.如下所示.那么表示这个包已经被丢弃.

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_UDX

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_UDX\_RETREP

unPacketSize = 0

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

### 3.9.4 通知协议

通知协议仅仅作为通知,接收端不需要任何回复

#### 3.9.4.1 序列号通知

协议头:wPacketSerial表示通知方当前已经接收到数据包的最新序列号,这个时候我们就可以不在缓存之前序列号的包了.

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_UDX

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_UDX\_NOTIFYACK

unPacketSize = 0

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.9.4.2 滑动窗口通知

协议头:unPacketSize表示通知方当前滑动窗口大小

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_UDX

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_UDX\_NOTIFYWINDOW

unPacketSize = WINDOWSIZE

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.9.4.3 关闭通知

协议头:表示对方关闭,此通知不是必须的

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_UDX

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_UDX\_NOTIFYCLOSE

unPacketSize = 0

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

## 3.10 RPC协议

RPC协议参考后台控制服务文档

## 3.11 心跳服务

心跳服务用来进行包活计时,在网络通信中尤为重要,在客户端上服务器后,就需要进行心跳通信.心跳通信每隔5秒钟一次,超过3次(15秒),服务器就会认为客户端掉线.

### 3.11.1 心跳同步

#### 3.11.1.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_HEARTBEAT

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_HB\_SYN

unPacketSize = sizeof(XENGINE\_PROTOCOLHEARTBEAT)

byVersion = 1

byIsReply = TRUE OR FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XNegine\_Protocol\_HeartBeat

{

*CHAR* tszMachineAddr[32]; //机器IP地址

int nMachineCode; //服务编号

\_\_int64x nTimer; //心跳时间 time(NULL)

struct

{

int nCpuUsage; //CPU占用率

int nMemoryUsage; //内存占用率

int nNetUsage; //网络占用率

int nDiskUsage; //硬盘占用率

int nGraphUsage; //显卡占用率

}st\_HBComputerInfo;

}XENGINE\_PROTOCOLHEARTBEAT,\*LPXENGINE\_PROTOCOLHEARTBEAT;

#### 3.11.1.2 回复

只有请求的时候byIsReply 为真的时候才会有回复包,不为真服务器将不会返回任何数据包给客户端.

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_HEARTBEAT

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_HB\_ACK

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

## 3.12 分包协议

Chunked协议为协议头专用的分包协议.对于一些大型数据包,或者对于后续数据包大小不明确的协议包类型,采用此协议包来进行分包发送.

CHUNKED包的格式是:协议头(START标志) + 数据(DATA...) + 协议头(END标志)

注意:Chunked模式只支持普通协议包!

### 3.12.1 分包开始

要开始一个分包,需要在数据的最开始位置发送协议头,告知组包服务这是一个Chunked包.在此协议头后面跟上数据.

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_CHUNKED

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_CHUNKED\_START

unPacketSize = 0

byVersion = 0

byIsReply = TRUE OR FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

数据:.......

### 3.12.2 分包结束

在发送数据完毕的时候,要通知组包服务完毕,需要在数据包末尾发送一个结束标志协议头.协议如下所示.至此,一个CHUNKED包才算完成.

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_CHUNKED

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_CHUNKED\_END

unPacketSize = 0

byVersion = 0

byIsReply = TRUE OR FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

## 3.13 离开协议

离开协议按照正确的说明,这并不是一个真正的协议包,他的作用只是通知组包服务这个包是最后一个包了.这样做的目的可以为一些安全性和准确性较高的服务提供帮助,让用户不管在任何时候都可以正确无误的处理完队列中的所有数据包!

注意:当组包服务收到这个离开包后,与这个标志关联的ID将无法继续组包.后续数据会被抛弃!协议头的CODE字段不是必须的!可以根据用户自身情况填充!

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_LEAVE

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_LEAVE

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

# 四 用户协议

用户协议指定此协议非系统协议,也不是系统保留的协议,用户协议的使用方式是

USERPROTO\_DEFINE = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_USER + 1

## 4.1 用户协议规范

使用用户协议,你需要自己定义,当然,你可以使用我们的组包模块来组包.但是无论你是用的系统协议,还是用户协议,你都应该遵守我们协议头的规定和特别说明规定.

# 五 特别说明

## 5.1 协议头

以下讲解了协议头一些保留类型和规范等!

注意:系统使用和系统保留字段,无论是用户协议还是系统自定义协议,用户都不应该继续使用这些值.

### 5.1.1 加密类型

在扩展协议头中,有加密标志符unPacketCrypt ,0表示不加密.下面1-5被系统预定了,6-15由用户自定义.系统预定的加密协议如下所示

* 1:AES
* 2:DES
* 3:RSA
* 4:XEngine加解密
* 5:系统保留
* 6-15:用户自定义

### 5.1.2 负载类型

数据负载类型表示后续数据包的负载类型,wPayload此字段被用作表明后续数据负载类型.其中0-5被使用,6- 9被保留.10以后用户可以自定义.系统定义的类型如下:

* 0:未定义,没有后续数据
* 1:二进制数据
* 2:JSON数据
* 3:BSON数据
* 4:XML数据
* 5:文本或者字符串
* 6:压缩数据
* 7:图片
* 8:视频
* 9:音频
* 10-99:系统保留
* 100-?:用户使用

## 5.2 权限级别

我们提供了权限级别相关操作,如果没有特殊说明,权限级别都按照下面的方式进行规定.权限级别表示方式为整数型,从0开始.最小的最大.

一般情况下,管理员级别使用0-9表示,VIP商业用户使用10-19表示,100过后的用户表示普通权限.下面的列表清楚的说明了权限级别使用方式.

* 0:超级管理员,ROOT级别,一般不启用
* 1:管理员权限,最高用户级别.可以拥有服务的所有功能
* 2.督察(审核))员权限,用于监督和管理服务器
* 3.运维管理:用于关注服务器运行情况
* 10:商业用户,VIP用户级别
* 100:普通用户级别

# 附录

## 附录1 类型定义

参考文件 XEngine\_CommHdr.h

## 附录2 协议定义

参考文件XEngine\_ProtocolHdr.h

## 附录3 转换定义

参考文件 XEngine\_Types.h 只有LINUX下才有!