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](#_Toc20189)

[前言 5](#_Toc22155)

[阅读者 5](#_Toc2195)

[概述 5](#_Toc26417)

[协议定义 5](#_Toc15717)

[相关模块 5](#_Toc13328)

[一 协议规范 5](#_Toc13797)

[1.1 协议头说明 6](#_Toc18901)

[1.2 协议头格式 6](#_Toc8426)

[1.2.1 普通协议头 6](#_Toc1387)

[1.2.2 扩展协议头 6](#_Toc29534)

[1.3 协议头字段 7](#_Toc3350)

[1.3.1 普通协议头字段 7](#_Toc13160)

[1.3.2 扩展协议头字段 7](#_Toc19313)

[1.4 协议头表示 8](#_Toc29191)

[1.5 协议展示 8](#_Toc17191)

[1.5.1 普通协议 8](#_Toc3666)

[1.5.2 扩展协议 8](#_Toc2462)

[二 使用协议 9](#_Toc30719)

[2.1 主协议 9](#_Toc26618)

[2.2 子协议 9](#_Toc4833)

[2.3 协议分类 9](#_Toc23829)

[三 系统协议 10](#_Toc11121)

[3.1 日志协议 10](#_Toc20580)

[3.2 验证协议 11](#_Toc11435)

[3.2.1 删除协议 11](#_Toc1251)

[3.2.2 注册协议 12](#_Toc2499)

[3.2.3 登录协议 13](#_Toc5244)

[3.2.4 充值协议 14](#_Toc8823)

[3.2.5 找回密码 15](#_Toc16260)

[3.2.6 获取时间 16](#_Toc24336)

[3.2.7 网络验证 18](#_Toc31044)

[3.2.8 公告协议 18](#_Toc26960)

[3.2.9 超时通知 19](#_Toc8616)

[3.3 流媒体协议 19](#_Toc20953)

[3.3.1 推送协议 19](#_Toc7563)

[3.3.2 拉流协议 21](#_Toc11736)

[3.3.3 控制协议 23](#_Toc18999)

[3.3.4 通知协议 24](#_Toc22263)

[3.4 存储服务协议 24](#_Toc28177)

[3.4.1 文件传输协议 24](#_Toc2015)

[3.4.2 上传文件协议 25](#_Toc6034)

[3.4.3 下载文件协议 26](#_Toc8660)

[3.5 后台服务协议 27](#_Toc9453)

[3.5.1 协议处理 27](#_Toc3124)

[3.5.2 后台管理协议 28](#_Toc16924)

[3.6 P2XP协议 30](#_Toc25819)

[3.6.1 登录协议 30](#_Toc22572)

[3.6.2 内网地址同步 32](#_Toc12475)

[3.6.3 外网地址同步 33](#_Toc32761)

[3.6.4 用户查询 34](#_Toc30253)

[3.6.5 请求连接 36](#_Toc23483)

[3.7 消息队列 37](#_Toc14016)

[3.7.1 投递包协议 37](#_Toc5653)

[3.7.2 获取包协议 38](#_Toc11414)

[3.7.3 删除包协议 40](#_Toc31367)

[3.7.4 创建主题 41](#_Toc3979)

[3.7.5 删除主题 42](#_Toc31336)

[3.8 消息分发 43](#_Toc2239)

[3.8.1 请求应答协议 43](#_Toc19170)

[3.8.2 通知协议 45](#_Toc28469)

[3.9 UDX协议 47](#_Toc27513)

[3.9.1 数据传输协议 47](#_Toc32421)

[3.9.2 登录协议 47](#_Toc8745)

[3.9.3 数据重传 48](#_Toc442)

[3.9.4 通知协议 49](#_Toc8481)

[3.10 XRPC协议 50](#_Toc29513)

[3.10.1 函数调用 50](#_Toc27555)

[3.10.2 命令调用 53](#_Toc30286)

[3.11 心跳服务 53](#_Toc31378)

[3.11.1 心跳同步 53](#_Toc17066)

[3.12 分包协议 54](#_Toc26285)

[3.12.1 分包开始 55](#_Toc21749)

[3.12.2 分包结束 55](#_Toc19863)

[3.13 离开协议 55](#_Toc25072)

[四 用户协议 56](#_Toc4116)

[4.1 用户协议规范 56](#_Toc21070)

[五 特别说明 56](#_Toc15134)

[5.1 协议头 56](#_Toc29162)

[5.1.1 加密类型 56](#_Toc15043)

[5.1.2 负载类型 57](#_Toc5551)

[5.2 权限级别 57](#_Toc11507)

[附录 57](#_Toc21946)

[附录1 类型定义 57](#_Toc7070)

[附录2 协议定义 57](#_Toc9591)

[附录3 转换定义 57](#_Toc24782)

[附录4 更新历史 58](#_Toc8655)

|  |  |  |  |
| --- | --- | --- | --- |
| 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： | V7.15 | |
| 发布日期： | 2021-07-02 | |
| 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

验证协议用于网络验证.适用于网络验证组件.可用于登录,注册,注销,删除等操作.

请注意:验证协议使用了协议头的保留字段wReserve用于回复客户端请求,告诉客户端这次操作成功还是失败,0表示成功,-1表示失败

wCrypto表示加解密,目前仅仅支持类型4,XCrypto算法.如果需要加密,请填充4,否则请填充0

### 3.2.1 删除协议

用于删除一个用户

#### 3.2.1.1 请求

请求一次删除用户,你需要使用协议头+注册协议来实现,这次协议里面的内容用于判断和验证用户是否有权限删除自己.当然,配合TOKEN使用,还可以实现权限分配.使用TOKEN来验证这个用户有没有权限删除另一个用户

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(XENGINE\_PROTOCOL\_USERREG)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_Protocol\_UserReg

{

*TCHAR* tszUserName[64]; //用户名

*TCHAR* tszUserPass[64]; //密码

*TCHAR* tszEMailAddr[64]; //电子邮件地址

*TCHAR* tszLoginTime[64]; //登录时间

*TCHAR* tszRegTime[64]; //注册时间

\_\_int64x nQQNumber; //QQ号

\_\_int64x nPhoneNumber; //电话号码

\_\_int64x nIDNumber; //身份证号

int nUserLeave; //用户等级

*int nUserState*; //状态

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

#### 3.2.1.2 回复

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

### 3.2.2 注册协议

#### 3.2.2.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(XENGINE\_PROTOCOL\_USERREG)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_Protocol\_UserReg

{

*TCHAR* tszUserName[64]; //用户名

*TCHAR* tszUserPass[64]; //密码

*TCHAR* tszEMailAddr[64]; //电子邮件地址

*TCHAR* tszLoginTime[64]; //登录时间

*TCHAR* tszRegTime[64]; //注册时间

\_\_int64x nQQNumber; //QQ号

\_\_int64x nPhoneNumber; //电话号码

\_\_int64x nIDNumber; //身份证号

int nUserLeave; //用户等级

*int nUserState*; //是否在线

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

#### 3.2.2.2 回复

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

{  
    **"Code"**:**0**  
}

### 3.2.3 登录协议

登录协议是客户端和服务器在请求服务器交换数据的时候所需要做的第一个步骤.只有登录成功,才能使用服务器,涉及的服务器有,P2XP服务器,存储服务器,验证服务器.

#### 3.2.3.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(XENGINE\_PROTOCOL\_USERAUTH)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_Protocol\_Auth

{

*CHAR* tszUserName[64]; //用户名

*CHAR* tszUserPass[64]; //密码

ENUM\_PROTOCOLCLIENT\_TYPE enClientType; //客户端类型

ENUM\_PROTOCOLDEVICE\_TYPE enDeviceType; //设备类型

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

#### 3.2.3.2 回复

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

##### 3.2.3.2.1 回复类别

服务器回复成功和失败,可以通过开发人员自定义,或者通过统一标识来确定.在这里.我们通过wReserver来确定,这个值在登录协议上面,返回的数值将表达以下意思:

* 0:成功
* 1:用户不存在
* 2:密码错误
* 3:用户没有权限
* 4:用户已过期
* 10:协议出错,服务器无法解析
* 11:服务器内部错误

### 3.2.4 充值协议

用户充值协议是通过充值卡来充值的,充值卡需要通过我们的序列号模块来生成.

#### 3.2.4.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(AUTHREG\_PROTOCOL\_USERPAY)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct

{

*CHAR* tszUserName[XENGINE\_AUTHREG\_SERVICE\_SQL\_MAX\_USERNAME]; //用户

*CHAR* tszSerialNumber[128]; //序列号

}AUTHREG\_PROTOCOL\_USERPAY, \*LPAUTHREG\_PROTOCOL\_USERPAY;

#### 3.2.4.2 回复

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

### 3.2.5 找回密码

#### 3.2.5.1 请求

协议头:请求的协议,电子邮件,用户名和身份证以及电话号码不能为空

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(AUTHREG\_USERTABLE)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_AuthReg\_UserTable

{

*CHAR* tszUserName[XENGINE\_AUTHREG\_SERVICE\_SQL\_MAX\_USERNAME]; //用户

*CHAR* tszUserPass[XENGINE\_AUTHREG\_SERVICE\_SQL\_MAX\_USERNAME]; //用户密码

*CHAR* tszRegData[64]; //注册日期 日期格式：2013/1/5-12:33:33

*CHAR* tszLeftTime[64]; //剩余日期

*CHAR* tszEmailAddr[64]; //电子邮件地址

*CHAR* tszHardCode[32]; //硬件码

ENUM\_AUTHREG\_GENERATESERIALTYPE en\_AuthRegSerialType; //充值卡类型

\_\_int64x dwIDCard; //身份证

\_\_int64x dwQQNumber; //QQ号

}AUTHREG\_USERTABLE, \*LPAUTHREG\_USERTABLE;

#### 3.2.5.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(XENGINE\_PROTOCOL\_USERAUTH)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_Protocol\_Auth

{

*CHAR* tszUserName[64]; //用户名

*CHAR* tszUserPass[64]; //密码

ENUM\_PROTOCOLCLIENT\_TYPE enClientType; //用户类型

ENUM\_PROTOCOLDEVICE\_TYPE enDeviceType; //设备类型

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

### 3.2.6 获取时间

获取时间表示获取用户所剩余的时间.

#### 3.2.6.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(XENGINE\_PROTOCOL\_USERAUTH)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_Protocol\_Auth

{

*CHAR* tszUserName[64]; //用户名

*CHAR* tszUserPass[64]; //密码

ENUM\_PROTOCOLCLIENT\_TYPE enClientType; //用户类型

ENUM\_PROTOCOLDEVICE\_TYPE enDeviceType; //设备类型

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

#### 3.2.6.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(AUTHREG\_PROTOCOL\_TIME)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct

{

*CHAR* tszUserName[XENGINE\_AUTHREG\_SERVICE\_SQL\_MAX\_USERNAME]; //用户

\_\_int64x nTimeONLine; //在线时间

\_\_int64x nTimeLeft; //剩余时间

ENUM\_AUTHREG\_GENERATESERIALTYPE enSerialType; //充值卡类型

}AUTHREG\_PROTOCOL\_TIME, \*LPAUTHREG\_PROTOCOL\_TIME;

### 3.2.7 网络验证

临时网络验证协议,可以不需要注册登录即可使用,通过一段唯一的识别码(CPUID,主板ID,硬盘ID)来进行注册.此注册会发送到服务器,服务器会自动进行记录保存,并且识别是否过期,方便用户使用.

此协议比本地验证更加安全.可以方便一些不想注册的用户使用.

#### 3.2.7.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = IDLEN

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:填写唯一标识码,可以是CPUID等

#### 3.2.7.2 回复

协议头:成功将返回下面的内容,表示时间还没有过期

失败将返回超时通知

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = 0

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

### 3.2.8 公告协议

此协议只用于发送文本内容.用于通知用户消息.

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = MSGLEN

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:协议体为要发送的内容.

### 3.2.9 超时通知

此协议为通知协议,服务器主动下发,当客户端收到此协议后需要主动断开连接,否则服务器会一直下发此通知.

此协议表示了客户端的时间到期,没有剩余时间了.

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

## 3.3 流媒体协议

流媒体协议用于进行RTSP推流,他首先需要推送H264数据流到服务器,在由服务器进行流媒体转发!

### 3.3.1 推送协议

流媒体推送协议的协议头TOKEN字段必填,在请求的时候填写,表明用户在创建成功后,这个流的唯一ID,这样用户可以清楚的知道自己的推流ID.比如流ID是123,那么你的推送地址就是:Rtsp://192.168.1.10/live/123

#### 3.3.1.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 流媒体推流唯一ID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_SMS\_REQXPUSH

unPacketSize = sizeof(XENGINE\_AVPROTOCOL)

byVersion = 0

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_AVProtocol

{

*TCHAR* tszPktName[4]; //封装格式,如果没有,可以为NULL,封装格式为后缀.比如:mkv flv mp4

*BYTE* byPktFlag; //数据包封装格式,发送的音视频流后续格式,0:无协议裸流数据,1:标准协议头,2:扩展协议头

//视频信息

struct

{

*BOOL* bEnable; //是否启用

int nWidth; //视频宽

int nHeight; //视频高

int enAvCodec; //使用的编码器

int nBitRate; //码率

int nFrameRate; //帧率

int nVLen; //SPSPPS大小,为0将交由模块处理

*TCHAR* tszVInfo[256]; //SPS与PPS,你可以通过AVHelp\_MetaInfo\_Get264Hdr来处理,在SPS和PPS前面添加起始字节后一起拷贝到这里面

}st\_PushVideo;

//音频信息

struct

{

*BOOL* bEnable; //是否启用

int nChannel; //通道个数

int nBitRate; //码率

int nSampleRate; //采样率.44100 ...

int nSampleFmt; //采样格式,S16 S32...

int enAvCodec; //编码器

}st\_PushAudio;

}XENGINE\_AVPROTOCOL, \* LPXENGINE\_AVPROTOCOL;

#### 3.3.1.2 回复

wReserve:如果此值为0表示成功.其他值表示失败,参考如下信息:

* 1:协议验证失败
* 2:不是推流协议
* 3.没有找到SPS或者PPS信息,这个错误在推流第一个包如果出错会出现,如果没有错误,不会继续推送,也就是说,如果有错误,你会收到两个包头

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 流媒体推流唯一ID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_SMS\_REPXPUSH

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

### 3.3.2 拉流协议

用于从流媒体服务器拉取数据流,开始播放前,这个是必须的

#### 3.3.2.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 流媒体推流唯一ID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_SMS\_REQXPULL

unPacketSize = 0

byVersion = 0

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.3.2.2 回复

wReserver字段为0,表示成功,其他值表示失败.失败的值如下:

1:没有找到这个推流的ID

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 流媒体推流唯一ID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_SMS\_REPXPULL

unPacketSize = sizeof(XENGINE\_AVPROTOCOL)

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_AVProtocol

{

*TCHAR* tszPktName[4]; //封装格式,如果没有,可以为NULL,封装格式为后缀.比如:mkv flv mp4

*BYTE* byPktFlag; //数据包封装格式,发送的音视频流后续格式,0:无协议裸流数据,1:标准协议头,2:扩展协议头

//视频信息

struct

{

*BOOL* bEnable; //是否启用

int nWidth; //视频宽

int nHeight; //视频高

int enAvCodec; //使用的编码器

int nBitRate; //码率

int nFrameRate; //帧率

int nVLen; //SPSPPS大小,为0将交由模块处理

*TCHAR* tszVInfo[256]; //SPS与PPS,你可以通过AVHelp\_MetaInfo\_Get264Hdr来处理,在SPS和PPS前面添加起始字节后一起拷贝到这里面

}st\_PushVideo;

//音频信息

struct

{

*BOOL* bEnable; //是否启用

int nChannel; //通道个数

int nBitRate; //码率

int nSampleRate; //采样率.44100 ...

int nSampleFmt; //采样格式,S16 S32...

int enAvCodec; //编码器

}st\_PushAudio;

}XENGINE\_AVPROTOCOL, \* LPXENGINE\_AVPROTOCOL;

### 3.3.3 控制协议

控制类协议没有回复,只有请求,如果发现操作没有成功,可以再次发送,指导成功为止

#### 3.3.3.1 播放控制

只有向服务器请求了此协议,服务器才会推送数据给拉流端

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 流媒体推流唯一ID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_SMS\_CTRLPLAY

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.3.3.2 播放暂停

如果不想接续接受数据,可以调用此函数来暂停服务器的拉流,如果想继续播放,可以调用播放协议

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 流媒体推流唯一ID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_SMS\_CTRLPAUSE

unPacketSize = 0

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

### 3.3.4 通知协议

通知协议用于处理推送拉流状态,告知另一方目前接受数据和处理数据情况,让发送端或者接收端调整发送频率!

#### 3.3.4.1 拉流通知协议

拉流协议和推流协议是一样的,只是CODE值不同,下面不在另外演示.

另外注意:推流协议不用用于拉流,同样的,拉流通知协议不能用于推流

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 流媒体推流唯一ID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_SMS\_STATPUSH

unPacketSize = sizeof(XENGINE\_SMSPROTOCOL)

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_SMSProtocol

{

XNETHANDLE xhXTPTime; //同步时间戳

ULONGLONG nPacketCount; //当前推送(接受)数据大小

ULONGLONG nPacketHandle; //已处理(已发送)数据大小

ULONGLONG nPacketLeft; //剩余数据大小

}XENGINE\_SMSPROTOCOL, \* LPXENGINE\_SMSPROTOCOL;

## 3.4 存储服务协议

存储服务协议用来操作存储服务器

### 3.4.1 文件传输协议

文件传输协议用于推送协议,非请求应答模式,而是请求完上传或下载文件协议后,服务器进行数据推送的协议,表示数据包为文件.

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 文件推送协议唯一码

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

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

unPacketSize = 文件大小

byVersion = 0

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

### 3.4.2 上传文件协议

在一个文件开始上传之前,需要先沟通好文件上传信息,只有上传信息确认后,才能进行文件推送.

#### 3.4.2.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_STORAGE\_REQUPDATE

unPacketSize = sizeof(XENGINE\_PROTOCOLFILE)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolFile

{

*CHAR* tszFilePath[*MAX\_PATH*]; //文件路径

*CHAR* tszFileName[*MAX\_PATH*]; //文件名称

*CHAR* tszFileUser[64]; //文件所属用户

*CHAR* tszFileMD5[64]; //文件MD5

*CHAR* tszFileTime[64]; //文件创建时间

\_\_int64x nFileSize; //文件大小

}XENGINE\_PROTOCOLFILE,\*LPXENGINE\_PROTOCOLFILE;

#### 3.4.2.2 回复

回复确认成功,协议头的xhToken会被填充值,在推送协议的TOKEN里面这个值必须填写.否则将无效!

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 文件推送协议唯一码

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_STORAGE\_REPUPDATE

unPacketSize = JSONSIZE

byVersion = 2

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

{  
    **"Code"**:**0**  
}

Code值可能是其他值,如果是其他值,表示有错误,系统会回复你错误信息.

### 3.4.3 下载文件协议

下载文件请求和上传文件请求协议流程一样

#### 3.4.3.1 请求

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_STORAGE\_REQDOWN

unPacketSize = sizeof(XENGINE\_PROTOCOLFILE)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolFile

{

*CHAR* tszFilePath[*MAX\_PATH*]; //文件路径

*CHAR* tszFileName[*MAX\_PATH*]; //文件名称

*CHAR* tszFileUser[64]; //文件所属用户

*CHAR* tszFileMD5[64]; //文件MD5

*CHAR* tszFileTime[64]; //文件创建时间

\_\_int64x nFileSize; //文件大小

}XENGINE\_PROTOCOLFILE,\*LPXENGINE\_PROTOCOLFILE;

#### 3.4.3.2 回复

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 文件推送协议唯一码

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_STORAGE\_REPDOWN

unPacketSize = sizeof(XENGINE\_PROTOCOLFILE) + JSONSIZE

byVersion = 2

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:此协议结构体会告诉客户端文件的一些基本信息

typedef struct tag\_XEngine\_ProtocolFile

{

*CHAR* tszFilePath[*MAX\_PATH*];

*CHAR* tszFileName[*MAX\_PATH*];

*CHAR* tszFileMD5[64];

*CHAR* tszFileTime[64];

\_\_int64x nFileSize;

}XENGINE\_PROTOCOLFILE,\*LPXENGINE\_PROTOCOLFILE;

{  
    **"Code"**:**0**  
}

当你收到此包后,后续数据将会是二进制文件流.直到文件发送完毕,不会在有其他数据存在!

## 3.5 后台服务协议

后台服务协议用于后台程序服务,适用于XEngine\_XContral服务,可以通过此服务对一个机器进行管理.后台服务协议没有回复确认包

后台服务协议采用HTTP RESTFUL API.请求-应答模式

### 3.5.1 协议处理

注意:每种协议的JSON里面都需要带上”unOperatorCode”:0x5001 字段.这个字段代表需要处理的协议类型.这个类型可以通过协议头找到,比如0x5001表示下载.

### 3.5.2 后台管理协议

#### 3.5.2.1 下载协议

{  
    **"DownloadUrl"**:**"[http://www.xyry.org/file.exe](http://www.xyry.org/file.exe" \t "https://www.json.cn/_blank)"**,  
    **"SaveUrl"**:**"C:\1\file.exe"**  
}

#### 3.5.2.2 删除文件协议

{  
    **"DeleteFile"**:**"C:\1\file.exe"**  
}

#### 3.5.2.3 删除文件夹协议

{  
    **"DeleteDir"**:**"C:\1\"**  
}

#### 3.5.2.4 上传文件协议

上传文件协议使用FTP进行上传文件,你必须准备好一个FTP服务器,当指令被客户端收到后,就会立即上传文件.

{  
    **"UPLoadFile"**:**"C:\1\file.exe"**,  
 **"UPLoadUrl"**:**"ftp://192.168.0.1/1.rar,用户名密码格式：ftp://ftpuser:123123@192.168.1.103/1.rar"**  
}

#### 3.5.2.5 获取文件列表协议

##### 3.5.2.5.1 请求

{  
    **"PostUrl"**:**"[http://www.xyry.org/api/listfile](http://www.xyry.org/api/listfile" \t "https://www.json.cn/json/_blank)"**,  
    **"FilePath"**:**"C:\1\file.exe"**  
}

##### 3.5.2.5.2 回复

如果成功,那么将发送一个HTTP POST请求到你指定的POSTURL地址中.并且负载如下格式的JSON内容

{  
    **"Count"**:**2**,  
    **"List"**:[  
        {  
            **"FileName"**:**"1.exe"**,  
            **"FileDir"**:**"C:\1\"**  
        },  
        {  
            **"FileName"**:**"2.exe"**,  
            **"FileDir"**:**"C:\1\"**  
        }  
    ]  
}

#### 3.5.2.6 执行文件协议

协议体:ExecShow,1代表前端显示,0代表运行到后台

{  
    **"ExecShow"**:**1**,  
    **"ExecFile"**:**"C:\1\1.exe"**  
}

#### 3.5.2.7 发送消息

此协议暂时不支持LINUX版本

{  
    **"MessageBox"**:**"弹出消息"**  
}

#### 3.5.2.8 关闭系统

协议体:支持的类型参考系统模块的SystemApi\_System\_Shutdown函数参数

{  
    **"ShutDownType"**:1  
}

#### 3.5.2.9 执行命令

{  
    **"tszExecCmd"**:**"要执行的命令"**  
}

#### 3.5.2.10 连接

{  
    **"tszIPAddr"**:**"ip地址"**,  
    **"nPort"**:**32**,  
    **"nIPType"**:**2**  
}

nIPType表示TCP还是UDP,需要按照socket头文件定义类型来设置

## 3.6 P2XP协议

P2XP协议用于点对点传输和打洞,可以通过局域网同步列表等协议获取相关相同局域网下的机器列表,来实现点对点,绕过公网服务实现数据传输.

P2XP协议与其他协议不同的是,他支持包跳转模式,所以,我们每次在协议头后面必须添加一个XENGINE\_P2XPPROTOCOL协议包,在添加其他协议数据.

### 3.6.1 登录协议

P2XP的登录协议类型使用的是标准协议,但是登录的数据包有变化,需要特定数据包传输给服务器进行解析保存,所以请求的格式是

协议头+P2XP协议+客户端信息+登录验证协议,这4个结构体组成一个包进行登录.

#### 3.6.1.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = sizeof(XENGINE\_PROTOCOL\_USERAUTH) + sizeof(XENGINE\_P2XPPEER\_PROTOCOL)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_Protocol\_Auth

{

*CHAR* tszUserName[64]; //用户名

*CHAR* tszUserPass[64]; //密码

ENUM\_PROTOCOLCLIENT\_TYPE enClientType; //客户端类型

ENUM\_PROTOCOLDEVICE\_TYPE enDeviceType; //设备类型

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

typedef struct tag\_XEngine\_P2XPPeer\_Protocol

{

*CHAR* tszUserName[64]; //用户信息

*CHAR* tszPrivateAddr[32]; //私有本地地址，内网地址

*CHAR* tszPublicAddr[32]; //外网地址

*CHAR* tszConnectAddr[32]; //链接地址

*CHAR* tszUserLocation[32]; //位置信息

*CHAR* tszUserArea[20]; //用户ISP

\_\_int64x dwConnectType; //连接类型

*WORD* dwPeerType; //节点类型

}XENGINE\_P2XPPEER\_PROTOCOL, \*LPXENGINE\_P2XPPEER\_PROTOCOL;

#### 3.6.1.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

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

unPacketSize = JSONSIZE

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

{  
    **"Code"**:**0**,  
    **"CodeMsg"**:**"如果为错误,这个字段将存在"**  
}

### 3.6.2 内网地址同步

用户请求同步内网IP地址协议,会根据此用户登录的内网IP地址,自动分析出当前内网下已经登录上的IP地址列表

#### 3.6.2.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_P2XP\_REQLANLIST

unPacketSize = 0

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.6.2.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_P2XP\_REPLANLIST

unPacketSize = JSONSIZE

byVersion = 2

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

{  
    **"Code"**:**0**,  
    **"ClientCount"**:**2**,  
    **"ClientArray"**:[  
        {  
            **"ClientAddr"**:**"192.168.1.101"**,  
            **"ClientUser"**:**"123123aa"**  
        },  
        {  
            **"ClientAddr"**:**"192.168.1.102"**,  
            **"ClientUser"**:**"123123bb"**  
        }  
    ]  
}

### 3.6.3 外网地址同步

内网地址只同步内网IP,外网地址同步可以同步在这个外网IP地址下面的所有内网IP地址,对于一些大型网络公司和环境复杂的内网,使用此功能可以方便判断出在同一网络下的用户.

#### 3.6.3.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_P2XP\_REQWLANLIST

unPacketSize = 0

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.6.3.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_P2XP\_REPWLANLIST

unPacketSize = JSONSIZE

byVersion = 2

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:GateArray表示外网IP下面分了多少个内网IP地址.ClientGet表示内网IP地址

{  
    **"Code"**:**0**,  
    **"GateCount"**:**2**,  
    **"GateArray"**:[  
        {  
            **"ClientGate"**:**"192.168.1"**,  
            **"ClientCount"**:**2**,  
            **"ClientArray"**:[  
                {  
                    **"ClientAddr"**:**"192.168.1.101"**,  
                    **"ClientUser"**:**"123123aa"**  
                },  
                {  
                    **"ClientAddr"**:**"192.168.1.102"**,  
                    **"ClientUser"**:**"123123bb"**  
                }  
            ]  
        },  
        {  
            **"ClientGate"**:**"192.168.2"**,  
            **"ClientCount"**:**2**,  
            **"ClientArray"**:[  
                {  
                    **"ClientAddr"**:**"192.168.2.101"**,  
                    **"ClientUser"**:**"aaadddccc"**  
                },  
                {  
                    **"ClientAddr"**:**"192.168.2.102"**,  
                    **"ClientUser"**:**"aaadddzzz"**  
                }  
            ]  
        }  
    ]  
}

### 3.6.4 用户查询

用户查询可以查询一个指定用户的网络连接信息.

#### 3.6.4.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_P2XP\_REQUSERQUERY

unPacketSize = sizeof(XENGINE\_P2XPPEER\_PROTOCOL)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:你需要填充P2XP客户端信息结构里面的用户信息成员,表明查找用户

typedef struct tag\_XEngine\_P2XPPeer\_Protocol

{

*CHAR* tszUserName[64]; //用户信息

*CHAR* tszPrivateAddr[32]; //私有本地地址，内网地址

*CHAR* tszPublicAddr[32]; //外网地址

*CHAR* tszConnectAddr[32]; //链接地址

*CHAR* tszUserLocation[32]; //位置信息

*CHAR* tszUserArea[20]; //用户ISP

\_\_int64x dwConnectType; //连接类型

*WORD* dwPeerType; //节点类型

}XENGINE\_P2XPPEER\_PROTOCOL, \*LPXENGINE\_P2XPPEER\_PROTOCOL;

#### 3.6.4.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_P2XP\_REPUSERQUERY

unPacketSize = JSONSIZE

byVersion = 2

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

{  
    **"Code"**:**0**,  
    **"dwConnectType"**:**1**,  
    **"dwPeerType"**:**1**,  
    **"tszConnectAddr"**:**"192.168.1.101:33990"**,  
    **"tszPrivateAddr"**:**"192.168.1.101"**,  
    **"tszPublicAddr"**:**"114.114.114.114"**,  
    **"tszUserArea"**:**"四川省成都市"**,  
    **"tszUserLocation"**:**"电信"**,  
    **"tszUserName"**:**"123123aa"**  
}

### 3.6.5 请求连接

请求连接协议可以要求一个客户端连接到你的指定地址和端口.这个协议用于点对点传输数据的时候最开始请求的协议,服务器在收到请求连接协议后,会把这条数据转发给客户端,客户端会受到与请求服务器一样的协议,这个时候客户端需要主动连接到指定的IP地址和端口!

我们没有制定点对点传输的协议,所以需要开发者自己指定,我们只负责打通数据!

#### 3.6.5.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_P2XP\_REQCONNECT

unPacketSize = sizeof(XENGINE\_P2XPPEER\_PROTOCOL)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:P2XP客户端IO结构你需要全部填充,表明要哪个DestUser连接到你,地址和端口必须优先绑定.

typedef struct tag\_XEngine\_P2XPIO\_Protocol

{

*CHAR* tszSourceUser[32]; //请求连接的用户

*CHAR* tszDestUser[32]; //要连接的用户

*CHAR* tszConnectAddr[32]; //连接的IP地址

int nDestPort; //要连接的端口

*BOOL* bIsTcp; //连接类型TCP,否则为UDP

}XENGINE\_P2XPIO\_PROTOCOL, \*LPXENGINE\_P2XPIO\_PROTOCOL;

#### 3.6.5.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

unOperatorType = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_TYPE\_P2XP

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_P2XP\_REPCONNECT

unPacketSize = JSONSIZE

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

{  
    **"Code"**:**0**  
}

## 3.7 消息队列

消息队列服务协议是XEngine新推出的一种消息队列服务程序所使用的协议,这种协议支持是基于TCP之上的,它支持私有XEngine通用协议和HTTP协议,他可以永久保存不掉线.

请注意:协议头的wReserver字段将作为服务器回复处理结果使用.为0表示处理成功,其他值表示失败!  
 协议负载无论是通用协议还是HTTP协议,他们负载的内容都可以负载JSON.具体参考协议说明.

### 3.7.1 投递包协议

要使用消息队列,你需要先投递一个包给消息队列服务器,这样其他程序才能从消息队列服务中取得一个包.

#### 3.7.1.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REQPOST

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ) + MSGLEN

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间，单位秒，如果为0，获取一次后被抛弃。-1 永久存在，如果有多个永久存在的包nSerial必须有值

int nGetTimer; //可以获取的次数

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

末尾跟上DATA数据

#### 3.7.1.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REPPOST

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

### 3.7.2 获取包协议

#### 3.7.2.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REQGET

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

#### 3.7.2.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REPGET

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ) + MSGLEN

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

协议体:如果成功,协议后面跟上获取到的数据

### 3.7.3 删除包协议

#### 3.7.3.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REQDEL

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

#### 3.7.3.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REPDEL

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

### 3.7.4 创建主题

要投递一个包到消息队列,你必须先创建一个主题,必须指明主题,数据才能进入消息队列.

#### 3.7.4.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REQCREATE

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

#### 3.7.4.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REPCREATE

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

### 3.7.5 删除主题

#### 3.7.5.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REQDELETE

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

#### 3.7.5.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_MQ\_REPDELETE

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XMQ)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

typedef struct tag\_XEngine\_ProtocolXmq

{

*TCHAR* tszMQKey[256]; //此消息的KEY，不能为空

\_int64x nSerial; //包序列号

int nKeepTime; //保存时间

int nGetTimer; //可以获取的次数

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

## 3.8 消息分发

消息分发服务不关心服务器和客户端,所以里面的内容如果要自己协议需要有几点注意的地方.

1:消息分发协议使用广播协议

2:消息分发内容使用组播协议

3:消息协议采用一请求一应答模式

### 3.8.1 请求应答协议

请求应答协议为一请求一应答模式,但是不会必定应答,所以需要有超时机制.如果超时没有接受到应答,说明局域网中不存在你查询的内容!

#### 3.8.1.1 查询域

用于查询当前环境中存在的域

##### 3.8.1.1.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_DDS\_REQDOMAINQUERY

unPacketSize = 0

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

##### 3.8.1.1.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_DDS\_REPDOMAINQUERY

unPacketSize = JSONSIZE

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

{  
    **"ObjectCount"**:**1**,  
    **"ObjectArray"**:[  
        {  
            **"nDDSDomain"**:**1**,  
            **"tszDDSAddr"**:**"192.168.1.2"**  
        }  
    ]  
}

#### 3.8.1.2 查询主题

用于查询指定域的指定主题是否存在

##### 3.8.1.2.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = DomainID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_DDS\_REQTOPICQUERY

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XDDS)

byVersion = 1

byIsReply = TRUE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:bCreater 只查询创建者,bTcp 查询TCP还是UDP协议的创建者

st\_XDDSProtocol.bCreater = TRUE;

st\_XDDSProtocol.bTcp = FALSE;

st\_XDDSProtocol.tszTopic = “查询的主题名称”

##### 3.8.1.2.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = DomainID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_DDS\_REPTOPICQUERY

unPacketSize = JSONSIZE

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

{  
    **"ObjectCount"**:**1**,  
    **"ObjectArray"**:[  
        {  
            **"tszTopic"**:**"aa"**,  
            **"tszDDSAddr"**:**"192.168.1.2"**,  
            **"bCreater"**:**1**,  
            **"bTcp"**:**0**,  
            **"nPort"**:**10000**  
        }  
    ]  
}

### 3.8.2 通知协议

通知协议不会有应答,通知协议通知一次后不在需要通知!通知协议不是必须的协议,用回可以忽略.通知协议需要上层调用者自己处理!

#### 3.8.2.1 域删除协议

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = DomainID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_DDS\_DELETEDOMAIN

unPacketSize = 0

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

#### 3.8.2.2 主题删除协议

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = DomainID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_DDS\_DELETETOPIC

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XDDS)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

st\_XDDSProtocol.bCreater = TRUE;

st\_XDDSProtocol.bTcp = FALSE;

st\_XDDSProtocol.tszTopic = “删除的主题名称”

#### 3.8.2.3 发布者创建协议

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = DomainID

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

unOperatorCode = XENGINE\_COMMUNICATION\_PROTOCOL\_OPERATOR\_CODE\_DDS\_PUBLISHCREATE

unPacketSize = sizeof(XENGINE\_PROTOCOL\_XDDS)

byVersion = 1

byIsReply = FALSE

wReserve = 0

wPacketSerial = 0

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

st\_XDDSProtocol.bCreater = TRUE;

st\_XDDSProtocol.bTcp = FALSE;

st\_XDDSProtocol.tszTopic = “创建的主题名称”

## 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 XRPC协议

XPRC协议为XEngine的远程过程调用协议,是一套C接口的协议.内部协议通过协议头+协议体的方式通信.XRPC支持下面的协议

### 3.10.1 函数调用

函数调用,用于调用服务器指定函数,函数调用允许多个执行参数.执行参数需要释义.用户在使用过程中需要注意参数类型定义.如下所示

* 0:未决的参数类型
* 1:无类型,一般用作返回值
* 2:逻辑型
* 3:整数型
* 4:无符号整数型
* 5:长整数型
* 6:无符号长整数型
* 7:浮点型
* 8:双精度浮点型
* 9:字符串
* 10:二进制

#### 3.10.1.1 标准协议

##### 3.10.1.1.1 请求

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_CODE\_XRPC\_REQFUNCALL

unPacketSize = JSONSIZE

byVersion = 2

byIsReply = TRUE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:

{  
    **"XRpc\_FunctionName"**:**"函数名称"**,

**"XRpc\_ReturnType"**:**2**,  
    **"XRpc\_ParamCount"**:**2**,  
    **"XRpc\_ParameteArray"**:[  
        {  
            **"ParameteType"**:**1**,  
            **"ParameteLen"**:**2**,  
            **"ParameteValue"**:**"wo"**  
        },  
        {  
            **"ParameteType"**:**2**,  
            **"ParameteLen"**:**4**,  
            **"ParameteValue"**:**10**  
        },  
        {  
            **"ParameteType"**:**2**,  
            **"ParameteLen"**:**4**,  
            **"ParameteValue"**:**20**  
        }  
    ]  
}

##### 3.10.1.1.2 回复

协议头:

wHeader = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_HEADER

xhToken = 0

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

unOperatorCode = ENUM\_XENGINE\_COMMUNICATION\_PROTOCOL\_CODE\_XRPC\_REPFUNCALL

unPacketSize = JSONSIZE

byVersion = 2

byIsReply = FALSE

wReserve = 0

wPacketSerial = 1

wTail = XENGIEN\_COMMUNICATION\_PACKET\_PROTOCOL\_TAIL

协议体:如果返回值是VOID,那么Len和VALUE值可以忽略

{  
    **"XRpc\_FunctionName"**:**"XEngine\_AddMethod"**,  
    **"XRpc\_ReturnLen"**:**4**,  
    **"XRpc\_ReturnType"**:**3**,  
    **"XRpc\_ReturnValue"**:**"4"**  
}

#### 3.10.1.2 HTTP协议

##### 3.10.1.2.1 请求

POST /XRPCApi/FuncCall HTTP/1.1

Host: 127.0.0.1:50002

Accept: \*/\*

Content-Length: 355

Content-Type: application/x-www-form-urlencoded

{  
    **"XRpc\_FunctionName"**:**"XEngine\_AddMethod"**,  
    **"XRpc\_ParamCount"**:**2**,  
    **"XRpc\_ParameteArray"**:[  
        {  
            **"ParameteLen"**:**4**,  
            **"ParameteType"**:**3**,  
            **"ParameteValue"**:**1**  
        },  
        {  
            **"ParameteLen"**:**4**,  
            **"ParameteType"**:**3**,  
            **"ParameteValue"**:**2**  
        }  
    ],  
    **"XRpc\_ReturnType"**:**3**  
}

##### 3.10.1.2.2 回复

HTTP/1.1 200 OK

Date: Fri, 15 May 2020 11:18:00 GMT

Server: NetEngine\_SimpleHttp/5.23.0.1001

Connection: close

Content-Type: application/json;charset=UTF-8

Content-Length: 127

{  
    **"XRpc\_FunctionName"**:**"XEngine\_AddMethod"**,  
    **"XRpc\_ReturnLen"**:**4**,  
    **"XRpc\_ReturnType"**:**3**,  
    **"XRpc\_ReturnValue"**:**3**  
}

### 3.10.2 命令调用

## 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 - 9:系统保留
* 10 - 16: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!

## 附录4 更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新说明 | 修改者 | 适用版本 |
| 2019.07.12 | 增加了协议规范说明  协议头说明  协议头格式  协议头字段  协议头表示方式  协议头展示效果  增加了使用协议说明  表示了主子分类说明  增加了通用协议说明  增加了日志协议  增加了验证删除协议  增加了验证注册协议  增加了验证登录协议  增加了流媒体推送协议  增加了存储传输协议  增加了存储上传协议  增加了存储查询协议  增加了存储删除协议  增加了存储下载协议  增加了心跳协议 | Qyt | V5.19 |
| 2019.07.19 | 更新了通用协议说明  增加了存储文件夹增删查协议  增加了P2XP的登录协议  增加了P2XP的内外网地址同步协议  增加了P2XP请求连接协议  增加了P2XP用户查询协议  增加了后台地址同步协议  增加了后台管理协议  增加下载协议  增加删除文件协议  增加删除文件夹协议  增加上传文件协议  增加获取文件列表协议  增加执行文件协议  增加发送消息协议  增加关闭系统协议  增加执行命令协议  增加了分包Chunked协议说明  增加了离开协议说明  增加了用户协议  增加了特别说明大类.用于解释一些情况  增加了权限级别说明  增加了负载类型说明  增加了加密类型说明 | Qyt | V5.19 |
| 2019.09.10 | 修改用户协议  修改了用户协议登录,注册,删除回复格式  增加了充值协议  增加了密码找回协议  增加了获取剩余时间协议  增加了获取公告信息协议  增加了超时协议  增加消息队列协议  增加了投递数据协议 | Qyt | V5.21 |
| 2019.09.22 | 消息队列协议  修改了投递包协议的类型  增加了获取包协议  增加了删除包协议  协议头  修改加密说明的范围错误问题  负载类型增加了XML类型  负载类型现在使用了新的字段说明了 | Qyt | V5.22 |
| 2020.04.27 | 登录协议  增加了回复确认码识别说明  流媒体协议  推流协议明确了错误和正确信息  存储协议  文件上传协议回复包增加了错误信息说明  文件下载协议完善了说明和缺少的内容  消息分发协议  修改消息分发协议注意事项 | Qyt | V6.10 |
| 2020.05.09 | 消息分发协议  增加了请求应答协议的查询域协议  增加了请求应答协议的查询主题协议  增加了域删除通知协议  增加了主题删除通知协议  增加了发布者创建通知协议  特别说明  修改协议负载类型定义说明  增加了一个权限级别字段  附录  增加了LINUX下的类型转换头说明  UDX协议  增加了UDX数据传输协议  协议规范  扩展协议头更新,增加了XTP时间字段  协议头扩展协议相关信息修改匹配  P2XP协议  修改了P2XP协议,现在不在需要专属协议  P2XP登录协议修改,简化协议 | Qyt | V6.11 |
| 2020.05.19 | 消息协议修改心跳,分包,离开协议编号  增加XRPC协议  增加了函数调用标准协议请求和返回示例  增加了函数调用HTTP协议请求和返回示例  UDX协议修改命名规范  协议修改了描述说明  增加了登录协议说明  增加了数据重传协议  增加了通知协议(序列号,滑动窗口,关闭) | Qyt | V6.12 |
| 2020.07.14 | 存储协议  增加了用户信息查询协议  修改了文件信息查询协议定义  流媒体协议  修改拉流和推流协议定义  增加了播放流和暂停流控制协议  增加了拉流和推流的通知协议 | Qyt | V6.17 |
| 2020.07.31 | 流媒体协议  状态通知协议进行了修改 | qyt | V6.19 |
| 2021.01.16 | 流媒体协议  推流和拉流协议更新增加了音视频参数  消息队列协议  消息队列负载的消息协议进行了优化  验证协议  增加网络临时验证协议 | qyt | V6.26 |
| 2021.02.17 | 后台服务协议  完成对后台服务协议适配优化  验证服务协议  更新了对公告协议的适配 | Qyt | V6.29 |
| 2021.04.01 | 协议头  保留字段进行了扩展增加了加解密标志  消息队列  消息队列协议增加了获取次数的字段  验证协议  验证协议对传输协议加解密进行了说明  特殊说明  加解密类型更新  修改负载类型说明有误的问题 | Qyt | V6.33 |
| 2021.05.03 | 全局更新  协议定义全部更新到V7版本名称  修改\_\_int64定义为\_\_int64x  验证协议  tszPassword修改为tszUserPass  文件传输协议  字段已经扩展了所属用户 | Qyt | V7.10 |
| 2021.06.25 | 存储协议  存储协议字段现在使用HASH而不是MD5  删除了不在使用的存储协议  控制协议  增加了连接协议使用说明 | Qyt | V7.14 |
| 2021.07.02 | 消息协议  修改了协议结构体  增加了创建和删除主题协议 | Qyt | V7.15 |