XEngine消息队列服务文档

目录

[XEngine消息队列服务文档 1](#_Toc29939)

[前言 4](#_Toc19469)

[阅读者 4](#_Toc1536)

[概述 4](#_Toc5459)

[相关模块 4](#_Toc32523)

[一 技术结构 4](#_Toc19772)

[1.1 TCP 4](#_Toc2377)

[1.2 HTTP 4](#_Toc18225)

[二 配置环境 4](#_Toc17792)

[2.1 WINDOWS 4](#_Toc14318)

[2.1.1 配置环境 5](#_Toc6208)

[2.1.2 编译运行 5](#_Toc4456)

[2.2 LINUX 6](#_Toc21704)

[2.2.1 环境配置 6](#_Toc18065)

[2.2.2 编译运行 6](#_Toc9256)

[2.3 版本要求 6](#_Toc32382)

[2.3.1 系统版本 6](#_Toc12896)

[2.3.2 软件需求 7](#_Toc14716)

[三 接口协议 7](#_Toc14734)

[3.1 TCP 7](#_Toc16409)

[3.1.1 投递包协议 7](#_Toc867)

[3.1.2 获取包协议 8](#_Toc13104)

[3.1.3 删除包协议 9](#_Toc31614)

[3.1.4 创建主题 10](#_Toc3432)

[3.1.5 删除主题 12](#_Toc347)

[3.1.6 订阅主题 13](#_Toc10717)

[3.1.7 通知消息 14](#_Toc24310)

[3.2 HTTP 14](#_Toc16575)

[3.2.1 请求 15](#_Toc12631)

[3.2.2 回复 15](#_Toc4648)

[3.3 WEBSOCKET 15](#_Toc15003)

[四 配置说明 16](#_Toc30849)

[4.1 服务器配置 16](#_Toc4414)

[4.1.1 基本配置 16](#_Toc10027)

[4.1.2 最大配置 16](#_Toc32427)

[4.1.3 时间配置 16](#_Toc16195)

[4.1.4 日志配置 16](#_Toc12130)

[4.1.5 数据库配置 17](#_Toc29629)

[五 高级配置 17](#_Toc10547)

[附录 17](#_Toc32558)

[附录1 类型定义 17](#_Toc24664)

[附录2 协议定义 17](#_Toc244)

[附录3 转换定义 17](#_Toc19074)

[附录4 更新历史 17](#_Toc17573)

|  |  |  |  |
| --- | --- | --- | --- |
| 文件状态：  [ ] 草稿  [√] 正式发布 | 文件名称： | XEngin消息队列服务文档 | |
| 适用版本： | V1.3 | |
| 发布日期： | 2021-07-23 | |
| 拟 制： qyt | | |

# 前言

## 阅读者

开发人员,测试人员

## **概述**

此文档包含了存储服务的相关技术说明和接口定义!

## 相关模块

此服务使用了XEngine作为开发包.要使用此服务代码必须配置安装好XEngine开发环境.

此服务还使用了第三方模块jsoncpp

# 一 技术结构

采用的是HTTP和TCP协议来作为基础通信协议.TCP和HTTP获取方式是不同的

服务器启动会绑定两个端口,一个TCP消息端口,一个HTTP消息端口

## TCP

TCP协议.TCP协议速度更快,功能更多.

## HTTP

消息通过通过POST方法提交.获取使用GET

# 二 配置环境

## 2.1 WINDOWS

需要下载XEngine.

通过VS2019来编写代码,编译,调试.

下载地址:https://gitee.com/xengine/libxengine

### 2.1.1 配置环境

下载XEngine后,如果你是压缩包下载,需要解压,解压后,添加用户环境变量

你需要在你的系统当中,添加下面两个用户环境变量

* XEngine\_Include  XEngine的头文件目录地址
* XEngine\_Library  XEngine的库目录文件地址

比如:



你还需要JSONCPP环境,你可以通过VCPKG来安装JSONCPP.

也可以自己下载:https://github.com/open-source-parsers/jsoncpp/

如果是自己下载,你需要自己安装和编译.并且在我们的项目中-属性-VC++目录 配置你的JSONCPP目录才可使用.

### 2.1.2 编译运行

在完成环境配置后.你可以进入代码目录.使用VS2019打开XEngine\_MQServiceApp.sln,然后选择x86 debug 编译.

如果环境没有错误,编译会直接成功.包含4个模块和一个EXE程序

成功后,你需要拷贝XEngine\_Release下的文件到你编译的目录下.然后拷贝依赖的XEngine模块与JSONCPP模块到你的编译目录下,运行XEngine\_MQServiceApp.exe即可.如果没有错误,你可以看到下面的界面信息



注意：你可以直接运行程序，系统会提示你需要哪些，你直接进入XEngine目录搜索.

## 2.2 LINUX

### 2.2.1 环境配置

如果使用LINUX来开发运行,那么你需要在UBUNTU或者CENTOS下面才可以使用,UBUNTU需要20.04 以上系统.CENTOS需要8.x版本(兼容STREAM版本)

在下载完毕后,你可能需要解压,解压后,在目录里面找到脚本安装文件并且执行下面的命令:

sudo XEngine\_RunEnv.sh -i 3

即可完成XEngine的环境配置.

当然,还需要JSONCPP的环境,你可以通过命令安装

Ubuntu:sudo apt install libjsoncpp-devel -y

Centos:sudo dnf install jsoncpp-devel -y

### 2.2.2 编译运行

配置完环境后,你可以编译它,打开终端,进入源码目录执行以下命令

编译:make

安装:make FLAGS=InstallAll

清理:make FLAGS=CleanAll

如果没有错误,你可以在XEngine\_Release目录下看到编译的XEngine\_MQServiceApp

然后直接在终端运行它即可.如果没有错误,你可以看到下面的信息:



## 2.3 版本要求

### 2.3.1 系统版本

最低版本要求:

WINDOWS: win7 sp1

Ubuntu:20.04

Centos:8.x

### 2.3.2 软件需求

最低版本要求:

XEngine:V7.15

JsonCpp:V1.9.4

# 三 接口协议

## 3.1 TCP

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.

### 3.1.1 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.

#### 3.1.1.1 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

#### 3.1.1.2 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;

### 3.1.2 获取包协议

#### 3.1.2.1 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;

#### 3.1.2.2 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

### 3.1.3 删除包协议

#### 3.1.3.1 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;

#### 3.1.3.2 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;

### 3.1.4 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.

#### 3.1.4.1 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;

#### 3.1.4.2 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;

### 3.1.5 删除主题

#### 3.1.5.1 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;

#### 3.1.5.2 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;

### 3.1.6 订阅主题

订阅主题允许用户在请求订阅成功后,每次主题有新内容,都会主动推送一次消息给订阅的客户端.

#### 3.1.6.1 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\_REQNOTIFY

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]; //要订阅的消息

\_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;

#### 3.1.6.2 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\_REPNOTIFY

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]; //订阅的消息

\_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;

### 3.1.7 通知消息

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\_MSGNOTIFY

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

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]; //要订阅的消息

\_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;

...body

## 3.2 HTTP

HTTP目前支持POST请求,通过负载JSON来确定请求事项

请求和回复的内容都与TCP的值相对应.

st\_Payload表示负载的数据,nPayType为0表示是字符串,1为BASE64编码,如果是1,那么服务器接受到数据就需要BASE64解码,tszPaydata为负载的具体数据.

如果没有负载内容,st\_Payload可以不存在.

在Get消息的时候序列号不能为0,必须指定

### 3.2.1 请求

{  
    **"unOperatorType"**:**7**,  
    **"unOperatorCode"**:**7001**,  
    **"st\_MQProtocol"**:{  
        **"tszMQKey"**:**"主题名"**,  
        **"nSerial"**:**0**,  
        **"nKeepTime"**:**0**,  
        **"nGetTimer"**:**0**  
    },  
    **"st\_Payload"**:{  
        **"nPayType"**:**0**,  
        **"nPayLen"**:**6**,  
        **"tszPayData"**:**"123456"**  
    }  
}

### 3.2.2 回复

{  
    **"unOperatorType"**:**7**,  
    **"unOperatorCode"**:**7001**,  
    **"wReserve"**:**0**,  
    **"st\_MQProtocol"**:{  
        **"tszMQKey"**:**"主题名"**,  
        **"nSerial"**:**0**,  
        **"nKeepTime"**:**0**,  
        **"nGetTimer"**:**0**  
    },  
    **"st\_Payload"**:{  
        **"nPayType"**:**0**,  
        **"nPayLen"**:**6**,  
        **"tszPayData"**:**"123456"**  
    }  
}

## 3.3 WEBSOCKET

WebSocket协议负载同HTTP,返回也是,WebSocket请求需要PING PONG来包活心跳,在没有数据交换的时候.

WebSocket的负载数据类型只能是TEXT.WEBSOCKET可以使用订阅模式

WEBSOCKET作为长连接,在获取消息的时候可以不设置序列号,顺序获取

# 四 配置说明

## 4.1 服务器配置

基础配置文件:XEngine\_Config.json

### 4.1.1 基本配置

* bDeamon: 1为守护进程后台运行0为前台运行
* tszTopic:默认公有主题名称
* nTCPPort:TCP通信端口
* nHttpPort:HTTP通信端口
* nWSPort:WEBSOCKET 通信端口

### 4.1.2 最大配置

XMax 配置

* MaxClient 最大允许客户端
* MaxQueue 最大允许队列
* IOThread:网络IO处理线程个数
* nTCPThread:TCP处理线程个数
* nHttpThread:HTTP处理线程个数
* nWSThread:WEBSOCKET处理线程个数

### 4.1.3 时间配置

XTime配置

* bHBTime是否启用,1为启用0不启用心跳
* nDBMonth:数据库保存时间,默认月
* nTimeCheck:检测次数
* nTCPTimeOut:TCP客户端过期时间
* nHttpTimeOut:HTTP客户端过期时间
* nWSTimeOut:WEBSOCKET客户端超时时间

### 4.1.4 日志配置

XLog 配置

* MaxSize:日志文件大小
* MaxCount:日志文件个数
* LogLeave:允许保存的级别

### 4.1.5 数据库配置

XSql 配置.需要MYSQL的支持

# 五 高级配置

# 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

## 附录4 更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新说明 | 修改者 | 适用版本 |