原 最简单的基于librtmp的示例：发布（FLV通过RTMP发布）

2014年12月26日 00:09:33 阅读数：30571

==========

最简单的基于libRTMP的示例系列文章列表：

最简单的基于librtmp的示例：接收（RTMP保存为FLV）

最简单的基于librtmp的示例：发布（FLV通过RTMP发布）

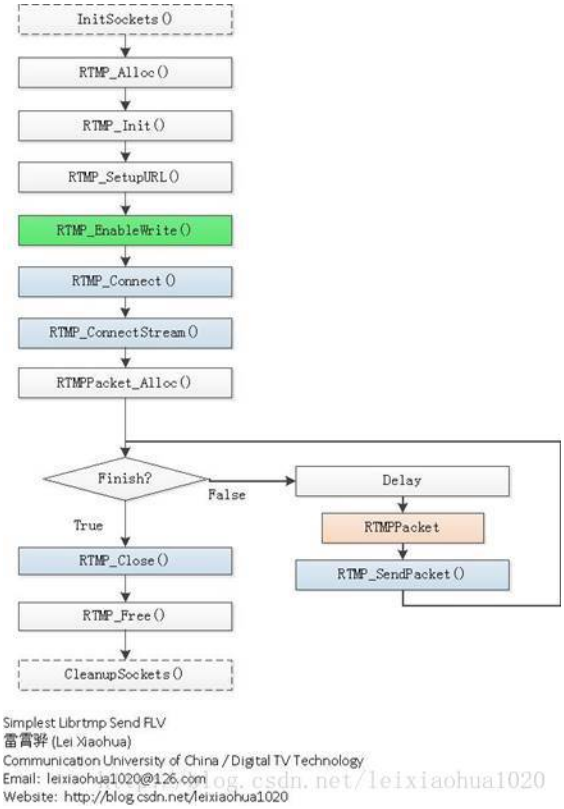
最简单的基于librtmp的示例：发布H.264（H.264通过RTMP发布）

==========

本文记录一个基于libRTMP的发布流媒体的程序：Simplest libRTMP Send FLV。该程序可以将本地FLV文件发布到RTMP流媒体服务器。是最简单的基于libRTMP的流媒体发布示例。

流程图

使用librtmp发布RTMP流的可以使用两种API：RTMP_SendPacket()和RTMP_Write()。使用RTMP_SendPacket()发布流的时候的函数执行流程图如下图所示。使用RTMP_Write()发布流的时候的函数执行流程图相差不大。



流程图中关键函数的作用如下所列：

InitSockets()：初始化Socket
RTMP_Alloc()：为结构体“RTMP”分配内存。
RTMP_Init()：初始化结构体“RTMP”中的成员变量。
RTMP_SetupURL()：设置输入的RTMP连接的URL。
RTMP_EnableWrite()：发布流的时候必须要使用。如果不使用则代表接收流。
RTMP_Connect()：建立RTMP连接，创建一个RTMP协议规范中的NetConnection。

RTMP_ConnectStream()：创建一个RTMP协议规范中的NetStream。
Delay：发布流过程中的延时，保证按正常播放速度发送数据。
RTMP_SendPacket()：发送一个RTMP数据RTMPPacket。
RTMP_Close()：关闭RTMP连接。
RTMP_Free()：释放结构体“RTMP”。
CleanupSockets()：关闭Socket。

源代码

源代码中包含了使用两种API函数RTMP_SendPacket()和RTMP_Write()发布流媒体的源代码，如下所示。

```cpp
/**
 * Simplest Librtmp Send FLV
 *
 * 雷霄骅，张晖
 * leixiaohua1020@126.com
 * zhanghuicuc@gmail.com
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 * 本程序用于将FLV格式的视音频文件使用RTMP推送至RTMP流媒体服务器。
 * This program can send local flv file to net server as a rtmp live stream.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#ifndef WIN32
#include <unistd.h>
#endif


#include "librtmp/rtmp_sys.h"
#include "librtmp/log.h"

#define HTON16(x)  ((x>>8&0xff)|(x<<8&0xff00))
#define HTON24(x)  ((x>>16&0xff)|(x<<16&0xff0000)|(x&0xff00))
#define HTON32(x)  ((x>>24&0xff)|(x>>8&0xff00)|\
        (x<<8&0xff0000)|(x<<24&0xff000000))
#define HTONTIME(x) ((x>>16&0xff)|(x<<16&0xff0000)|(x&0xff00)|(x&0xff000000))

/*read 1 byte*/
int ReadU8(uint32_t *u8,FILE*fp){
        if(fread(u8,1,1,fp)!=1)
                return 0;
        return 1;
}
/*read 2 byte*/
int ReadU16(uint32_t *u16,FILE*fp){
        if(fread(u16,2,1,fp)!=1)
                return 0;
        *u16=HTON16(*u16);
        return 1;
}
/*read 3 byte*/
int ReadU24(uint32_t *u24,FILE*fp){
        if(fread(u24,3,1,fp)!=1)
                return 0;
        *u24=HTON24(*u24);
        return 1;
}
/*read 4 byte*/
int ReadU32(uint32_t *u32,FILE*fp){
        if(fread(u32,4,1,fp)!=1)
                return 0;
        *u32=HTON32(*u32);
        return 1;
}
/*read 1 byte,and loopback 1 byte at once*/
int PeekU8(uint32_t *u8,FILE*fp){
        if(fread(u8,1,1,fp)!=1)
                return 0;
        fseek(fp,-1,SEEK_CUR);
        return 1;
}
/*read 4 byte and convert to time format*/
int ReadTime(uint32_t *utime,FILE*fp){
        if(fread(utime,4,1,fp)!=1)
                return 0;
        *utime=HTONTIME(*utime);
        return 1;
}

int InitSockets()
{
```

```
	WORD version;
	WSADATA wsaData;
	version=MAKEWORD(2,2);
	return (WSAStartup(version, &wsaData) == 0);
}

void CleanupSockets()
{
	WSACleanup();
}

//Publish using RTMP_SendPacket()
int publish_using_packet(){
	RTMP *rtmp=NULL;
	RTMPPacket *packet=NULL;
	uint32_t start_time=0;
	uint32_t now_time=0;
	//the timestamp of the previous frame
	long pre_frame_time=0;
	long lasttime=0;
	int bNextIsKey=1;
	uint32_t preTagsize=0;

	//packet attributes
	uint32_t type=0;
	uint32_t datalength=0;
	uint32_t timestamp=0;
	uint32_t streamid=0;

	FILE*fp=NULL;
	fp=fopen("cuc_ieschool.flv","rb");
	if (!fp){
		RTMP_LogPrintf("Open File Error.\n");
		CleanupSockets();
		return -1;
	}

	/* set log level */
	//RTMP_LogLevel loglvl=RTMP_LOGDEBUG;
	//RTMP_LogSetLevel(loglvl);

	if (!InitSockets()){
		RTMP_LogPrintf("Init Socket Err\n");
		return -1;
	}

	rtmp=RTMP_Alloc();
	RTMP_Init(rtmp);
	//set connection timeout,default 30s
	rtmp->Link.timeout=5;
	if(!RTMP_SetupURL(rtmp,"rtmp://localhost/publishlive/livestream"))
	{
		RTMP_Log(RTMP_LOGERROR,"SetupURL Err\n");
		RTMP_Free(rtmp);
		CleanupSockets();
		return -1;
	}

	//if unable,the AMF command would be 'play' instead of 'publish'
	RTMP_EnableWrite(rtmp);

	if (!RTMP_Connect(rtmp,NULL)){
		RTMP_Log(RTMP_LOGERROR,"Connect Err\n");
		RTMP_Free(rtmp);
		CleanupSockets();
		return -1;
	}

	if (!RTMP_ConnectStream(rtmp,0)){
		RTMP_Log(RTMP_LOGERROR,"ConnectStream Err\n");
		RTMP_Close(rtmp);
		RTMP_Free(rtmp);
		CleanupSockets();
		return -1;
	}

	packet=(RTMPPacket*)malloc(sizeof(RTMPPacket));
	RTMPPacket_Alloc(packet,1024*64);
	RTMPPacket_Reset(packet);

	packet->m_hasAbsTimestamp = 0;
	packet->m_nChannel = 0x04;
	packet->m_nInfoField2 = rtmp->m_stream_id;

	RTMP_LogPrintf("Start to send data ...\n");

	//jump over FLV Header
	fseek(fp,9,SEEK_SET);
	//jump over previousTagSizen
	fseek(fp,4,SEEK_CUR);
	start_time=RTMP_GetTime();
```

```
        while(1)
        {
                if((((now_time=RTMP_GetTime())-start_time)
                          <(pre_frame_time)) && bNextIsKey){
                        //wait for 1 sec if the send process is too fast
                        //this mechanism is not very good,need some improvement
                        if(pre_frame_time>lasttime){
                                RTMP_LogPrintf("TimeStamp:%8lu ms\n",pre_frame_time);
                                lasttime=pre_frame_time;
                        }
                        Sleep(1000);
                        continue;
                }

                //not quite the same as FLV spec
                if(!ReadU8(&type,fp))
                        break;
                if(!ReadU24(&datalength,fp))
                        break;
                if(!ReadTime(×tamp,fp))
                        break;
                if(!ReadU24(&streamid,fp))
                        break;

                if (type!=0x08&&type!=0x09){
                        //jump over non_audio and non_video frame,
                        //jump over next previousTagSizen at the same time
                        fseek(fp,datalength+4,SEEK_CUR);
                        continue;
                }

                if(fread(packet->m_body,1,datalength,fp)!=datalength)
                        break;

                packet->m_headerType = RTMP_PACKET_SIZE_LARGE;
                packet->m_nTimeStamp = timestamp;
                packet->m_packetType = type;
                packet->m_nBodySize  = datalength;
                pre_frame_time=timestamp;

                if (!RTMP_IsConnected(rtmp)){
                        RTMP_Log(RTMP_LOGERROR,"rtmp is not connect\n");
                        break;
                }
                if (!RTMP_SendPacket(rtmp,packet,0)){
                        RTMP_Log(RTMP_LOGERROR,"Send Error\n");
                        break;
                }

                if(!ReadU32(&preTagsize,fp))
                        break;

                if(!PeekU8(&type,fp))
                        break;
                if(type==0x09){
                        if(fseek(fp,11,SEEK_CUR)!=0)
                                break;
                        if(!PeekU8(&type,fp)){
                                break;
                        }
                        if(type==0x17)
                                bNextIsKey=1;
                        else
                                bNextIsKey=0;

                        fseek(fp,-11,SEEK_CUR);
                }
        }

        RTMP_LogPrintf("\nSend Data Over\n");

        if(fp)
                fclose(fp);

        if (rtmp!=NULL){
                RTMP_Close(rtmp);
                RTMP_Free(rtmp);
                rtmp=NULL;
        }
        if (packet!=NULL){
                RTMPPacket_Free(packet);
                free(packet);
                packet=NULL;
        }

        CleanupSockets();
        return 0;
}

//Publish using RTMP_Write()
int publish_using_write(){
```

```
	uint32_t start_time=0;
	uint32_t now_time=0;
	uint32_t pre_frame_time=0;
	uint32_t lasttime=0;
	int bNextIsKey=0;
	char* pFileBuf=NULL;

	//read from tag header
	uint32_t type=0;
	uint32_t datalength=0;
	uint32_t timestamp=0;

	RTMP *rtmp=NULL;

	FILE*fp=NULL;
	fp=fopen("cuc_ieschool.flv","rb");
	if (!fp){
		RTMP_LogPrintf("Open File Error.\n");
		CleanupSockets();
		return -1;
	}

	/* set log level */
	//RTMP_LogLevel loglvl=RTMP_LOGDEBUG;
	//RTMP_LogSetLevel(loglvl);

	if (!InitSockets()){
		RTMP_LogPrintf("Init Socket Err\n");
		return -1;
	}

	rtmp=RTMP_Alloc();
	RTMP_Init(rtmp);
	//set connection timeout,default 30s
	rtmp->Link.timeout=5;
	if(!RTMP_SetupURL(rtmp,"rtmp://localhost/publishlive/livestream"))
	{
		RTMP_Log(RTMP_LOGERROR,"SetupURL Err\n");
		RTMP_Free(rtmp);
		CleanupSockets();
		return -1;
	}

	RTMP_EnableWrite(rtmp);
	//1hour
	RTMP_SetBufferMS(rtmp, 3600*1000);
	if (!RTMP_Connect(rtmp,NULL)){
		RTMP_Log(RTMP_LOGERROR,"Connect Err\n");
		RTMP_Free(rtmp);
		CleanupSockets();
		return -1;
	}

	if (!RTMP_ConnectStream(rtmp,0)){
		RTMP_Log(RTMP_LOGERROR,"ConnectStream Err\n");
		RTMP_Close(rtmp);
		RTMP_Free(rtmp);
		CleanupSockets();
		return -1;
	}

	printf("Start to send data ...\n");

	//jump over FLV Header
	fseek(fp,9,SEEK_SET);
	//jump over previousTagSizen
	fseek(fp,4,SEEK_CUR);
	start_time=RTMP_GetTime();
	while(1)
	{
		if((((now_time=RTMP_GetTime())-start_time)
			<(pre_frame_time)) && bNextIsKey){
			//wait for 1 sec if the send process is too fast
			//this mechanism is not very good,need some improvement
			if(pre_frame_time>lasttime){
				RTMP_LogPrintf("TimeStamp:%8lu ms\n",pre_frame_time);
				lasttime=pre_frame_time;
			}
			Sleep(1000);
			continue;
		}

		//jump over type
		fseek(fp,1,SEEK_CUR);
		if(!ReadU24(&datalength,fp))
			break;
		if(!ReadTime(×tamp,fp))
			break;
		//jump back
		fseek(fp,-8,SEEK_CUR);

		
```

```
                pFileBuf=(char*)malloc(11+datalength+4);
                memset(pFileBuf,0,11+datalength+4);
                if(fread(pFileBuf,1,11+datalength+4,fp)!=(11+datalength+4))
                        break;

                pre_frame_time=timestamp;

                if (!RTMP_IsConnected(rtmp)){
                        RTMP_Log(RTMP_LOGERROR,"rtmp is not connect\n");
                        break;
                }
                if (!RTMP_Write(rtmp,pFileBuf,11+datalength+4)){
                        RTMP_Log(RTMP_LOGERROR,"Rtmp Write Error\n");
                        break;
                }

                free(pFileBuf);
                pFileBuf=NULL;

                if(!PeekU8(&type,fp))
                        break;
                if(type==0x09){
                        if(fseek(fp,11,SEEK_CUR)!=0)
                                break;
                        if(!PeekU8(&type,fp)){
                                break;
                        }
                        if(type==0x17)
                                bNextIsKey=1;
                        else
                                bNextIsKey=0;
                        fseek(fp,-11,SEEK_CUR);
                }
        }

        RTMP_LogPrintf("\nSend Data Over\n");

        if(fp)
                fclose(fp);

        if (rtmp!=NULL){
                RTMP_Close(rtmp);
                RTMP_Free(rtmp);
                rtmp=NULL;
        }

        if(pFileBuf){
                free(pFileBuf);
                pFileBuf=NULL;
        }

        CleanupSockets();
        return 0;
}

int main(int argc, char* argv[]){
        //2 Methods:
        publish_using_packet();
        //publish_using_write();
        return 0;
}
```

运行结果

程序运行后，会将“cuc_ieschool.flv”文件以直播流的形式发布到“rtmp://localhost/publishlive/livestream”的URL。修改文件名称和RTMP的URL可以实现将任意flv文件发布到任意RTMP的URL。

下载

Simplest LibRTMP Example

项目主页

SourceForge： https://sourceforge.net/projects/simplestlibrtmpexample/

Github： https://github.com/leixiaohua1020/simplest_librtmp_example

开源中国： http://git.oschina.net/leixiaohua1020/simplest_librtmp_example

CSDN下载： http://download.csdn.net/detail/leixiaohua1020/8291757

本工程包含了LibRTMP的使用示例，包含如下子工程：
simplest_librtmp_receive: 接收RTMP流媒体并在本地保存成FLV格式的文件。
simplest_librtmp_send_flv: 将FLV格式的视音频文件使用RTMP推送至RTMP流媒体服务器。
simplest_librtmp_send264: 将内存中的H.264数据推送至RTMP流媒体服务器。

文章标签： flv RTMP 流媒体 libRTMP

个人分类： 我的开源项目 libRTMP