RTP打包与发送H.264实时码流

由于项目要求在DM6467T平台上添加实时RTP打包发送模块，这才找了找有没有人分享 这方面的经验。这里需要感谢网友：yanyuan9527，他写的文章对我帮助很大，可以说让一个完全小白的人了解了RTP打包，链接在此：http://www.chinavideo.org/forum.php?mod=viewthread&tid=7575

一、请大家阅读上面提到的文章，我这里就不详细写了，读了之后应该对RTP打包有一定了解了。不过那篇文章是在windows下实现的，我要说的是linux。首先说linux与windows下socket的几点区别：1.linux下的socket不用初始换。2.linux下定义socketfd直接是Int型，而windows下是SOCKET结构体。下图是linux和windows下socket的区别

```
1)头文件
windows下winsock.h/winsock2.h
linux下sys/socket.h
错误处理：errno.h

2)初始化
windows下需要用WSAStartup
linux下不需要

3)关闭socket
windows下closesocket(...)
linux下close(...)

4)类型
windows下SOCKET
linux下int
如我用到的一些宏：
#ifdef WIN32
typedef int socklen_t;
typedef int ssize_t;
#endif

#ifdef __LINUX__
typedef int SOCKET;
typedef unsigned char BYTE;
typedef unsigned long DWORD;
#define FALSE 0
#define SOCKET_ERROR (-1)
#endif
```

```
5)获取错误码
windows下getlasterror()/WSAGetLastError()
linux下errno变量

6)设置非阻塞
windows下ioctlsocket()
linux下fcntl() <fcntl.h>

7)send函数最后一个参数
windows下一般设置为0
linux下最好设置为MSG_NOSIGNAL，如果不设置，在发送出错后有可 能会导致程序退出
。

8)毫秒级时间获取
windows下GetTickCount()
linux下gettimeofday()

3、多线程
多线程：(win)process.h --》(linux)pthread.h
_beginthread --> pthread_create
_endthread --> pthread_exit
```

二、在linux系统下移植好那篇文章提供的源代码，应该就可以跑通发包了，可以用抓包工具Wireshark抓下包试试。下面上代码：

1.rtp.h

```
// MPEG2RTP.h
#include <stdio.h>
#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

//#include "mem.h"

//


#define PACKET_BUFFER_END            (unsigned int)0x00000000


#define MAX_RTP_PKT_LENGTH     1400

#define DEST_IP                "192.168.0.30"
#define DEST_PORT            1234

#define H264                    96

typedef int SOCKET;

typedef struct
{
    /**//* byte 0 */
    unsigned char csrc_len:4;        /**//* expect 0 */
    unsigned char extension:1;        /**//* expect 1, see RTP_OP below */
```

相关文章

使用ORTP发送原始H.264码流
将H264码流打包成RTP包
RTP(载荷H264码流)解包与封包
RTP-H264码流
3、RTP传输H.264
H.264打包MPEG-TS流
RTP协议解析和H264码流提取
码流格式：ANNEX-B，AVCC(H.264)与HVCC(H.265)，EXTRADATA详解
H264帧通过RTP打包
SPRINGBOOT开发环境安装(JDK+MAVEN+INTELLIJ IDEA)

热门文章

JDBC自建工具类
数据增强
我的一个博客文章
CF - C. PLASTICINE ZEBRA
MATPLOTLIB.PYPLOT画子图的两种方式(面向对象和PYPLOT)
DOCKER-COMPOSE搭建REDIS主从复制
SPRINGCLOUD与CONSUL整合实现负载均衡时不生效的问题
易语言大漠插件模块制作后台找字FINDSTREXS和FINDSTRFASTEXS
RETROFIT--请求方法那些事儿
ANDROID全局异常捕获

推荐文章

DUBBO+ZOOKEEPER示例代码_SPRING|SPRINGBOOT集成DUBBO

ANDROID 进程模块获取
PYTHON-元组AND字典
学习笔记之ANDROID篇——SERVICE(2.0)
C++之STRING类
ANDROID STUDIO:通过ARTIFACTORY搭建本地仓库优化编译速度
转，JAVA H5 复杂TABLE导出EXCEL并下载
WDATEPICKER时间插件
基于GEC6818智能家居的实现--点亮LED灯
【LEETCODE每日一题】[困难]57. 插入区间

相关标签
ANDROID
H.264
编码
C语言
RTP 流媒体
音视频开发
RTSP专栏
MPEGTS
编码解码
ANNEX-B


```
    unsigned char padding:1;        /**//* expect 0 */
    unsigned char version:2;        /**//* expect 2 */
    /**//* byte 1 */
    unsigned char payload:7;        /**//* RTP_PAYLOAD_RTSP */
    unsigned char marker:1;        /**//* expect 1 */
    /**//* bytes 2, 3 */
    unsigned short seq_no;
    /**//* bytes 4-7 */
    unsigned  long timestamp;
    /**//* bytes 8-11 */
    unsigned long ssrc;            /**//* stream number is used here. */
} RTP_FIXED_HEADER;

typedef struct {
    //byte 0
        unsigned char TYPE:5;
    unsigned char NRI:2;
        unsigned char F:1;

} NALU_HEADER; /**//* 1 BYTES */

typedef struct {
    //byte 0
    unsigned char TYPE:5;
        unsigned char NRI:2;
        unsigned char F:1;


} FU_INDICATOR; /**//* 1 BYTES */

typedef struct {
    //byte 0
    unsigned char TYPE:5;
        unsigned char R:1;
        unsigned char E:1;
        unsigned char S:1;
} FU_HEADER; /**//* 1 BYTES */



//BOOL InitWinsock();
```

2. rtp.c

```
// NALDecoder.cpp : Defines the entry point for the console application.
//


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include "rtp.h"

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

typedef struct
{
  int startcodeprefix_len;      //! 4 for parameter sets and first slice in picture, 3 for everything else (suggested)
  unsigned len;                 //! Length of the NAL unit (Excluding the start code, which does not belong to the NALU)
  unsigned max_size;            //! Nal Unit Buffer size
  int forbidden_bit;            //! should be always FALSE
  int nal_reference_idc;        //! NALU_PRIORITY_xxxx
  int nal_unit_type;            //! NALU_TYPE_xxxx
  char *buf;                    //! contains the first byte followed by the EBSP
  unsigned short lost_packets;  //! true, if packet loss is detected
} NALU_t;

FILE *bits = NULL;                //!< the bit stream file
static int FindStartCode2 (unsigned char *Buf);//查找开始字符0x000001
static int FindStartCode3 (unsigned char *Buf);//查找开始字符0x00000001
//static bool flag = true;
static int info2=0, info3=0;
RTP_FIXED_HEADER        *rtp_hdr;

NALU_HEADER   *nalu_hdr;
FU_INDICATOR *fu_ind;
FU_HEADER   *fu_hdr;

/*BOOL InitWinsock()
{
    int Error;
    WORD VersionRequested;
    WSADATA WsaData;
```

```
    VersionRequested=MAKEWORD(2,2);
    Error=WSAStartup(VersionRequested,&WsaData); //启动WinSock2
    if(Error!=0)
    {
        return FALSE;
    }
    else
    {
        if(LOBYTE(WsaData.wVersion)!=2||HIBYTE(WsaData.wHighVersion)!=2)
        {
            WSACleanup();
            return FALSE;
        }

    }
    return TRUE;
}*/

//为NALU_t结构体分配内存空间
NALU_t *AllocNALU(int buffersize)
{
  NALU_t *n;

  if ((n = (NALU_t*)calloc (1, sizeof (NALU_t))) == NULL)
  {
          printf("AllocNALU: n");
          exit(0);
  }

  n->max_size=buffersize;

  if ((n->buf = (char*)calloc (buffersize, sizeof (char))) == NULL)
  {
    free (n);
    printf ("AllocNALU: n->buf");
        exit(0);
  }

  return n;
}
//释放
void FreeNALU(NALU_t *n)
{
  if (n)
  {
    if (n->buf)
    {
      free(n->buf);
      n->buf=NULL;
    }
    free (n);
  }
}

void OpenBitstreamFile (char *fn)
{
  if (NULL == (bits=fopen(fn, "rb")))
  {
          printf("open file error\n");
          exit(0);
  }
}
//这个函数输入为一个NAL结构体，主要功能为得到一个完整的NALU并保存在NALU_t的buf中，获取他的长度，填充F,IDC,TYPE位。
//并且返回两个开始字符之间间隔的字节数，即包含有前缀的NALU的长度
int GetAnnexbNALU (NALU_t *nalu)
{
  int pos = 0;
  int StartCodeFound, rewind;
  unsigned char *Buf;

  if ((Buf = (unsigned char*)calloc (nalu->max_size , sizeof(char))) == NULL)
          printf ("GetAnnexbNALU: Could not allocate Buf memory\n");

  nalu->startcodeprefix_len=3;//初始化码流序列的开始字符为3个字节

   if (3 != fread (Buf, 1, 3, bits))//从码流中读3个字节
           {
                free(Buf);
                return 0;
           }
    info2 = FindStartCode2 (Buf);//判断是否为0x000001
    if(info2 != 1)
    {
        //如果不是，再读一个字节
     if(1 != fread(Buf+3, 1, 1, bits))//读一个字节
                {
                 free(Buf);
                 return 0;
                }
     info3 = FindStartCode3 (Buf);//判断是否为0x00000001
```

```
    if (info3 != 1)//如果不是，返回-1
            {
                 free(Buf);
                 return -1;
                }
    else
                {
                //如果是0x00000001,得到开始前缀为4个字节
        p o s    =  4;
     n a l u - > s t a r t c o d e p r e f i x _ l e n  = 4;
                }
   }

   else
           {
           //如果是0x000001,得到开始前缀为3个字节
   n a l u - > s t a r t c o d e p r e f i x _ l e n  = 3;
      p o s    =  3;
           }
   //查找下一个开始字符的标志位
   StartCodeFound = 0;
   info2 = 0;
   info3 = 0;

  while (!StartCodeFound)
  {
    if (feof (bits))//判断是否到了文件尾
    {
      nalu->len = (pos-1)-nalu->startcodeprefix_len;
      memcpy (nalu->buf, &Buf[nalu->startcodeprefix_len], nalu->len);
      nalu->forbidden_bit = nalu->buf[0] & 0x80; //1 bit
    nalu->nal_reference_idc = nalu->buf[0] & 0x60; // 2 bit
    nalu->nal_unit_type = (nalu->buf[0]) & 0x1f;// 5 bit
      free(Buf);
      return pos-1;
    }
    Buf[pos++] = fgetc (bits);//读一个字节到BUF中
    info3 = FindStartCode3(&Buf[pos-4]);//判断是否为0x00000001
    if(info3 != 1)
      info2 = FindStartCode2(&Buf[pos-3]);//判断是否为0x000001
    StartCodeFound = (info2 == 1 || info3 == 1);
  }



  // Here, we have found another start code (and read length of startcode bytes more than we should
  // have.  Hence, go back in the file
  rewind = (info3 == 1)? -4 : -3;

  if (0 != fseek (bits, rewind, SEEK_CUR))//把文件指针指向前一个NALU的末尾
  {
    free(Buf);
        printf("GetAnnexbNALU: Cannot fseek in the bit stream file");
  }

  // Here the Start code, the complete NALU, and the next start code is in the Buf.
  // The size of Buf is pos, pos+rewind are the number of bytes excluding the next
  // start code, and (pos+rewind)-startcodeprefix_len is the size of the NALU excluding the start
code

  nalu->len = (pos+rewind)-nalu->startcodeprefix_len;
  memcpy (nalu->buf, &Buf[nalu->startcodeprefix_len], nalu->len);//拷贝一个完整NALU，不拷贝起始前缀0x0
00001或0x00000001
  nalu->forbidden_bit = nalu->buf[0] & 0x80; //1 bit
  nalu->nal_reference_idc = nalu->buf[0] & 0x60; // 2 bit
  nalu->nal_unit_type = (nalu->buf[0]) & 0x1f;// 5 bit
  free(Buf);

  return (pos+rewind);//返回两个开始字符之间间隔的字节数，即包含有前缀的NALU的长度
}
//输出NALU长度和TYPE
void dump(NALU_t *n)
{
        if (!n)return;
        //printf("a new nal:");
        printf(" len: %d  ", n->len);
        printf("nal_unit_type: %x\n", n->nal_unit_type);
}

int main(int argc, char* argv[])
{
        //FILE *stream;
        //stream=fopen("Test.264", "wb");

 O p e n B i t s t r e a m F i l e ("./-a.264");//打开264文件，并将文件指针赋给bits,在此修改文件名实现打开别的264文
件。
  N A L U _ t  * n ;
        char* nalu_payload;
        char sendbuf[1500];

        unsigned short seq_num =0;
```

```
        //printf("seq_num=%d\n",seq_num);//added by
        int bytes=0;
        //InitWinsock(); //初始化套接字库
  SOCKET     socket1;
        struct sockaddr_in server;
        int len =sizeof(server);
        float framerate=25;
        unsigned int timestamp_increase=0,ts_current=0;
  timestamp_increase=(unsigned int)(90000.0 / framerate); //+0.5);

  server.sin_family=AF_INET;
    server.sin_port=htons(DEST_PORT);
    server.sin_addr.s_addr=inet_addr(DEST_IP);
    socket1=socket(AF_INET,SOCK_DGRAM,0);
    connect(socket1, (const struct sockaddr *)&server, len) ;//申请UDP套接字
  n = AllocNALU(8000000);//为结构体nalu_t及其成员buf分配空间。返回值为指向nalu_t存储空间的指针



        while(!feof(bits))
        {
    GetAnnexbNALU(n);//每执行一次，文件的指针指向本次找到的NALU的末尾，下一个位置即为下个NALU的起始码0x000001
     dump(n);//输出NALU长度和TYPE

                memset(sendbuf,0,1500);//清空sendbuf;此时会将上次的时间戳清空，因此需要ts_current来保存上次的时间戳值
        //rtp固定包头，为12字节,该句将sendbuf[0]的地址赋给rtp_hdr，以后对rtp_hdr的写入操作将直接写入sendbuf。
   rtp_hdr =(RTP_FIXED_HEADER*)&sendbuf[0];
                //设置RTP HEADER，
    rtp_hdr->payload     = H264;  //负载类型号，
    rtp_hdr->version     = 2;  //版本号，此版本固定为2
    rtp_hdr->marker      = 0;   //标志位，由具体协议规定其值。
        rtp_hdr->ssrc       = htonl(10);    //随机指定为10，并且在本RTP会话中全局唯一

        //      当一个NALU小于1400字节的时候，采用一个单RTP包发送
                if(n->len<=1400)
           {
                    //设置rtp M 位；
      rtp_hdr->marker=1;
     rtp_hdr->seq_no     = htons(seq_num ++); //***，每发送一个RTP包增1
                    //设置NALU HEADER,并将这个HEADER填入sendbuf[12]
     nalu_hdr =(NALU_HEADER*)&sendbuf[12]; //将sendbuf[12]的地址赋给nalu_hdr，之后对nalu_hdr的写入就将写入sendbuf中；
     nalu_hdr->F=n->forbidden_bit;
     nalu_hdr->NRI=n->nal_reference_idc>>5;//有效数据在n->nal_reference_idc的第6，7位，需要右移5位才能将其值赋给nalu_hdr->NRI。
     nalu_hdr->TYPE=n->nal_unit_type;

      nalu_payload=&sendbuf[13];//同理将sendbuf[13]赋给nalu_payload
                    memcpy(nalu_payload,n->buf+1,n->len-1);//去掉nalu头的nalu剩余内容写入sendbuf[13]开始的字符串。

     ts_current=ts_current+timestamp_increase;
     rtp_hdr->timestamp=htonl(ts_current);
       bytes=n->len   + 13     ;                                    //获得sendbuf的长度,为nalu的长度(包含NALU头但除去起始前缀)加上rtp_header的固定长度12字节
     send( socket1, sendbuf, bytes, 0 );//发送rtp包
                    //sleep(1);
//fwrite(sendbuf,bytes, 1, stream);
                }

                else if(n->len>1400)
                {
                    //得到该nalu需要用多少长度为1400字节的RTP包来发送
                    int k=0,l=0;
         k=n->len/1400;//需要k个1400字节的RTP包
         l=n->len%1400;//最后一个RTP包的需要装载的字节数
                    int t=0;//用于指示当前发送的是第几个分片RTP包
     ts_current=ts_current+timestamp_increase;
     rtp_hdr->timestamp=htonl(ts_current);
                    while(t<=k)
                    {
       rtp_hdr->seq_no  = htons(seq_num ++); //***，每发送一个RTP包增1
                        if(!t)//发送一个需要分片的NALU的第一个分片，置FU HEADER的S位
                        {
                            //设置rtp M 位；
             rtp_hdr->marker=0;
                            //设置FU INDICATOR,并将这个HEADER填入sendbuf[12]
          fu_ind  =(FU_INDICATOR*)&sendbuf[12]; //将sendbuf[12]的地址赋给fu_ind，之后对fu_ind的写入就将写入sendbuf中；
           fu_ind->F=n->forbidden_bit;
          fu_ind->NRI=n->nal_reference_idc>>5;
               fu_ind->TYPE=28;

                            //设置FU HEADER,并将这个HEADER填入sendbuf[13]
          fu_hdr  =(FU_HEADER*)&sendbuf[13];
                fu_hdr->E=0;
                fu_hdr->R=0;
                fu_hdr->S=1;
          fu_hdr->TYPE=n->nal_unit_type;

```

```
306.
307.           n a l u _ p a y l o a d = & s e n d b u f [14];//同理将sendbuf[14]赋给nalu_paylo
ad
308.                                   memcpy(nalu_payload,n->buf+1,1400);//去掉NALU头
309.
310.                     b   y   t   e   s   =1400+14;                                   //
获得sendbuf的长度,为nalu的长度(除去起始前缀和NALU头)加上rtp_header, fu_ind, fu_hdr的固定长度14字节
311.           s e n d (   s o c k e t 1 ,   s e n d b u f ,   b y t e s ,  0 );//发送rtp包
312. //fwrite(sendbuf,bytes, 1, stream);
313.                                   //sleep(1);
314.                         t   +   +   ;
315.
316.                             }
317.                             //发送一个需要分片的NALU的非第一个分片，清零FU HEADER的S位，如果该分片是该
NALU的最后一个分片，置FU HEADER的E位
318.                             else if(k==t)//发送的是最后一个分片，注意最后一个分片的长度可能超过1400字
节(当l>1386时)。
319.                             {
320.
321.                                   //设置rtp M 位;当前传输的是最后一个分片时该位置1
322.             r t p _ h d r - > m a r k e r =1;
323.                                   //设置FU INDICATOR,并将这个HEADER填入sendbuf[12]
324.           f u _ i n d   = ( F U _ I N D I C A T O R * ) & s e n d b u f [12]; //将sendbuf[12]的地址
赋给fu_ind，之后对fu_ind的写入就将写入sendbuf中；
325.            f u _ i n d - > F = n - > f o r b i d d e n _ b i t ;
326.           f u _ i n d - > N R I = n - > n a l _ r e f e r e n c e _ i d c > >5;
327.                f u _ i n d - > T Y P E  =28;
328.
329.                                   //设置FU HEADER,并将这个HEADER填入sendbuf[13]
330.           f u _ h d r   = ( F U _ H E A D E R * ) & s e n d b u f [13];
331.                 f u _ h d r - > R  =0;
332.                 f u _ h d r - > S  =0;
333.           f u _ h d r - > T Y P E = n - > n a l _ u n i t _ t y p e ;
334.                 f u _ h d r - > E  =1;
335.
336.            n a l u _ p a y l o a d = & s e n d b u f [14];//同理将sendbuf[14]的地址赋给nalu_
payload
337.                                   memcpy(nalu_payload,n->buf+t*1400+1,l-1);//将nalu最后剩余的
l-1(去掉了一个字节的NALU头)字节内容写入sendbuf[14]开始的字符串。
338.                     b   y   t   e   s   =   l-1+14;          //获得sendbuf的长度,为剩余nalu的长度l
-1加上rtp_header, FU_INDICATOR,FU_HEADER三个包头共14字节
339.           s e n d (   s o c k e t 1 ,   s e n d b u f ,   b y t e s ,  0 );//发送rtp包
340. //fwrite(sendbuf,bytes, 1, stream);
341.                         t   +   +   ;
342.                                   //sleep(1);
343.                             }
344.                             else if(t<k&&0!=t)
345.                             {
346.                                   //设置rtp M 位;
347.               r t p _ h d r - > m a r k e r =0;
348.                                   //设置FU INDICATOR,并将这个HEADER填入sendbuf[12]
349.           f u _ i n d   = ( F U _ I N D I C A T O R * ) & s e n d b u f [12]; //将sendbuf[12]的地址
赋给fu_ind，之后对fu_ind的写入就将写入sendbuf中；
350.            f u _ i n d - > F = n - > f o r b i d d e n _ b i t ;
351.           f u _ i n d - > N R I = n - > n a l _ r e f e r e n c e _ i d c > >5;
352.                f u _ i n d - > T Y P E  =28;
353.
354.                                   //设置FU HEADER,并将这个HEADER填入sendbuf[13]
355.           f u _ h d r   = ( F U _ H E A D E R * ) & s e n d b u f [13];
356.                                   //fu_hdr->E=0;
357.                 f u _ h d r - > R  =0;
358.                 f u _ h d r - > S  =0;
359.                 f u _ h d r - > E  =0;
360.           f u _ h d r - > T Y P E = n - > n a l _ u n i t _ t y p e ;
361.
362.            n a l u _ p a y l o a d = & s e n d b u f [14];//同理将sendbuf[14]的地址赋给nalu_
payload
363.                                   memcpy(nalu_payload,n->buf+t*1400+1,1400);//去掉起始前缀的na
lu剩余内容写入sendbuf[14]开始的字符串。
364.                     b   y   t   e   s   =1400+14;                                   //
获得sendbuf的长度,为nalu的长度(除去原NALU头)加上rtp_header, fu_ind, fu_hdr的固定长度14字节
365.           s e n d (   s o c k e t 1 ,   s e n d b u f ,   b y t e s ,  0 );//发送rtp包
366. //fwrite(sendbuf,bytes, 1, stream);
367.                                   //sleep(1);
368.                         t   +   +   ;
369.                             }
370.                     }
371.                 }
372.         //usleep(40000);
373.         }
374.   F r e e N A L U ( n ) ;
375.         return 0;
376. }
377.
378. static int FindStartCode2 (unsigned char *Buf)
379. {
380.  if(Buf[0]!=0 || Buf[1]!=0 || Buf[2] !=1) return 0; //判断是否为0x000001,如果是返回1
381.  else return 1;
382. }
383.
384. static int FindStartCode3 (unsigned char *Buf)
385. {
386.  if(Buf[0]!=0 || Buf[1]!=0 || Buf[2] !=0 || Buf[3] !=1) return 0;//判断是否为0x00000001,如果是返回1
387.  else return 1;
```

```
388.  }
```

三、程序跑通之后下一步，就是要移植到DM6467T下encode工程中进行实时打包，也就是编码一帧完成后直接RTP打包，而不是读取文件流。在encode工程中有个writer线程，里面的fwrite函数就是writer线程的核心，这里我将fwrite替换成rtp()打包函数。

列出几点注意事项：

1. 移植过程中将所有对文件的操作改成对BUFFER指针的操作

2. 注意对RTP头中seq_no的修改

3. 注意对时间戳的修改

4. 由于实时发送需要不停地调用rtp打包程序，就需要将建立socket的部分程序移到循环之外。这是因为linux的文件操作符是有限的，不停地创建会用光文件操作符，而对socket的close不能立即释放资源，会有延时，即使close也解决不了问题（至少我没能解决，如果有解决的朋友希望不吝赐教）。

第三部分只是简单列出移植过程中所遇到的问题以及解决思路，未上详尽解释以及代码，不过网上应该都有具体方法，有问题也可以留言讨论，我用的是UDP包。（关于讨论的事非常抱歉，由于楼主不做那个项目很久了，基本忘光了，不过近期会贴上当时的代码）

将程序移植完成以后通过VLC实时解码，可能还会出现1秒钟的延迟，那是由于VLC默认的网络缓存时间是1000ms，网络条件好的可以改成300ms，延时将大大减小。

附件内容是将文中所提到的网友的windows程序做简单修改移植linux下的RTP打包发送程序：点击打开链接

图像增强库ALBUMENTATIONS（一）

1.大体的认识、有用没用的bb 官网：https://albumentations.ai/ github：https://github.com/albumentations-team/albumentations 例子：https://github.com/albumentations-team/albumentations_examples 特点：分类、目标检测、分割等任务都支持增强，与pyto...

SWITCH语句练习题

练习题一、写一个switch语句，不生产大表也不生产小表，贴出对应的反汇编 反汇编代码： 练习二、写一个switch语句，只生成大表,贴出对应的反汇编 反汇编代码： 练习题三、写一个switch语句，生成大表和小表,贴出对应的反汇编 反汇编代码： switch语句总结 编译器会根据相应的条件生成多种代码1.2.3. 1、当条件数量不多或者条件差值太乱时就会生成第一种代码判断跳转判断条件的代码。 2...

.NET STANDARD 来日苦短去日长

作者：Richard 翻译：精致码农-王亮 原文：http://dwz.win/Q4h 自从 .NET 5 宣贯以来，很多人都在问这对 .NET Standard 意味着什么，它是否仍然重要。在这篇文章中，我将解释 .NET 5 是如何改进代码共用并取代 .NET Standard 的，我还将介绍什么情况下你仍然需要 .NET Standard。 概要 .NET 5 将是一个具有统一功能和 API...