**FFMPEG源代码分析**

目录：

Avformat\_open\_input

Avformat\_find\_stream\_info

setDataSource中给ISO文件加上bluray://协议头。

**1、avformat\_open\_input**

参数AVFormatContext指针为NULL，执行s = avformat\_alloc\_context()，得到默认的AVFormatContext实例，具体填充了什么数据？

**1.1、avformat\_alloc\_context**

ic = av\_malloc(sizeof(AVFormatContext));分配AVFormatContext堆空间

avformat\_get\_context\_defaults(ic); AVFormatContext结构体填充

{

memset(s, 0, sizeof(AVFormatContext)); //对之前分配的内存清零

s->av\_class = &av\_format\_context\_class; //ffmepg设计技巧，用来操作avformat\_options数据，av\_class必须放在类定义的第一位，方便找到各个元素的偏移地址。

s->io\_open = io\_open\_default; //得到aviocontext urlcontext URLProtocol

s->io\_close = io\_close\_default;

av\_opt\_set\_defaults(s); //给av\_class中的option avformat\_options填充默认值，默认值在option定义时已经指定，例如：

static const AVOption avformat\_options[] = {

{"avioflags", NULL, OFFSET(avio\_flags), AV\_OPT\_TYPE\_FLAGS, {.i64 = DEFAULT }, INT\_MIN, INT\_MAX, D|E, "avioflags"},

…

{NULL},

};

}

static const AVClass av\_format\_context\_class = {

.class\_name = "AVFormatContext",

.item\_name = format\_to\_name,

.option = avformat\_options,

.version = LIBAVUTIL\_VERSION\_INT,

.child\_next = format\_child\_next,

.child\_class\_next = format\_child\_class\_next,

.category = AV\_CLASS\_CATEGORY\_MUXER,

.get\_category = get\_category,

};

AVFormatInternal创建和填充初始值

{

ic->internal = av\_mallocz(sizeof(\*ic->internal));

ic->internal->offset = AV\_NOPTS\_VALUE;// 0x8000000000000000

ic->internal->raw\_packet\_buffer\_remaining\_size = RAW\_PACKET\_BUFFER\_SIZE; //2500000

}

**1.2、ret = init\_input(s, filename, &tmp)**

如果有pb（AVIOContext指针），判断是否有AVInputFormat，没有则需要探测av\_probe\_input\_buffer2，后函数直接返回。

如果没有pb，则判断是否有AVInputFormat，没有则av\_probe\_input\_format2，找到AVInputFormat后函数直接返回。

上面探测失败继续走到io\_open和av\_probe\_input\_buffer2（改变参数再来一次）。

**1.2.1、av\_probe\_input\_format2(AVProbeData \*pd, int is\_opened, int \*score\_max)**

av\_probe\_input\_format3

av\_iformat\_next获取一个iformat（demuxer）

fmt1->read\_probe(&lpd)demuxer探测函数根据uri打分，probe函数读取媒体流的前4个字节，判断'F'、'L'、'V'等标识，符合打满分100，

如果没有probe函数，则根据extensions与后缀名比对，符合打分50

根据mime\_type判断，符合打分75

遍历iformat（demuxer）后得到打分最高的iformat作为demuxer

**1.2.2、s->io\_open(s, &s->pb, filename, AVIO\_FLAG\_READ | s->avio\_flags, options)**

io\_open\_default

{

ffio\_open\_whitelist(pb, url, flags, &s->interrupt\_callback, options, s->protocol\_whitelist)

}

int ffio\_open\_whitelist(AVIOContext \*\*s, const char \*filename, int flags,

const AVIOInterruptCB \*int\_cb, AVDictionary \*\*options,

const char \*whitelist

)

{

URLContext \*h;

err = ffurl\_open\_whitelist(&h, filename, flags, int\_cb, options, whitelist);

err = ffio\_fdopen(s, h);

}

ffurl\_open\_whitelist

{

ffurl\_alloc(puc, filename, flags, int\_cb)

{

p = url\_find\_protocol(filename) //根据filename中的协议头找到protocol，这里是ff\_bluray\_protocol

url\_alloc\_for\_protocol(puc, p, filename, flags, int\_cb)

创建URLContext \*uc，分配堆内存，并填充数据，初始化protocol私有结构体。

uc->av\_class = &ffurl\_context\_class;

const AVClass ffurl\_context\_class = {

.class\_name = "URLContext",

.item\_name = urlcontext\_to\_name,

.option = options, //这里只有一个数据 protocol\_whitelist，字符串，默认是空。

.version = LIBAVUTIL\_VERSION\_INT,

.child\_next = urlcontext\_child\_next,

.child\_class\_next = urlcontext\_child\_class\_next,

};

strcpy(uc->filename, filename);

uc->prot = up;

uc->flags = flags;

\*(const AVClass \*\*)uc->priv\_data = up->priv\_data\_class; // bluray\_context\_class

av\_opt\_set\_defaults(uc->priv\_data); //初始化bluray\_context\_class的option，即bluray\_context初始化

} ffurl\_alloc这里得到URLProtocol和URLContext，URLContext中的prot既是找到的URLProtocol。

ffurl\_connect(\*puc, options)

{

uc->protocol\_whitelist = uc->prot->default\_whitelist

uc->prot->url\_open(uc, uc->filename, uc->flags)，这里调用bluray\_open，得到BLURAY \*bd、main playlist，成功则返回0，protocol初始化

uc->is\_connected = 1;

return 0;

}

}

ffio\_fdopen(s, h)

{ 主要是得到AVIOContext pb

avio\_alloc\_context(buffer, buffer\_size, h->flags & AVIO\_FLAG\_WRITE, h,

(int (\*)(void \*, uint8\_t \*, int)) ffurl\_read,

(int (\*)(void \*, uint8\_t \*, int)) ffurl\_write,

(int64\_t (\*)(void \*, int64\_t, int)) ffurl\_seek);

{

s->buffer = buffer;

s->buffer\_size = buffer\_size; 默认值32768字节

s->opaque = opaque; // URLContext \*h

s->buf\_end = s->buffer;

s->write\_flag = 0;

s->write\_packet = ffurl\_write;

s->read\_packet = ffurl\_read; //avio.c中

s->seek = ffurl\_seek;

s->seekable = seek ? AVIO\_SEEKABLE\_NORMAL : 0;

s->short\_seek\_threshold = SHORT\_SEEK\_THRESHOLD; (256 \* 1024)

s->read\_pause = NULL;

s->read\_seek = NULL;

}

(\*s)->protocol\_whitelist = av\_strdup(h->protocol\_whitelist);

(\*s)->seekable = h->is\_streamed ? 0 : AVIO\_SEEKABLE\_NORMAL;

(\*s)->max\_packet\_size = max\_packet\_size; // 0

(\*s)->read\_pause = (int (\*)(void \*, int))h->prot->url\_read\_pause; //空

(\*s)->read\_seek = (int64\_t (\*)(void \*, int, int64\_t, int))h->prot->url\_read\_seek; //空

(\*s)->av\_class = &ff\_avio\_class; //option只有一个protocol\_whitelist 字符串

}到这里得到了AVIOContext，成员opaque是URLContext，并且封装了ffurl\_read、ffurl\_seek函数指针，调用protocol中的read、seek、open等方法。

**1.2.3、av\_probe\_input\_buffer2**

av\_probe\_input\_buffer2(s->pb, &s->iformat, filename, s, 0, s->format\_probesize);

probe\_size从2048到(1 << 20)，默认情况下。调用avio\_read通过protocol的read packet读取数据大小为probe\_size，读出数据后调用av\_probe\_input\_format2(&pd, 1, &score)打分，找到iformat后退出。

for循环中avio\_read 读数据，最大读出PROBE\_BUF\_MAX大的数据，读一次数据之后调用av\_probe\_input\_format2猜测demux类型，如果猜测的分数比较低，则增大读取的数据大小，再次调用avio\_read和av\_probe\_input\_format2，记录打分最高的AVInputFormat

av\_probe\_input\_format3、av\_iformat\_next、AVInputFormat read\_probe，这里找到的AVInputFormat是mpegts——ff\_mpegts\_demuxer

这个函数执行完得到AVInputFormat，demuxer。返回值是所得到的AVInputFormat的分数赋值在s->probe\_score中，s->protocol\_whitelist是default\_whitelist的值。

**1.2.4、其它**

s->iformat->read\_header读头，填充AVFormatContext probesize，avformat\_new\_stream给每一个流创建一个AVStream结构体，并赋值相关参数，st->codecpar信息获取填充等。后面以mpeg2ts为例介绍。

\*(const AVClass \*\*) s->priv\_data = s->iformat->priv\_class这里是mpegts中的mpegts\_class

s->internal->data\_offset = avio\_tell(s->pb)，这里是seek到0位置的返回值

**2、avformat\_find\_stream\_info**

avio\_tell(ic->pb) IO seek到起始位置

2.1、第一个for (i = 0; i < ic->nb\_streams; i++)：

av\_parser\_init(st->codecpar->codec\_id)找parser，比如aac\_parser

调用parser\_init，aac\_parse\_init赋值ADTS头大小为AAC\_ADTS\_HEADER\_SIZE 7，s->sync = aac\_sync;

parser\_parse：ff\_aac\_ac3\_parse：s->sync找到一个ADTS帧的开始（通过同步字0xFFF），并解析出采样率、通道数、通道布局、计算码率、采样数、adts帧长度、adts包含音频帧个数等参数，音频帧默认含有1024个采样点。

AAC头结构如下：同步字12位为0xFFF

get\_bits(gbc, 12) sync

skip\_bits1(gbc); /\* id \*/

skip\_bits(gbc, 2); /\* layer \*/

crc\_abs = get\_bits1(gbc); /\* protection\_absent \*/

aot = get\_bits(gbc, 2); /\* profile\_objecttype \*/

sr = get\_bits(gbc, 4); /\* sample\_frequency\_index \*/

skip\_bits1(gbc); /\* private\_bit \*/

ch = get\_bits(gbc, 3); /\* channel\_configuration \*/

skip\_bits1(gbc); /\* original/copy \*/

skip\_bits1(gbc); /\* home \*/

skip\_bits1(gbc); /\* copyright\_identification\_bit \*/

skip\_bits1(gbc); /\* copyright\_identification\_start \*/

size = get\_bits(gbc, 13); /\* aac\_frame\_length \*/

skip\_bits(gbc, 11); /\* adts\_buffer\_fullness \*/

rdb = get\_bits(gbc, 2); /\* number\_of\_raw\_data\_blocks\_in\_frame \*/

AVCodecContext \*avctx用AVStream的AVCodecParameters \*codecpar赋值

根据st->codecpar->codec\_id找到AVCodec

2.1.1、avcodec\_open2打开AVCodec：

AVCodecInternal \*internal;创建填充， avctx->priv\_data codec私有数据创建

ff\_frame\_thread\_encoder\_init ff\_thread\_init线程启动。

2.2、for (;;)：

try\_decode\_frame尝试解码一些帧获取codec信息

2.3、estimate\_timings

算duration：

如果是mpeg，使用estimate\_timings\_from\_pts，读取文件末尾packet的dts作为duration

如果AVStream中有duration，直接获取后填充到ic中

如果没有，则estimate\_timings\_from\_bit\_rate，根据码率计算duration，把所有AVStream的码率相加得到整个文件的码率，根据文件大小，相除得到duration。

2.4、compute\_chapters\_end

把下一个章节的开始时间作为本章节的end时间