**问题背景：**

上一篇文章让大家看了下MP4的主要结构和推荐了一些分析工具，如果你对MP4没有任何了解，还是先看上文，了解MP4的基本结构，其中还有许多工具需要下载和使用。这篇文章借助一个实例分析下核心Box的语法结构和字段含义，其次总结下H264码流的是如何打包到MP4中的，这篇文章也是下篇文章计算一些音视频信息，实现音视频操作和对MP4码流处理的基础。

学习完本篇文章，基本可以清楚了解MP4结构，再对着ISO-14996标准写一个MP4的muxer或者demuxer都不是难事，也可以分析一些线上MP4播放问题。

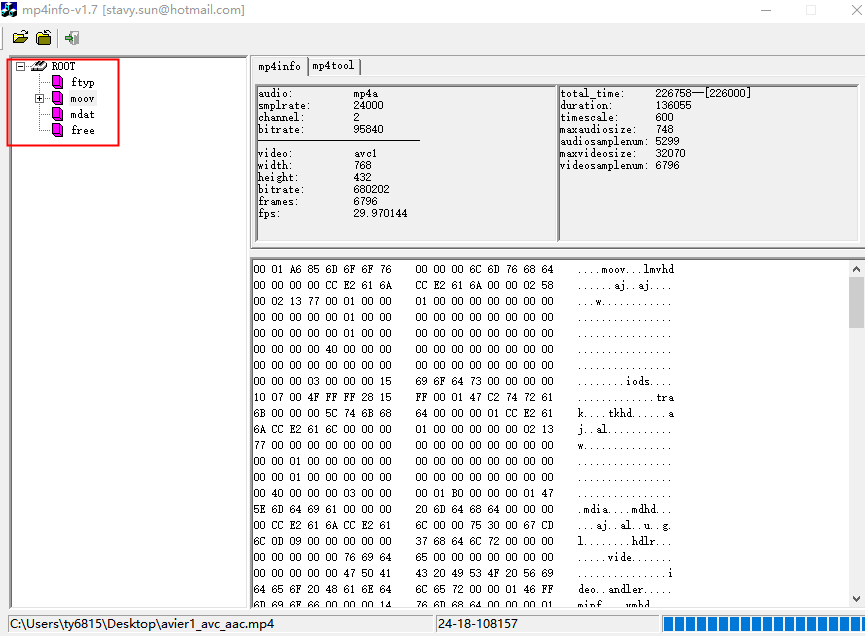
**实例分析：**

下面会实际分析一段码流，然后看下各个Box语法和字段，为了让大家清楚分析的先后顺序关系，约定三点：

1. 分析时会从外到里进行分析，先分析第一层的Box各个字段，再分析第二层Box各个字段，以此类推；
2. Box由于种类非常多，这里只分析一个标准的MP4文件的Box含义，有些特殊和不太用的Box就一笔带过了，如果你万一项目用到直接查ISO标准文档即可，毕竟这里不是翻译ISO文档；
3. Box里面字段也不是所有的都需要关注，我们只需要关注核心和有用的，对于一些不太用的就可以忽略不计了，实际解封装我们也是提取我们想要的信息或者打包时对于不关注的字段直接赋默认值即可，还是希望读者看完这篇文章知道如何分析MP4文件，了解这种思路即可；

* **分析第一层Ftyp、Moov、Mdat、Free等Box:**

用Mp4Info工具打开MP4文件后，不打开内部结构，我们先看到外边的四大Box，现在依次分析下各个Box.



**ftyp Box**

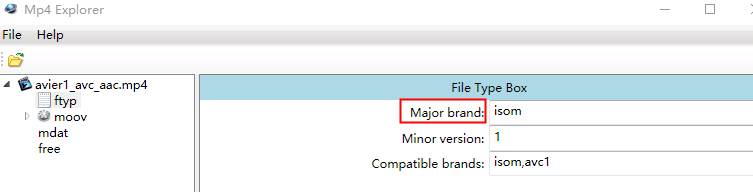
**简介：**

ftyp是MP4文件的第一个Box，包含了视频文件使用的编码格式、标准等，这个Box作用基本就是MP4这种封装格式的标识，同时在一份MP4文件中只有一个这样的Box。ftyp box通常放在文件的开始，通过对该box解析可以让我们的软件（播放器、demux、解析器）知道应该使用哪种协议对这该文件解析，是后续解读文件基础。

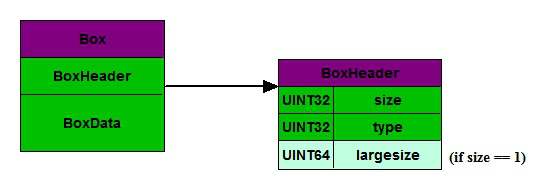
实例分析：

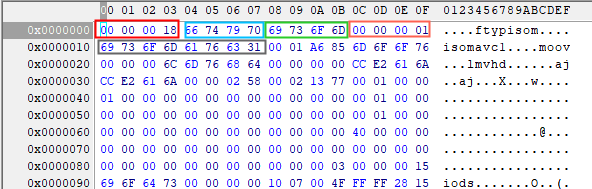
**定义：**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 占用字节 | 含义 | 说明 |
| Box length | 4 | 表示这个Box的长度，包含Box的header长度。 | Box Header，整个包含Box header+Box Data的总长度。 |
| Box type | 4 | 固定值：”ftyp”的ASCII码，为该Box的标识。 | Box Header，固定值 |
| Major brand | 4 | 主商标 | **MP4这种格式基于QuickTime，然后成为ISO国标，后面还有其它厂家基于这个标准的自己扩展的格式如F4V、M4V等，所以后面不同厂商要实现这种规范都要向ISO注册一个四字节的标识码。**  **还有那些商标，可以参考网站：**  <http://www.ftyps.com/> |
| Minor version | 4 | 商标版本号 | 表示Major brand的版本号。 |
| Compatible brand | 12 | 兼容其他的版本号 | 表示可以兼容和遵从那些协议标准，是一个列表。 |



上篇文章已经知道了Box的基本结构，所以我们先找到ftyp的Box，然后进行逐字段分析：

****



**Box Header:**

**Box Length：**

十六进制：0x00 00 00 18

十进制：24

表示该Box长度为24字节

**Box type：**

十六进制：0x66 74 79 70

这就是”ftyp”的ASCII值，标识了该Box的类型

**Box Data:**

major brand**：**

十六进制：0x69 73 6F 6D

即isom，说明本文件是符合这个规范的。

minor version**：**

十六进制：0x00 00 00 01

十进制：1

**isom的版本号，一般默认值是1**

compatible brand**：**

十六进制：0x69 73 6F 6D 61 76 63 31

ASCII值为iosmavc1表示本文件可以兼容iso\avc1等协议和标准

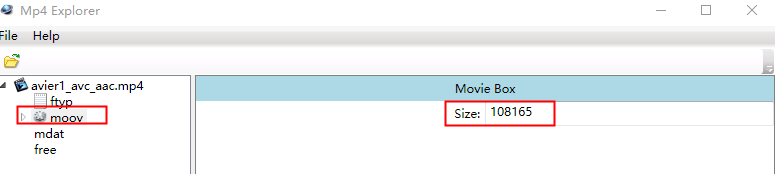
**Moov Box**

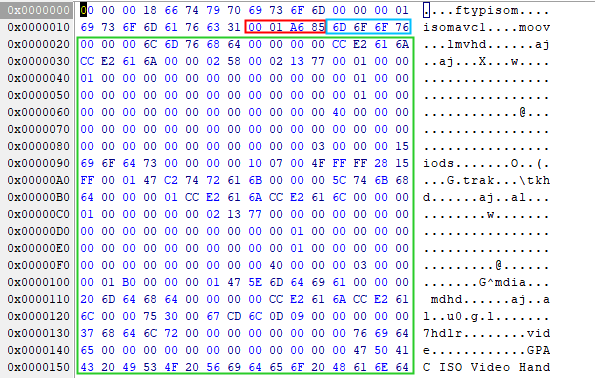
**简介：**

Moov Box这个Box也是MP4文件中必须有但是只存在一个的Box,这个Box里面一般存的是媒体文件的元数据，这个Box本身是很简单的，是一种Container Box，里面的数据是子Box,自己更像是一个分界标识。

所谓的媒体元数据主要包含类似SPS PPS的编解码参数信息，还有音视频的时间戳等信息。对于MP4还有一个重要的采样表stbl信息，这里面定义了采样Sample、Chunk、Track的映射关系，是MP4能够进行随机拖动和播放的关键，也是需要好好理解的部分，对于实现一些音视频特殊操作很有帮助。

**实例分析：**





**Box Header:**

**Box Length：**

十六进制：0x00 01 A6 85

十进制：108165

表示该Box长度为108165字节

**Box type：**

十六进制：0x6D 6F 6F 76

这就是”moov”的ASCII值，标识了该Box的类型

**Box Data:**

十六进制：0x00 00 00 6C......

由于这个Box是Container Box，里面还有好几层子Box，所以暂时不分析，等把第一层分析完再详细讲解这个Box内部。

**Mdat Box**

**简介：**

Mdat Box这个Box是存储音视频数据的Box，要从这个Box解封装出真实的媒体数据。当然这个Box一般都会存在，但是不是必须的。

在前面的文章中已经讲解了H264的基本结构是由一系列的NALU组成。原始的NALU单元组成：

**Start code + NALU header + NALU payload**

但是在MP4格式文件中，H264 slice并不是以00 00 00 01 Start Code来进行分割，而是存储在Mdat Box的Data中。

Mdat Box的格式：

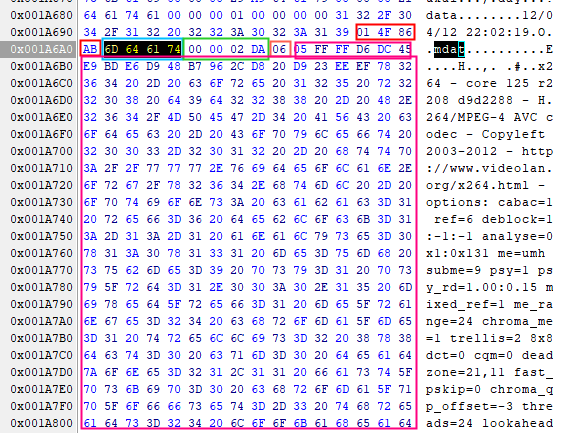
**Box header + Box Data**

**Box length + Box Type + NALU length + NALU header + Nalu Data....... NALU length + NALU header + Nalu Data**

**说明：**

1. Mdat Box基本组成还是有头部和数据两部分组成，但是这里注意如果Box length长度不够时，要用后面8字节的扩展长度字段，Box Type还是“mdat”的ASCII码值；
2. 真实的数据字段是一个个NALU header + NALU Data，但是没有用NALU的分解符00 00 00 01,是在每个NALU前面加了长度字段，和RTP打包类似，可以参考前面文章；
3. 这里的NLAU一般不再包含SPS PPS等数据，这些数据已经放到Moov Box里面了，至于是如何放到Moob Box的，下面文章会讲解。这里一般NALU 类型就是I\B\P帧数据以及SEI用户增强信息；

下面我们就分析下Mdat的Box Data里面数据是不是一个个NALU，进行验证下：



**Box Header:**

**Box Length：**

十六进制：0x01 4F 86 AB

十进制：21989035‬

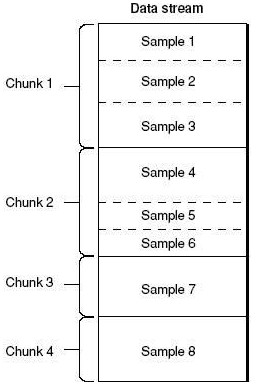
表示该Box长度为108165字节，即后面的NALU整个长度为21989035-8字节。

**Box type：**

十六进制：0x6D 64 61 74

这就是”mdat”的ASCII值，标识了该Box的类型

**Box Data:**



**第一个Nalu Length:**

十六进制：0x00 00 02 DA

十进制：730

说明第一个NALU的长度是730字节，下一个NALU的起始位置应该在：0x1A6A0+0x08+0x2DA即0x1A982处，这个NALU分析完，我们找下下一个NALU继续分析下验证下猜想。

**第一个Nalu Header:**

十六进制：0x06

十进制：0x06&0x1F即6

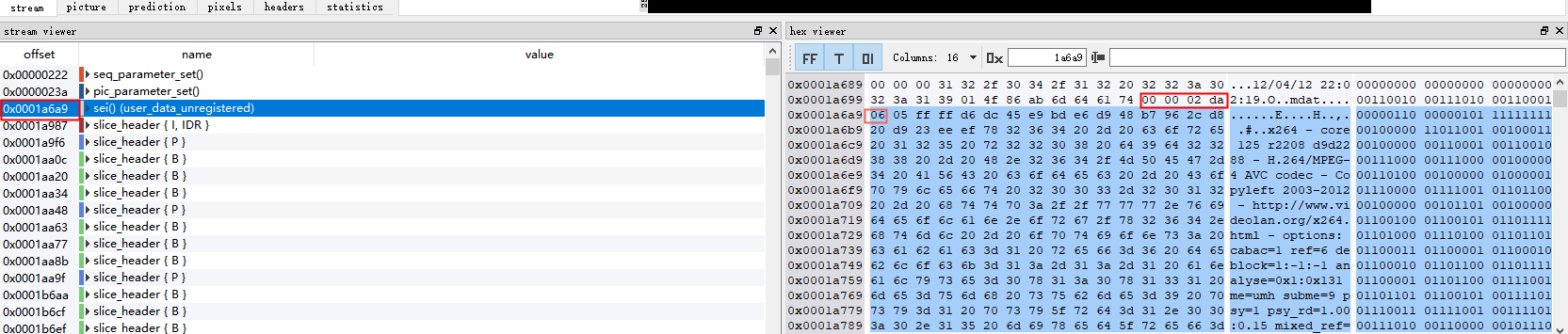
我们判断这个NALU的类型是SEI，这里面存储了大量的用户私有定义的数据。

**第一个Nalu Data:**

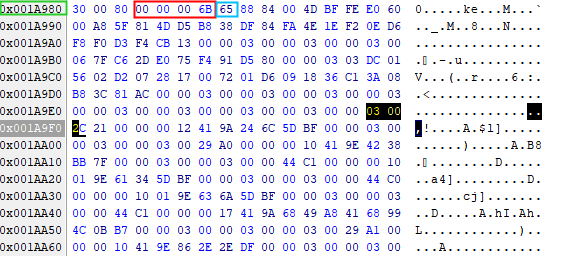
十六进制：0x05 FF FF D6.......

这个后面的730字节都是SEI Nalu的值。

为了验证我分析的对不对，我用Elecard StreamEye交叉验证下第一个NALU的SEI信息如下：



这跟上面分析结果是一致的，继续按照上面的猜想分析Mdat Box的第二个NALU信息如下：



上面已经分析第二个NALU的位置是0x1A982，现在分析下第二个NALU的具体信息。

**第二个Nalu Length:**

十六进制：0x00 00 00 6B

十进制：107

说明第一个NALU的长度是107字节

**第二个Nalu Header:**

十六进制：0x65

十进制：0x65&0x1F即5，

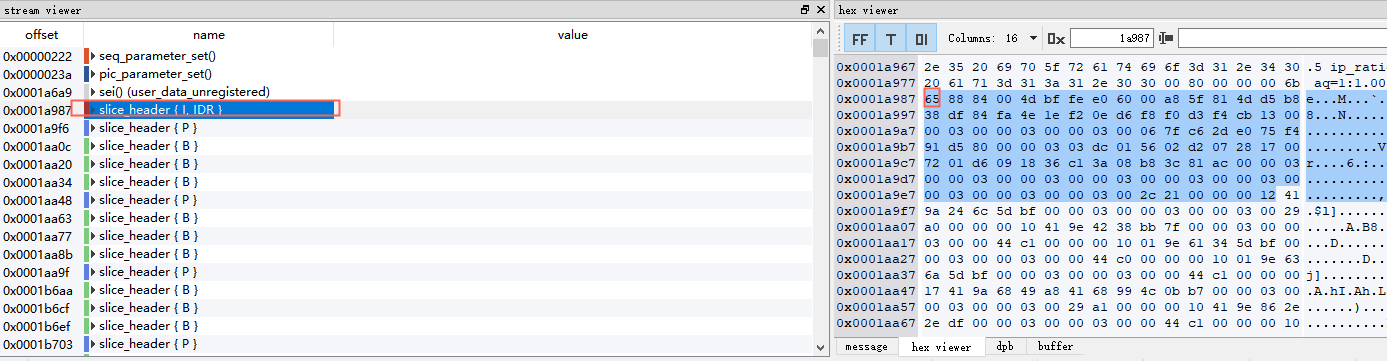
我们判断这个NALU的类型为I帧

**第二个Nalu Data:**

十六进制：0x88 84 00 4D.......

这个后面的107字节都是I帧 Nalu的值，依次类推我们就能扎到Mdat Box里面所有的NALU了。

为了验证猜想，同样我用Elecard StreamEye进行了交叉验证，后面依次发现了P、B帧的NALU分片都是在Mdat Box按照上面说的方式一次排列；

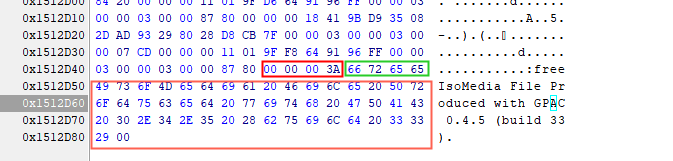


**Free Box**

**简介：**

Free Box中的内容是无关紧要的，可以被忽略即该box被删除后，不会对播放产生任何影响。这种类型的Box也不是必须的，可有可无，类似的Box还有Sikp Box.虽然在解析是可以忽略，但是需要注意该Box的删除对其它Box的偏移量影响，特别是当Moov Box放到Mdat Box后面的情况。

**实例分析：**



**Box Header:**

**Box Length：**

十六进制：0x00 00 00 3A

十进制：58

表示该Box长度为58字节

**Box type：**

十六进制：0x66 72 65 65

这就是”free”的ASCII值，标识了该Box的类型

**Box Data:**

十六进制：0x49 73 6F 4D......

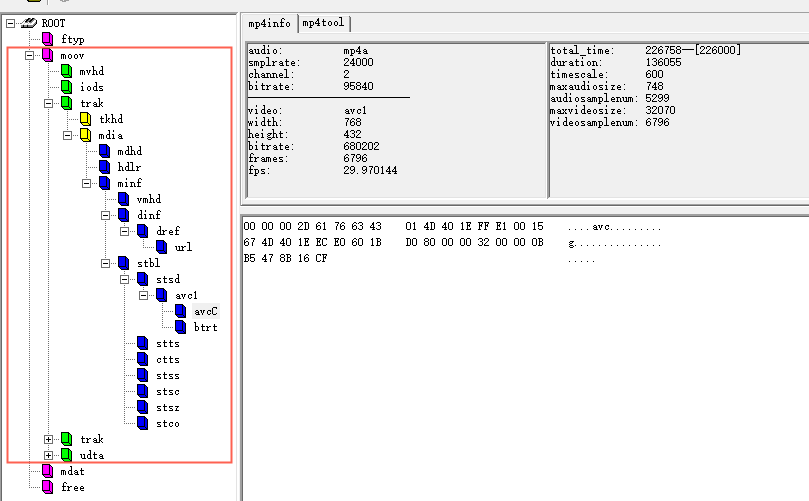
从内容来看，这里在Free Box的Data里面填充了一些MP4打包软件的相关信息，一般没啥用，忽略即可。

讲到这里基本把MP4文件最外层的几个核心Box已经讲解完了，这些东西也没啥难度，就是对照标准文档理解了就行。下面核心讲解下Moov Box里面的各个层次的子Box，这里面内容丰富。既有时间戳信息，也有H264里面对SPS PPS的封装，是进行音视频播放和正常解码的关键，同时里面还有Stbl Box的存在，这里面有一定的设计算法，比较精巧，非常方便实现对MP4文件的拖动和暂停等操作，需要好好理解下。

* **分析第二层Moov Box的Mvhd、Trck、Iods等 Box:**

Moov Box详解：

先用Mp4Info工具看下Moov Box的详细内部结构，将按照从上到下从外及里依次进行分析各个Box的字段和含义：

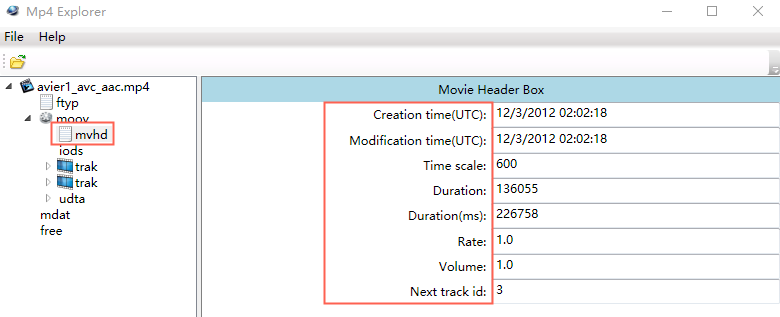


**Mvhd Box**

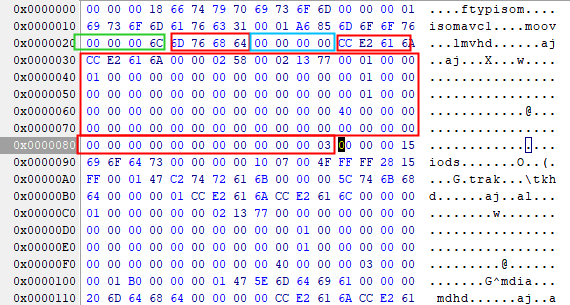
**简介：**

这个Box也是全文件唯一的一个Box,一般处于Moov Box的第一个子Box,这个Box对整个媒体文件所包含的媒体数据（包含Video track和Audio Track等）进行全面的描述。其中包含了媒体的创建和修改时间，默认音量、色域、时长等信息。

**实例分析：**



这个是用Mp4Explorer分析的结果，下面实际对照数据分析下各个字段含义：



**Box Header:**

**这里需要注意下该Box的头，因为这个Box是Full Box也就是在原来的头后面有扩展字段即在后面加上了8Bit的version和24Bit的 flags字段。**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| box size | 4 | box大小 |
| box type | 4 | box类型 |
| version | 1 | box版本，0或1，一般为0 |
| flags | 3 | 根据各个Box类型定义是0还是1 |

**Box Length：**

十六进制：0x00 00 00 6C

十进制：108

表示该Box长度为108字节,包含了Bod Header和Box Data

**Box type：**

十六进制：0x6D 76 68 64

这就是”mvhd”的ASCII值，标识了该Box的类型

**Box Version:**

十六进制：0x00

实际代码就是默认填充0，不用太纠结这个字段值；

**Box Tag：**

十六进制：0x00 00 00

实际代码就是默认填充0，不用太纠结这个字段值；

**Box Data:**

先说明下mvhd的数据部分字段含义：

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| creation time | 4 | 创建时间（相对于UTC时间1904-01-01零点的秒数） |
| modification time | 4 | 修改时间（相对于UTC时间1904-01-01零点的秒数） |
| time scale | 4 | 文件媒体在1秒时间内的刻度值，可以理解为1秒长度的时间单元数，这个只用来计算了该Mp4文件的长度，但是没有参与PTS和DTS的计算。 |
| duration | 4 | 该影片的播放时长，该值除以time scale字段即可以得到该影片的总时长单位秒s |
| rate | 4 | 推荐播放速率，高16位和低16位分别为小数点整数部分和小数部分，即[16.16] 格式，该值为1.0（0x00010000）表示正常前向播放 |
| volume | 2 | 与Rate类似，[8.8] 格式，1.0（0x0100）表示最大音量 |
| reserved | 10 | 保留位 |
| matrix | 36 | 视频变换矩阵，实际填写都是默认值，一般不关心这个字段 |
| pre-defined | 24 | 预览相关的信息，一般都是填充0，所以不用太关心 |
| next track id | 4 | 下一个Track使用的id号，通过该值减去1可以判断当前文件的Track数量 |

creation time**：**

十六进制：0xCC E2 61 6A

十进制：3437388138

这个时间就是从1904/1/1经过的秒数，可以换算下刚好是2012/12/03 22:02:18

modification time**：**

十六进制：0xCC E2 61 6A

十进制：3437388138

这个时间就是从1904/1/1经过的秒数，可以换算下刚好是2012/12/03 22:02:18

time scale**：**

十六进制：0x00 00 02 58

十进制：600

1秒的时间单位是600，即把1秒划分为600分，这样描述的更精确；

duration**：**

十六进制：0x00 02 13 77

十进制: 136055

这样我们换算下该段MP4文件的时长是136055/600即226.75秒

rate**：**

十六进制：0x00 01 00 00

十进制：1.0

**表示采样原始倍速播放，一般就是1倍速进行播放该视频；**

volume**：**

十六进制：0x01 00

十进制：1.0

**表示采样原始视频蛮音量进行播放；**

reserved**：**

十六进制：0x00 00 00 00 00 00 00 00 00 00

一般默认0即可

matrix**：**

十六进制：0x00 01 00 00......00 00 00 40

**这36字节也是一般用默认值，视频的空间变换矩阵，实际使用也没啥用，按照本例默认填写即可；**

pre-defined**：**

十六进制：0x00 00 00 00......

这24字节一般也是默认00;

next track id**：**

十六进制：0x00 00 00 03

这个值表示如果要增加下一个Track时，需要的编号是3，那同时也就说明本文件里面有2个Track，实际发现刚好是2个Track;

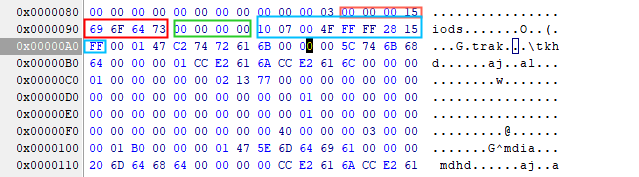
**Iods Box**

**简介：**

这个Box也是非必须Box，不算核心Box,实际也是24字节的固定值**，解析时直接跳过即可。封装直接给填写固定24字节即可**，注意该Box也是Full Box意味这Header里面有1字节的Version和3字节的Flag字段。

定义的内容应该是Audio和Video ProfileLevel方面的描述。但是现在没有用。

**实例分析：**



实现参考：



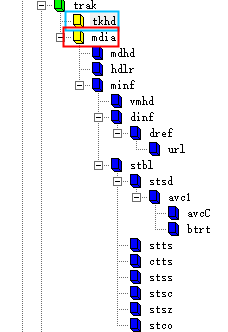
从这里开始我们将从上到下从外到里解析Track Box，这个Box最复杂也最重要，需要理解好，特别是内部子Stbl Box。其中Trak Box有两个一个是视频Video Track，一个是音频Audio Track，如果视频的Track和音频的Track 的Trck Box下面的子Box是一致的，则不讲，如果有差异会在讲完该视频的Box，然后紧接着讲解音频 Track的该Box对应含义，阅读时注意。

**Trak Box**

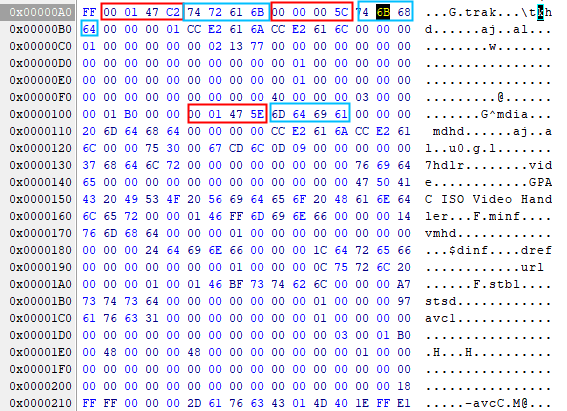
**简介：**

Trak Box定义了媒体文件中媒体中一个Track的信息，视频有Video Track,音频有Audio Track，媒体文件中可以有多个Track，每个Track具有自己独立的时间和空间的信息，可以进行独立操作。

每个Track Box都需要有一个Tkhd Box和Mdia Box，其它的Box都是可选择的：



实例分析：



Trak Box这里自身只是一个分界符，属于Container Box，所以它只有Box Header。

**Box Length：**

十六进制：0x00 01 47 C2

十进制：83906

表示该Box长度为83906字节,包含了Bod Header和Box Data

**Box type：**

十六进制：0x74 72 61 6

这就是”trak”的ASCII值，标识了该Box的类型

同样我们发现下面一级子Box就是Tkhd和Mdia,然后分别计算Box的大小是92和83806，二者之和刚好就是父Box Trak的大小92+83806即83898再加上Trak自身的8字节即83896字节。

下面分析thhd和mdia两个Box以及子Box详细内容。

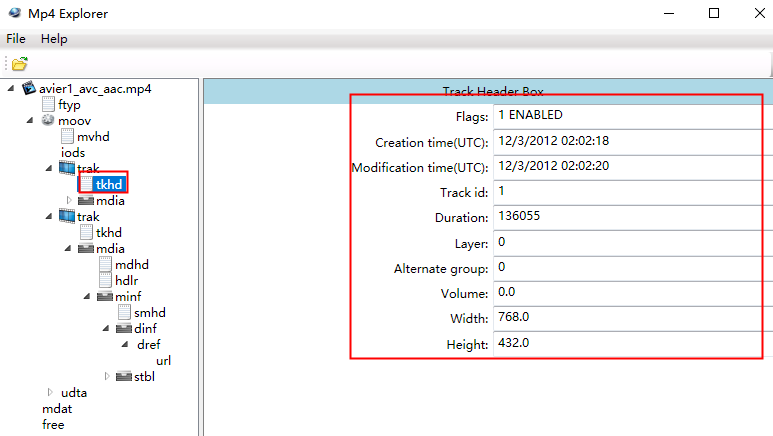
* **分析第三层Tkhd、 Mdia等Box:**

**Tkhd Box**

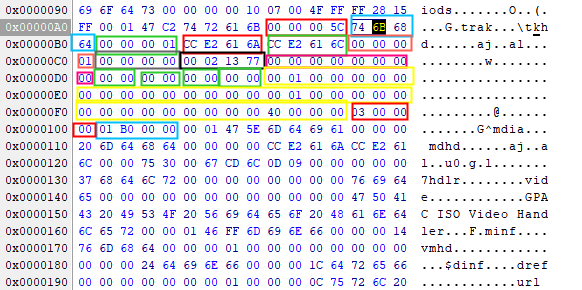
**简介：**

该Box描述了该Track的媒体整体信息包括时长、图像的宽度和高度等，实际比较重要，同时该Box是Full Box即Box Header后面有Box Header Version一字节和Box Header Flag三字节字段。

**实例分析：**



用Mp4Explprer已经分析出来各个字段，下面可以进行实际验证下：



**Box Header:**

**这里需要注意下该Box的头，以为这个Box是Full Box也就是在原来的头后面有扩展字段即在后面加上了8Bit的Version和24Bit的 flags字段。**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| box size | 4 | box大小 |
| box type | 4 | box类型 |
| version | 1 | box版本，0或1，一般为0。（以下字节数均按version=0） |
| flags | 3 |  |

**Box Length：**

十六进制：0x00 00 00 5C

十进制：92

表示该Box长度为92字节,包含了Bod Header和Box Data

**Box type：**

十六进制：0x74 6B 68 64

这就是”tkhd”的ASCII值，标识了该Box的类型

**Box Version:**

十六进制：0x00

实际代码就是默认填充0，不用太纠结这个字段值；

**Box Tag：**

十六进制：0x00 00 01

该Box实际代码就是默认填充01，不用太纠结这个字段值；

**Box Data:**

先说明下mvhd的数据部分字段含义：

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| creation time | 4 | 创建时间（相对于UTC时间1904-01-01零点的秒数） |
| modification time | 4 | 修改时间（相对于UTC时间1904-01-01零点的秒数） |
| track ID | 4 | 唯一标识该Track的一个非零值 |
| reserve | 4 | 保留位8字节 |
| duration | 4 | 该Track的播放时长，需要和时间戳进行一起计算，然后得到播放的时间坐标。 |
| reserver | 8 | 保留位8字节 |
| layer | 2 | 视频层，值小的在上层 |
| alternate group | 2 | track分组信息，一般默认为0，表示该track和其它track没有建立群组关系 |
| volume | 2 | 播放此track的音量，1.0为正常音量 |
| reserver | 2 | 默认值0 |
| matrix | 36 | 视频变换矩阵 |
| width | 4 | 如果该Track为Video Track，则表示图像的宽度（16.16浮点表示） |
| height | 4 | 如果该Track为Video Track，则表示图像的高度（16.16浮点表示） |

creation time**：**

十六进制：0xCC E2 61 6A

十进制：3437388138

这个时间就是从1904/1/1经过的秒数，可以换算下刚好是2012/12/03 22:02:18

表示该Track生效并且用在这个影片中的时间就是2012/12/03 22:02:18。

modification time**：**

十六进制：0xCC E2 61 6C

十进制：3437388140

这个时间就是从1904/1/1经过的秒数，可以换算下刚好是2012/12/03 22:02:20

track ID**：**

十六进制：0x00 00 00 01

十进制：1

这是该Track的唯一标识。

reserve：

十六进制：0x00 00 00 00

默认为0；

duration**：**

十六进制：0x00 02 13 77

十进制: 136055

这样我们换算下该段MP4文件的时长是136055/600即226.75秒

reserved**：**

十六进制：0x00 00 00 00 00 00 00 00 00 00

8字节，一般默认0即可

**layer：**

十六进制：0x00 00

十进制：0

**视频层，默认值为0，值小的在最上层**

alternate group**：**

十六进制：0x00 00

一般默认0即可

volume**：**

十六进制：0x00 00

十进制：0

由于是Video Track，所以这个值肯定是0；

reserved**：**

十六进制：0x00 00

一般默认0即可

matrix**：**

十六进制：0x00 01 00 00......40 00 00 00

**这36字节也是一般用默认值，视频的空间变换矩阵，实际使用也没啥用，按照本例默认填写即可；**

width**：**

十六进制：0x03 00 00 00

十进制：768.00

**表示该视频图像的宽度是768；**

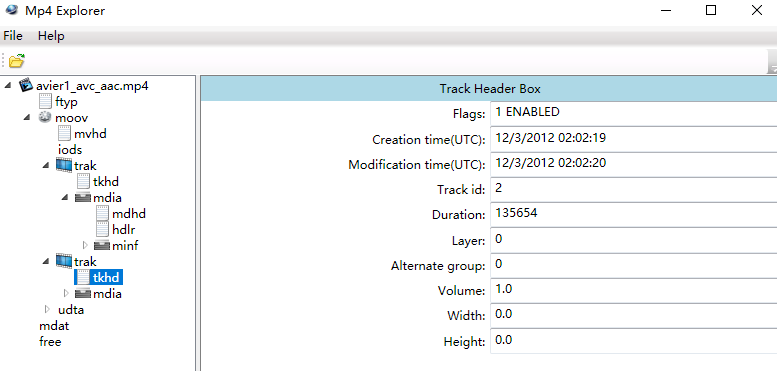
**height：**

十六进制：0x01 B0 00 00

十进制：432.00

表示该视频图像的高度是432；

视频Track的分析完我们看下音频Track的**Tkhd Box：**



这个基本定义和上面是一样的，就是音频的Volume字段为1.0表示按照原始音频的音量进行播放，其它已经分析不再赘述。

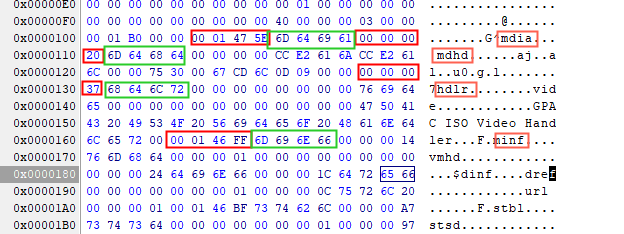
**Mdia Box**

**简介：**

这个Box也是Container Box，里面包含子Box，一般必须有Mdhd Box、Hdlr Box、Minf Box。基本就是当前Track媒体头信息和媒体句柄以及媒体信息。

Box自身非常简单，就是一个标识而已，最复杂的还是里面包含的子Box.

**实例分析：**



Mdia Box这里自身只是一个分界符，属于Container Box，所以它只有Box Header。

**Box Length：**

十六进制：0x00 01 47 5E

十进制：83806

表示该Box长度为83806字节,包含了Bod Header和Box Data

**Box type：**

十六进制：0x6D 64 69 61

这就是”Mdia”的ASCII值，标识了该Box的类型

同样我们发现下面一级子Box就是Mdhd、Hdlr、Minf,然后分别计算Box的大小是0x20即32、0x37即55、0x146FF即83711，三者之和的大小32+55+83711即83798再加上Mdia本身的8字节即83806字节。

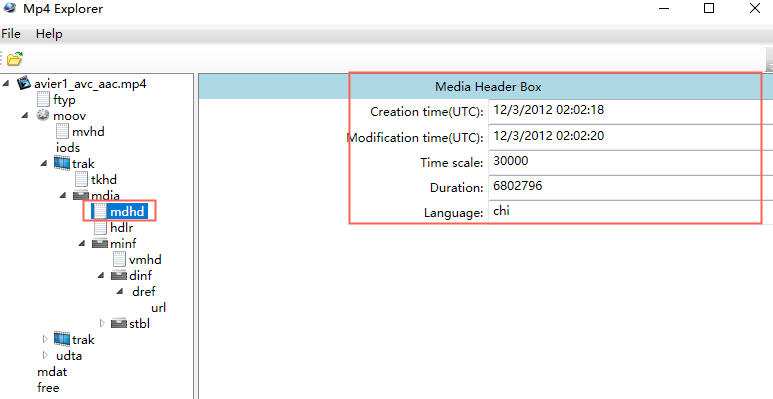
* **分析第四层Mdhd、Hdlr、Minf等Box:**

**Mdhd Box**

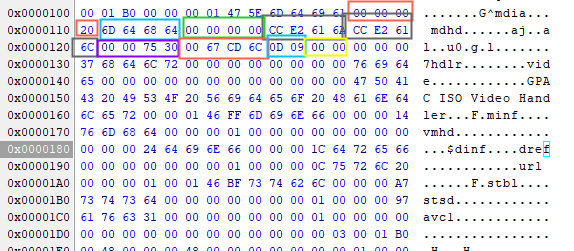
**简介：**

这个Box是Full Box，意味这Box Header有Version和Flag字段，该Box里面主要定义了该Track的媒体头信息，其中我们最关心的两个字段是Time scale和Duration，分别表示了该Track的时间戳和时长信息，这个时间戳信息也是PTS和DTS的单位。

**实例分析：**



这是用Mp4Explorer工具分析的结果，再从二进制原始数据交叉验证下：



**Box Header:**

**这里需要注意下该Box的头，以为这个Box是Full Box也就是在原来的头后面有扩展字段即在后面加上了8Bit的Version和24Bit的 flags字段。**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| box size | 4 | box大小 |
| box type | 4 | box类型 |
| version | 1 | box版本，0或1，一般为0。（以下字节数均按version=0） |
| flags | 3 | 该Box该字段填充0 |

**Box Length：**

十六进制：0x00 00 00 20

十进制：32

表示该Box长度为32字节,包含了Bod Header和Box Data

**Box type：**

十六进制：0x6D 64 68 64

这就是”mdhd”的ASCII值，标识了该Box的类型

**Box Version:**

十六进制：0x00

实际代码就是默认填充0，不用太纠结这个字段值；

**Box Tag：**

十六进制：0x00 00 00

该Box的默认填充0，不用太纠结这个字段值；

**Box Data:**

先说明下mdhd的数据部分字段含义：

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| creation time | 4 | 当前Track创建时间（相对于UTC时间1904-01-01零点的秒数） |
| modification time | 4 | 当前Track修改时间（相对于UTC时间1904-01-01零点的秒数） |
| **timescale** | 4 | 当前Track的时间计算单位 |
| **duration** | 4 | 该Track的播放时长，需要参考前面的timescale计算 |
| language | 2 | 媒体语言码，参考其他标准 |
| qualiiy | 2 | 媒体的回放质量 |

creation time**：**

十六进制：0xCC E2 61 6A

十进制：3437388138

这个时间就是从1904/1/1经过的秒数，可以换算下刚好是2012/12/03 22:02:18

表示该Track生效并且用在这个影片中

modification time**：**

十六进制：0xCC E2 61 6C

十进制：3437388140

这个时间就是从1904/1/1经过的秒数，可以换算下刚好是2012/12/03 22:02:20

表示当前Track的修订时间

timescale**：**

十六进制：0x00 00 75 30

十进制：30000

表示当前Track的时间戳单位是30000；

**duration：**

十六进制：0x00 67 CD 6C

十进制：6802796

表示当前Track的时长是：6802796/30000即226.75秒，这个和前面计算的时长是一致的；

**language：**

十六进制：0x0D 09

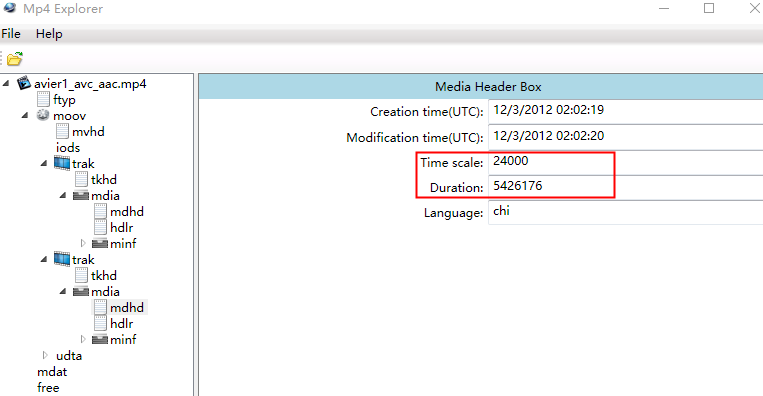
最高位为0，后面15代表三个字符，可以参考标准ISO639-2/T查看代表的预研，这个码查下来代表的china。

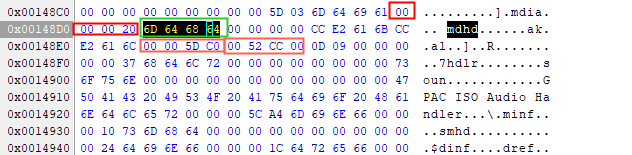
**quality：**

十六进制：0x00 00

媒体的回放质量，这里也没有明确规定，一般默认值00即可；

音频Track对照分析，只分析Box Data不一样的部分：





timescale**：**

十六进制：0x00 00 5D C0

十进制：24000

表示当前Audio Track的时间戳单位是24000；

**duration：**

十六进制：0x00 52 CC 00

十进制：5426176

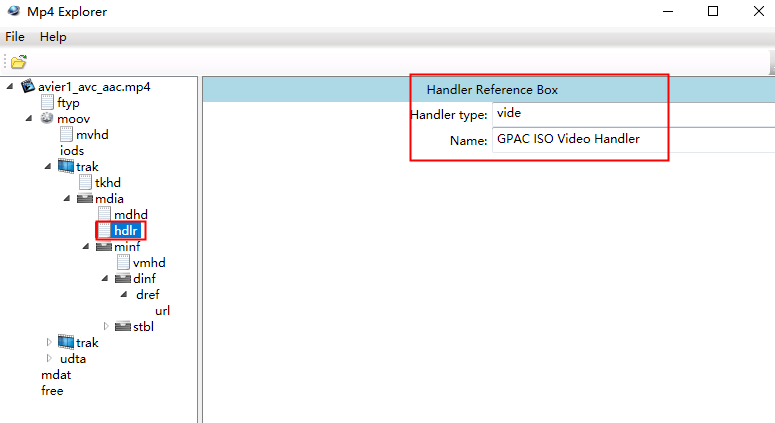
表示当前Track的时长是：5426176/24000即226.095秒，这个和前面计算的时长是一致的；

**Hdlr Box**

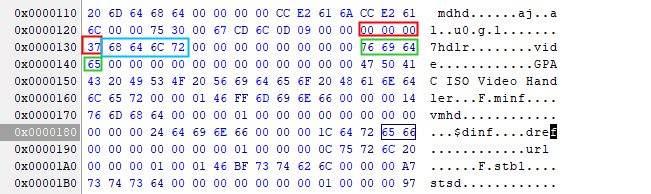
**简介：**

这个Box是Full Box，意味这Box Header有Version和Flag字段，该Box解释了媒体的播放过程信息，用来设置不同Track的处理方式，标识了该Track的类型。

**实例分析：**



我们能看到当前Track的类型是vide Track，标识了该Track描述了视频Track的媒体元数据。



该Box是Full Box，所以Box Header的头部数据已经在前面其它Box多次介绍，这里后面遇到类似就忽略不计了，我们主要看各个Box Data部分。

**Box Data:**

先说明下mdhd的数据部分字段含义：

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| handle type | 4 | Handle 的类型：  ‘mhlr’:media handles  ‘dhlr’:data handle  实际处理时默认为0即可 |
| hand sub type | 4 | 视频：‘vide’  音频：‘soun’ |
| reserved | 12 | 保留字段，默认为0 |
| component name | 可变 | 这个component的名字，其实这里就是你给该Track其的名字，打包时填写一个有意义字符串就可以。 |

handle type **：**

十六进制：0x00 00 00 00

默认值一般填0即可

hand sub type**：**

十六进制：0x76 69 64 65

表示当前的Track类型是vide即表示的视频track;

reserved**：**

十六进制：0x00 00 00 00 00 00 00 00 00 00 00 00

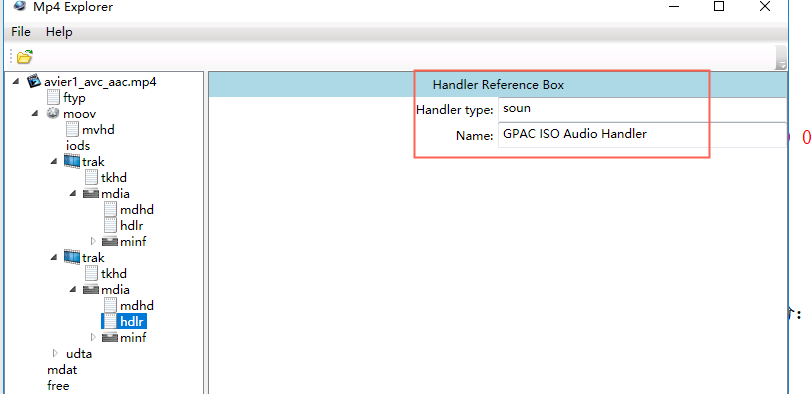
12字节的0x00，一般默认值即可

component name**：**

十六进制：0x47 50 41 43 .....65 72

给这个视频Track起的名字是GPAC ISO Video Handler

音频Track对照分析，只分析Box Data不一样的部分：



**Minf Box**

**简介：**

这个Box是我认为Moov Box里面最重要最复杂的Box，内部还有子Box，我们还是从上到下从外到内的分析各个Box。该Box建立了时间到真实音视频Sample的映射关系，对于音视频数据操作时很有帮助的。

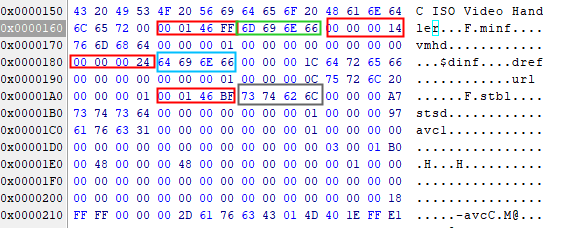
同时该Box是Container Box，下面一般含有三大必须的子Box:

媒体信息头Box: Vmhd Box或者Smhd Box;

数据信息Box：Dinf Box

采样表Box：Stbl Box

**实例分析：**



minf作为Container Box，也就是临界符，所以仅仅包含8字节的头部数据就不进行分析了，参考前面的Box分析结果即可，其中该Box和子Box长度为：0x00 01 46 FF即83711，然后分析第一级子Box的长度：

Vmhd Box Size:0x00 00 00 14即20；

Dinf Box Size:0x00 00 00 24即36；

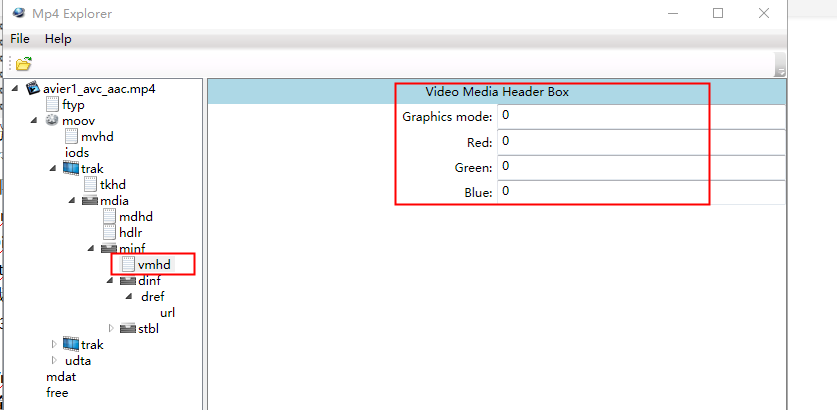
Stbl Box Size:0x00 01 46 BF即83647；

我们将三者之和加起来20 + 36 +83467即83703再加上Minf Box的本身8字节，刚好为83711。

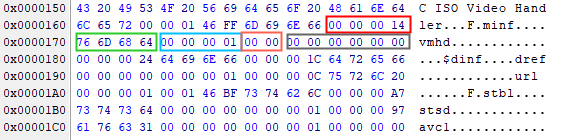
* **分析第五层Vmhd、Smhd、Dinf、Stbl等Box:**

**Vmhd Box**

**简介：**

这个Box的大小是固定的，如果是Vmhd则大小就是20（应该是20，不是24）字节，如果是Smhd则大小是16字节，一般都是用默认值填充。

同时该Box为Full Box意味着Box Header多了四字节的version和Flag字段:



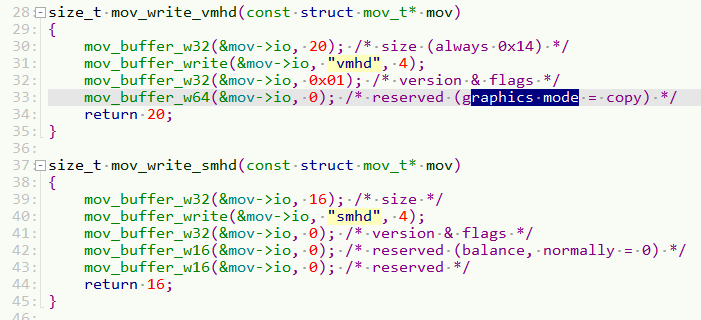
Box Header:

前面已经分析很多Full Box的Box Header，此处不再分析，参考前面即可：

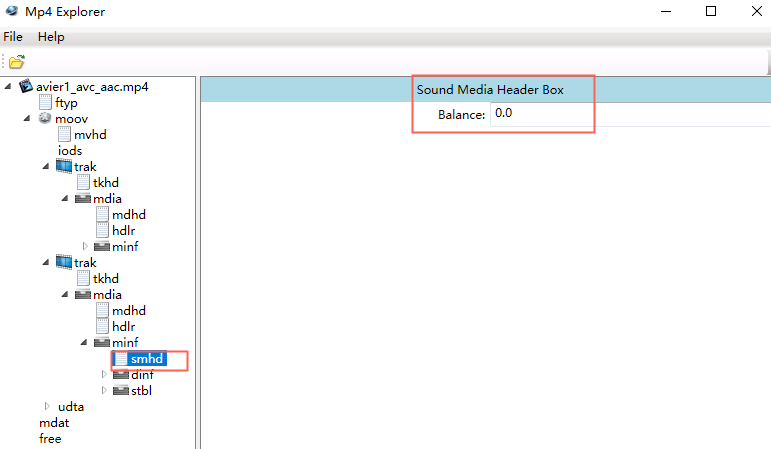
Box Data:

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| raphics mode | 2 | 视频合成模式，为0时拷贝原始图像，否则与opcolor进行合成  一般默认0x00 00即可 |
| Opcolor | 6（2\*3） | 颜色值，RGB颜色值，｛red，green，blue｝  一般默认0x00 00 00 00 00 00 |

上面已经框出来了，就不用再分析这两个字段，下面参考下代码实现即可：



音频Track对照分析，只分析Box Data不一样的部分：



Balance:

十进制：0.0

音频的均衡是用来控制计算机的两个扬声器的声音混合效果，一般是0。一般值是0。

**Dinf Box**

**简介：**

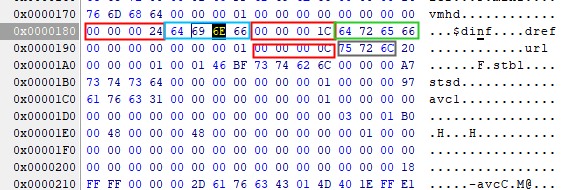
这个Box也是一个container box，一般用来定位媒体信息。一般会包含一个Dref Box即data reference box，Dref下面会有若干个Url Box或者也叫 Urn Box，这些Box组成一个表，用来定位Track的数据。

Track可以被分成若干个段，每一段都可以根据Url或者Urn指向的地址来获取数据，sample描述中会用这些片段的序号将这些片段组成一个完整的track,一般情况下当数据完全包含在文件中，Url和Urn Box的字符串是空的。

这个Box存在的意义就是允许MP4文件的媒体数据分开最后还能进行恢复合并操作，但是实际上，Track的数据都保存再文件中，所以该字段的重要性还体现不出来。

**实例分析：**

Dinf Box也作为Container Box，也就是临界符，所以仅仅包含12字节的头部数据就不进行分析了，参考前面的Box分析结果即可，真实数据还是要看子Dref Box。



Dinf Box的长度是0x24即36字节，其子Dref Box的长度是0x1C即28字节，所以：

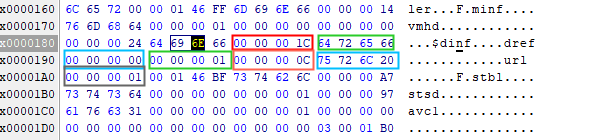
28+8刚好就是Dinf的长度，下面分析Dref Box.

**Dref Box**

**简介：**

由于Track可以分为多个段，所以该Box是用来定义当前Track各个段的URl。

**实例分析：**



**Box Header:**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| box size | 4 | box大小 |
| box type | 4 | box类型 |
| version | 1 | box版本，0或1，一般为0。（以下字节数均按version=0） |
| flags | 3 | 标记 |

**Box Length：**

十六进制：0x00 00 00 1C

十进制：28

表示该Box长度为28字节,包含了Bod Header和Box Data

**Box type：**

十六进制：0x64 72 65 66

这就是”dref”的ASCII值，标识了该Box的类型

**Box Version:**

十六进制：0x00

实际代码就是默认填充0;

**Box Tag：**

十六进制：0x00 00 00

该Box的默认填充0，不用太纠结这个字段值；

**Box Data:**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| entry count | 4 | Data reference 数目 |
| url or urn list | 4 | url 或者urn Box |

entry count:

十六进制：0x00 00 00 01

十进制：1

说明下面的url 列表数量是1，只有一个Url Box.

**Url Box：**

十六进制：0x00 00 00 0C 75 72 6C 20 00 00 01

这里直接把Url Box作为Dref 的Box Data 进行分析。

Url Box Header Length:

十六进制：0x00 00 00 0C

十进制：12

表明当前Url Box的大小为12字节

Url Box Header Type:

十六进制：0x75 72 6C 20

当前Box 的url 的ASCII码，表明当前Box的类型

Url Box Header Version:

十六进制：0x00

默认值，一般填充0即可；

Url Box Header Tag:

十六进制：0x00 00 01

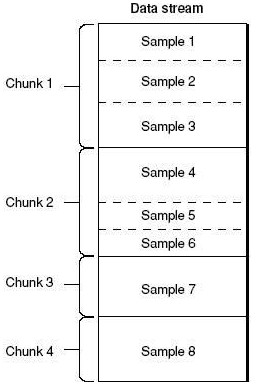
值为1则表明“url”中的字符串为空，表示track数据已包含在文件中，所以Url的Url Box Data部分为空。

**下面我们分析最关键的Stbl Box，看下到底是怎样把媒体数据的Sample和时间进行映射的，再看下Sample-Trunk-Track的三者关系。**

**Stbl Box**

**简介：**

上篇文章我们知道Sample是媒体数据的存储单元，其中存储在Media的chunk中，在分析Mdat Box的H264 NALU打包时已经体现出来了，这种关系如下表所示：



其中每个Sample的时间和位置、编解码信息、和chunk关系等都是由Stbl Box来描述的，该Box又称为采样参数列表即Sample Table。

Stbl Box包含了Track中media媒体的所有时间和索引，利用这个容器的Sample信息，就可以定位Sample的媒体时间、类型、大小以及和其相邻的Sample。同时该Box是必须在Trak Box中存在的。

其一般要包含下列子Box：

采样描述容器： Sample Description即Stsd Box

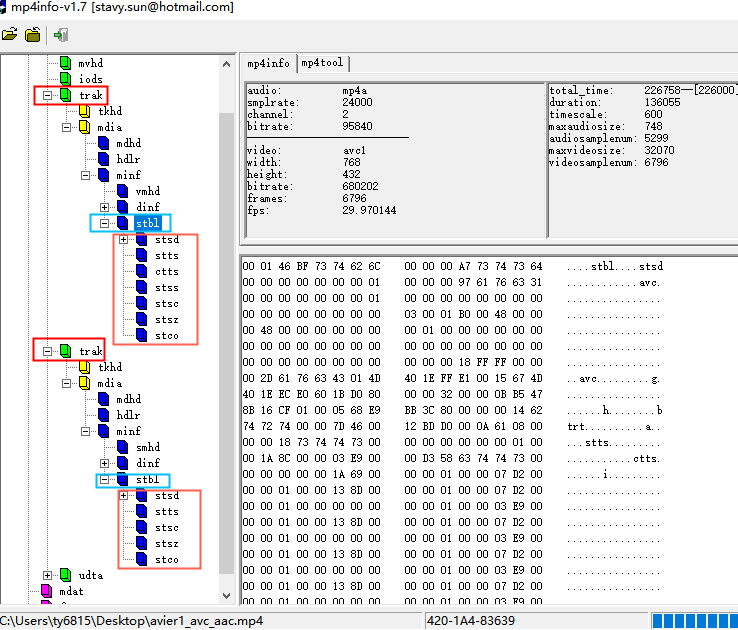
采样时间容器： Time To Sample 即Stts Box

采样同步容器： Sync Sample即Stss

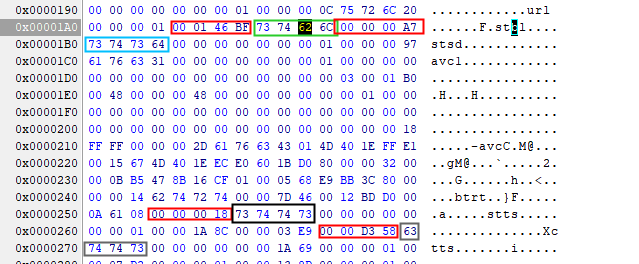
Chunk采样容器: Sample To Chunk即Stsc

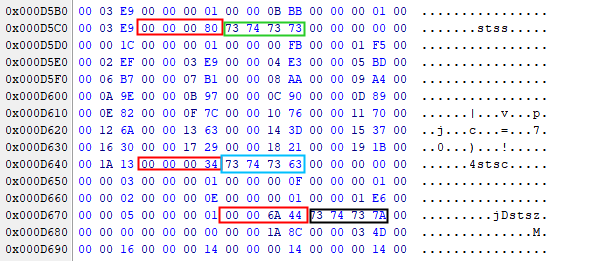
采样大小容器： Sample Size即Stsz

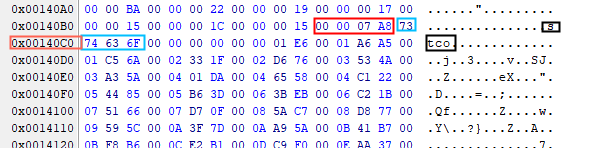
Chunk偏移容器：Chunk Offest即Stco



**实例分析：**







Stbl Box同样是Container Box，所以这里也仅仅是8字节的分解符，这里Box Header部分不在分析，其大小为0x00 01 46 BF即83647现在我们分析其下一级子Box的长度：

Stsd Box：0x00 00 00 A7即167

Stts Box: 0x00 00 00 18即24

Ctts Box: 0x00 00 D3 58即3381

Stss Box: 0x00 00 00 80即128

Stsc Box: 0x00 00 00 34即52

Stsz Box: 0x00 00 6A 44即27204

Stco Box: 0x00 00 07 A8即1960

* **分析第六层Stsd、Stts、Ctts、Stss、Stsc、Stsz、Stco等Box:**

**Stsd Box**

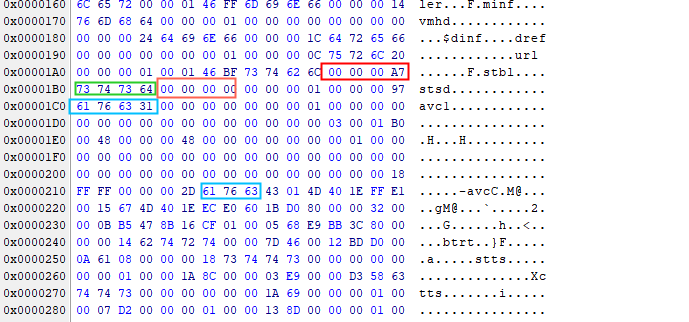
简介：

该Box存储了编码类型和初始化解码器需要的信息，于特定的track-type有关，根于不同的Track使用不一样的编码标准。

box header和version字段后会有一个entry count字段，根据entry的个数，每个entry会有type信息，如“vide”、“sund”等，根据type不同sample description会提供不同的信息，例如对于video track，会有“VisualSampleEntry”类型信息，对于audio track会有“AudioSampleEntry”类型信息。视频的编码类型、宽高、长度，音频的声道、采样等信息都会出现在这个box中。

这个Box也是Full Box，对于Video Track里面而言，H264的SPS PPS就存在该Box里面，对于解码非常重要。

**实例分析：**



**Box Header:**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| box size | 4 | box大小 |
| box type | 4 | box类型 |
| version | 1 | box版本，0或1，一般为0。（以下字节数均按version=0） |
| flags | 3 | 标记 |

**Box Length：**

十六进制：0x00 00 00 A7

十进制：167

表示该Box长度为28字节,包含了Bod Header和Box Data

**Box type：**

十六进制：0x73 74 73 64

这就是”stsd”的ASCII值，标识了该Box的类型

**Box Version:**

十六进制：0x00

实际代码就是默认填充0;

**Box Tag：**

十六进制：0x00 00 00

该Box的默认填充0，不用太纠结这个字段值；

**Box Data**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| sample description entry count | 4 | sample description 数目,同时不同的Track有不同的sample description,当然sample description也是以Box描述的。 |
| sample description entry |  | Sample description的实际内容 |

sample description entry count：

十六进制：00 00 00 01

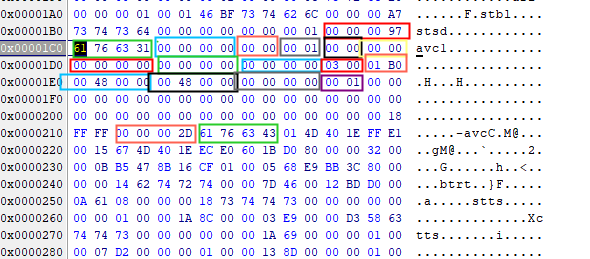
说明后面跟着的只有一个sample description entry，本质还是Box，对于编码类型是H264数据的这里的子Box则为Avc1 Box，我们继续进行分析：

**Avc1 Box:**

简介：

当前为Video Track，所以VisualSampleEntry为Avc1 Box，内部含有SPS PPS 等音视频解码信息。

**实例分析：**



**Box Header:**

**Box Length：**

十六进制：0x00 00 00 97

十进制：151

表示该Box长度为28字节,包含了Bod Header和Box Data

**Box type：**

十六进制：0x61 76 63 31

这就是”avc1”的ASCII值，标识了该Box的类型

**Box Data：**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| reserved | 4 | 保留位默认0x00 |
| reserver | 2 | 保留位默认0x00 |
| data reference ID | 2 | 引用参考Dref Box |
| code stream version | 4 | 一般都是默认值0 |
| code stream version | 4 | 一般都是默认值0 |
| reserver | 12 | 保留位一般都是默认值0 |
| width | 2 | 图像宽度 |
| heigth | 2 | 图像高度 |
| horizontal resolution | 4 | 默认值一般都是72dpi(前两个字节整数部分，后面两个字节小数部分) |
| vertical resolution | 4 | 默认值一般都是72dpi(前两个字节整数部分，后面两个字节小数部分) |
| reserver | 4 | 保留位一般都是默认值0 |
| frame count | 2 | 默认值一般是0x00 01  每采样点图像的帧数，一般为1，有些情况下，每个采样点有多帧 |
| compressorname | 32 | 默认填充32字节0即可 |
| depth | 2 | 两个字节默认值是0x00 0x18 |
| reserver | 2 | 两个字节默认值是0xFF0xFF |
| avcc/esds/hvcc | 可变 | 根据视频编码格式，这里填充不同的Box:  如果是avc/h.264编码则构造Avcc Box  如果是MPEG-4编码则构造Esds Box  如果是MPEG-4编码则构造Hvcc Box  这里暂时先分析如果视频编码格式是H.264的Avcc Box |
| btrt | 20 | 描述了解码器需要的Buffer大小和码率相关信息。 |

Box Data的字段已经在上面框出来了，对于默认值的就不一一列举了，现在看下视频的宽和高：

**width:**

十六机制：0x03 00

十进制：768

表示视频图像的宽度是768，这跟前面分析的结论是一致的；

**higth:**

十六机制：0x01 B0

表示视频的高度是432，这跟前面分析的结论是一致的；

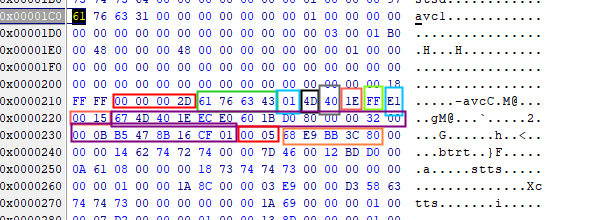
其它基本都是默认值，就不过多分析，现在分析Avcc Box:

**AVCC Box**

**简介:**

该Box则包含了真实的SPS PPS等信息，包含这音视频编解码参数：

**实例分析：**



该Box的头部就不用分析了，和前面的Box一致我，我们分析Box Data即可。

**Box Data：**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| configuration version | 1 | 保留位默认0x01 |
| avc profile indication | 1 | 保留位默认0x00  H.264编码配置：  0x64即100表示Baseline  0x4D即77表示Main |
| profile compatibility | 1 |  |
| avc level indication | 1 | H.264编码级别 |
| reserved | 6 bit | 默认填1 |
| length size minus one | 2 bit | 读出的每个packet的前几字节代表数据大小 |
| reserved | 3 bit | 默认填1 |
| number of sps | 5 bit | sps的个数，一般情况是1 |
| reserved | 3bit | 默认填1 |
| sps length | 2 | sps 长度 |
| sps | sps length | sps实际内容，以后会专门讲解 |
| reserved | 3 bit | 默认填0 |
| number of pps | 5 bit | sps的个数，一般情况是1 |
| pps length | 2 | pps 长度 |
| pps | pps legth | sps实际内容，以后会专门讲解 |

configuration version:

十六进制：0x01

一般默认就是0x01

avc profile indication:

十六进制：0x4D

十进制：77

代表H.264的编码算法采用Main级别，说明含有B帧

profile compatibility:

十六进制：0x40

十进制：64

avc level indication：

十六进制：0x1E

十进制：30

length size minus one:

十六进制：0xFF

二进制：1111 1111

这里后面的bit位为11即3再加1表示每个Packet的长度用几个字节表示。

number of sps：

十六进制：0xE1

二进制：1110 0001

十进制：1

表示后面只有一个SPS.

sps length:

十六进制：0x00 15

十进制：21

说明后面的SPS长度是21字节；

sps:

十六进制：0x67 4D.....16 CF

SPS的实际内容

number of pps：

十六进制：0x01

二进制：0000 0001

十进制：1

表示后面只有一个PPS

sps length:

十六进制：0x00 05

十进制：5

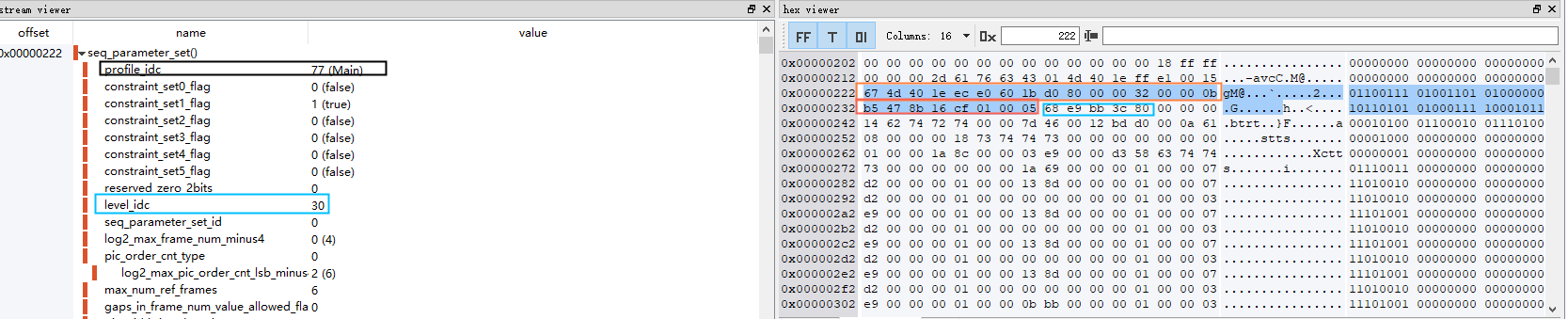
说明后面的PPS长度是5字节；

pps:

十六进制：0x68 E9 BB 3C 80

PPS的实际内容

为了验证AVCC Box的内容我们用Elcard StreamEye进行了交叉验证下：

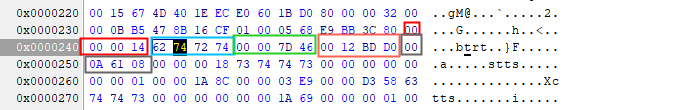


**Btrt Box**

**简介:**

该Box是非必须Box，描述了解码器需要的缓存空间大小和码率描述信息。解码器如果收到该Box，可以在解码时作为参考。

**实例分析：**



**Box Data:**

buffer size db:

十六进制：0x00 00 7D 46

十进制：32070

告诉*decoder*开辟缓冲区大小32070字节

max bitrate:

十六进制：0x00 12 BD D0

十进制：1228240

该路视频的最大码率是1228240bps即1228kb/s

avg bitrate：

十六进制：0x00 0A 61 08

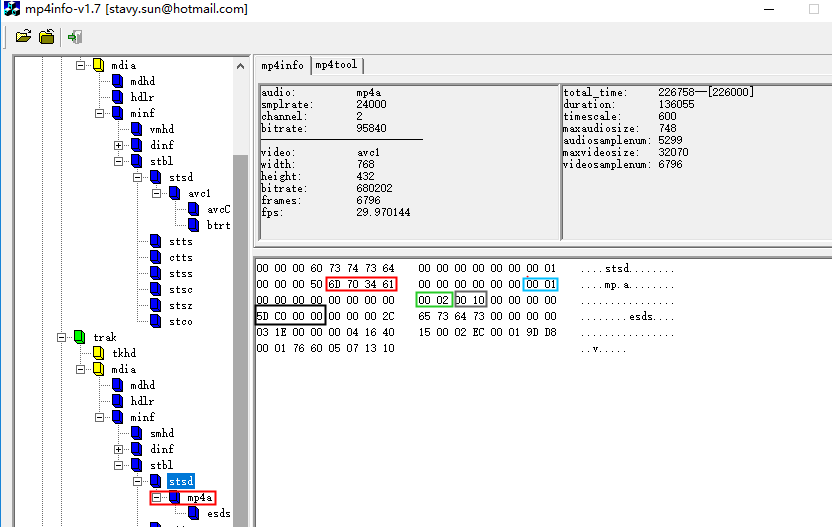
十进制：680200

该路视频的平均码率是680kb/s

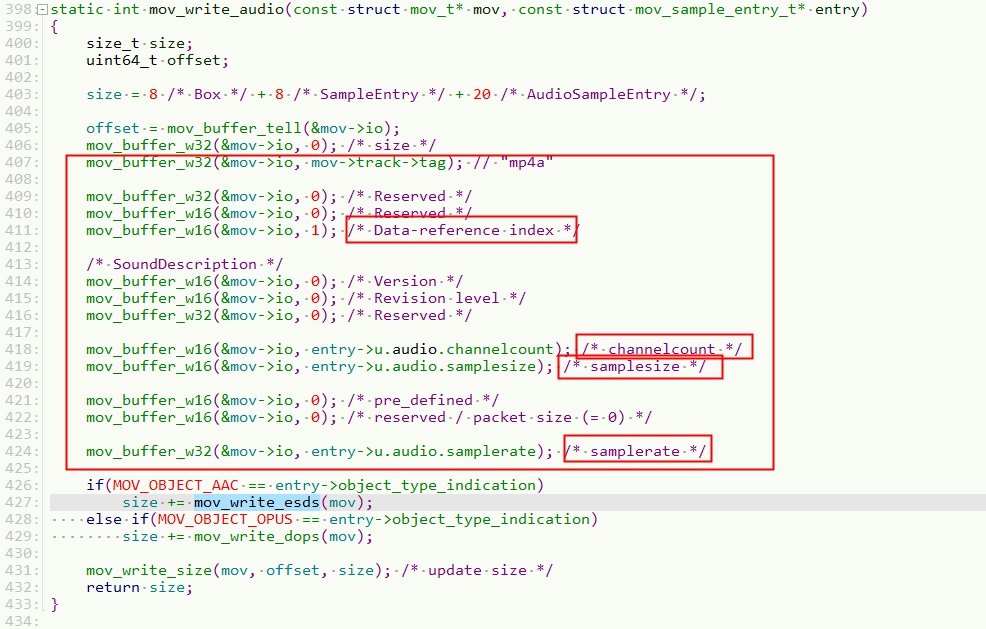
这个数据用MediInfo进行交叉验证了下结果正确：



分析完了Video Track的Stsd Box，下面分析音频的Stsd Box，音频的STSD的Box Data主要由mp4a和esds Box嵌套组成，里面包含了通道个数，采样率、音频AAC编码级别等信息，所以我们只分析音频的这两个嵌套Box即可，先分析mp4a：



各个字段已经圈出来了，然后对照代码看下各个字段解释：



mp4a box 这个box实际就是继承了audio sample entry box里面的关键信息：  
**BoxData:**

 channelcount:

十六进制：0x00 02

十进制：2

音频通道个数是2；

samplesize:

十六进制：0x00 10

十进制：16  
表示音频采样sample的大小为16用两个字节表示，一般默认；

samplerate:

十六进制：0x5D C0 00 00

十进制：24000，需要整体右移16bit进行计算

表示音频采样率为240000Hz

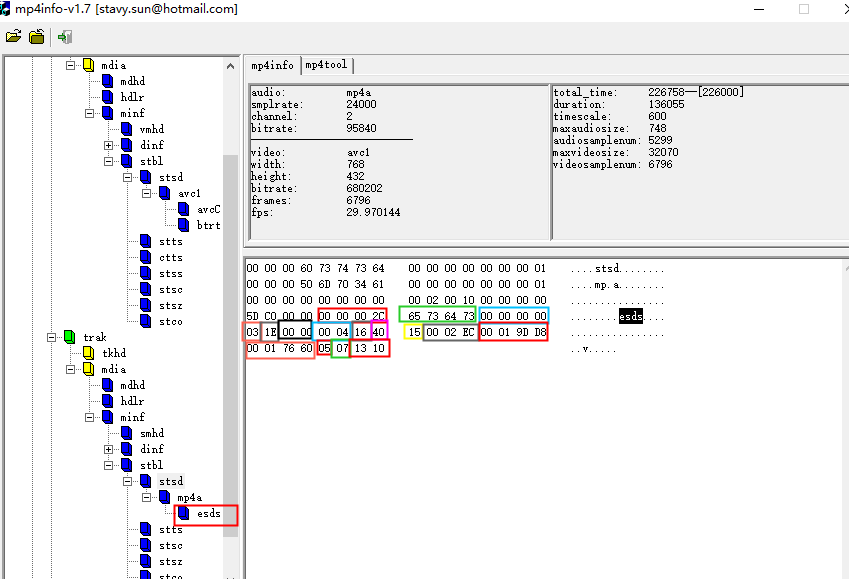
在Box Data的Sample后面还有一个子Esds Box，现在详解下：

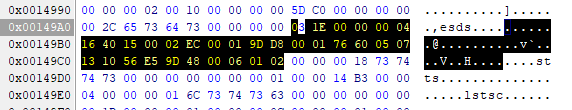
**Esds Box**

**简介：**

这个Box里面主要是将音频的编码信息和音频码率信息放到该Box里面，所以解码音频时非常关键。

**实例分析：**





**Box Header:**

该Box同样是Full Box，所以不做分析，主要分析下Box Data。

**Box Data：**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 字节数 | 意义 | 实际值 |
| es description(ed) tag | 1 | 基本流描述标记：  默认0x03 | 0x03 |
| ed tag szie | 1 | 长度 | 0x1E  30，表示后面有30字节 |
| ed track id | 2 | es id 原始音频流的id | 00 00  0表示音频的原始es数据的id是0，一般一路音频，这个值就默认是0； |
| ed flag | 1 | 一般默认00  :0x00  :00000000  其中每个bit还代表是否后面有相应的字段。  第一bit为1，则有16bit的dependOn\_ES\_IS字段；  第二bit为1，则有8bit的URL ing字段；  第三bit为1，则有16bit的OCR\_ES\_ID字段；  最后5bit代表streamPriority | 0x00  默认值为0 |
| decoder config descriptor(dcd) tag | 1 | 解码配置参数描述标记：默认0x04 | 默认值0x04 |
| dcd tag size | 1 | Length Field长度 | Length Field16 |
| dcd mepg-4 audio | 1 | 如果是mp4则计算得到：  0x40 | 14496-1 Table8, 0x40是Audio ISO/IEC 14496-3 |
| dcd audio stream | 1 | 计算得到默认值：  0x15 | 按照标准或计算得到：此处一般默认0x15 |
| dcd buffersize db | 3 | 建议的解码器缓存大小 | 748字节，最大的Audio Pack大小为748字节 |
| dcd max bitrate | 4 | 音频数据最大码率 | 0x00 01 9D D8  105944bps即约106Kb/s |
| dcd avg bitrate | 4 | 音频数据平均码率 | 0x00 01 76 60  95840即约96Kb/s |
| decoder specific info description(dsid) tag | 1 | 解码规格标记  默认值：0x05 | 0x05 |
| dsid tag szie | 1 | 解码规格标记极其后面值大小 | 0x07 |
| dsid audio specific config(asc) | 2 | 音频规格数据，见下满详细2字节各个bit位解释 | 0x13 10  0001 0011 0001 0000 |
| asc object type | 5bit | MPEG-4 Audio Object Types: 0: Null 1: AAC Main 2: AAC LC (Low Complexity) 3: AAC SSR (Scalable Sample Rate) | 00010  即2  表示采用的音频编码规格为AAC Main |
| asc frequency index | 4bit | 0: 96000 Hz 1: 88200 Hz 2: 64000 Hz 3: 48000 Hz 4: 44100 Hz  5: 32000 Hz  6: 24000 Hz  7: 22050 Hz  8: 16000 Hz  9: 12000 Hz  10: 11025 Hz  11: 8000 Hz  12: 7350 Hz  13: Reserved  14: Reserved  15: frequency is written explictly | 0 110  即6表示采样率24000 |
| asc channel configuration | 4bit | 0: Defined in AOT Specifc Config 1: 1 channel: front-center 2: 2 channels: front-left, front-right 3: 3 channels: front-center, front-left, front-right | 0010  2即表示左右双通道 |
| asc frame length flag | 1bit | 0: Each packet contains 1024 samples 1: Each packet contains 960 samples | 0  每个包的大小为1024字节也就是一帧音频的大小。 |
| asc depends on core coder | 1bit | 暂不关心 | 0 |
| asc extesion flag | 1bit | 暂不关心 | 0 |

**参考链接：**

[**https://wiki.multimedia.cx/index.php?title=MPEG-4\_Audio**](https://wiki.multimedia.cx/index.php?title=MPEG-4_Audio)

**Stts Box**

**简介：**

这个Box是sample number和解码时间DTS之间的映射表，通过这个表格，我们可以找到任何时间的sample。Stts Box这个表格有两项值，其中一项是连续的样点数目即sample count和样点相对时间差值即sample delta即表格中每个条目提供了在同一个时间偏移量里面连续的sample序号以及sample偏移量。当然这里的相对时间差单位是由该Track的Mdhd Box描述的时间单位，这里值为30000，可以参考上文Mdhd Box的解析。

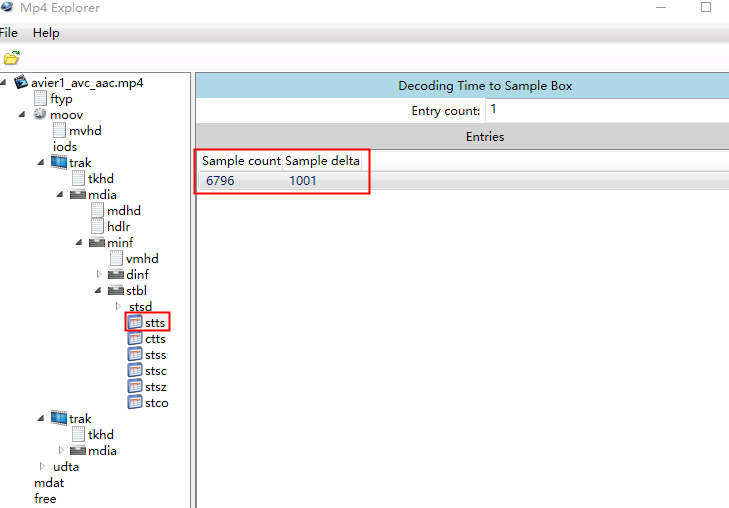
当然用这个表格只是能找到当前时间的sample的序号，但是该sample的长度和指针还要靠其它表格获取。

Sample delta简单可以理解为采样点sample的duration,所有duration相加应该为总的Track的时长。

大家把这个时间理解为该帧数据当前的解码时间即可，计算公式：

DT(n+1) = DT(n) + STTS(n)

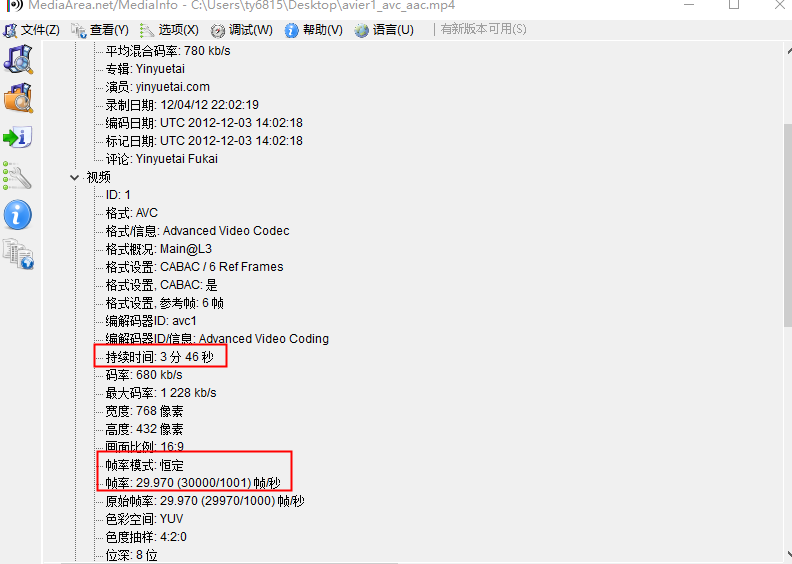
**实例分析：**

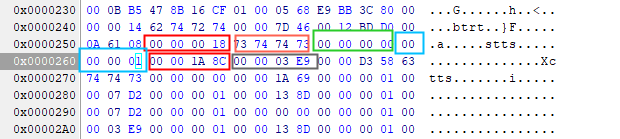


从Mp4Explorer分析来看，这里的表项只有一项，含义是每个sample delta值为6796个sample。从这个表格能得出两项信息是不是固定帧率和帧率计算以及Track 时长。：

1. 这个所有的sample的Sample delta值一样，说明是固定帧率，同时帧率为30000/1001即29.97。
2. 其次可以计算出该Video Track的时长为6796\*1001即6802796，这个值能在mdhd里面已经分析过了，那么换算成时间时长就是6802796/30000即226.75秒；

再次验证下这两个结论：





**Box Header:**

其中该Box是Full Box，后面有四字节的version和flag字段，其中flag默认值用0，这里分析略过。

**Box Data：**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| entry count | 4 | 描述了下面sample conuth和sample delta组成的二元组信息个数。 |
| sample count | 4 | 连续相同时间长度sample delta的sample个数。 |
| sample delta | 4 | 每个sample delta即以timescale为单位的时间长度。  如果所有的sample delta值相同则说明该视频文件是固定帧率。 |

entry count:

十六进制：0x00 00 00 01

十进制：1

说明下面的sample count和sample delta组成的二元组信息只有一对；

sample count:

十六进制：0x00 00 1A BC

十进制：6796

值为sample delta的sample个数为6796个

sample delta:

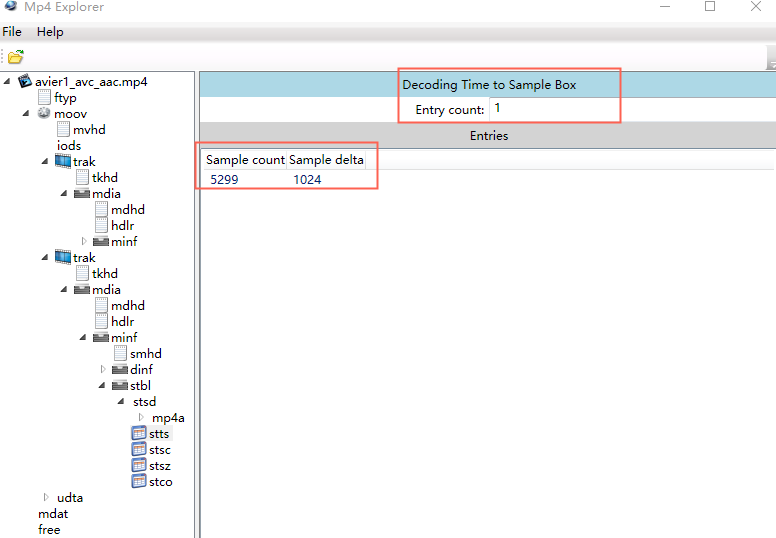
十六进制：0x00 00 03 E9

十进制：1001

Sample的delta值为1001，这样我们计算出帧率就是30000/1001即29.75帧/s,这是一个比较重要的信息。

根据stts box可以计算出每个sample的dts，其中sample delta为该sample的dts相对于上一个sample的差值，比如entry\_count=1，sample\_count=5，sample\_delta=1024时，5个sample的dts将依次为0 1024 2048 3072 4096。

然后分析下音频Track的Stts box:



1. 说明音频的Sample个数是5299个，然后每个Sample的对应DTS差值的1024;
2. 只有一项说明所有帧的音频DTS差值都相等，则（5299\*1024）/24000即226.09秒，和分析结果一致；

Stts Box分析了每个帧的DTS信息，得到了时间到帧的映射关系，但是我们还没有得到PTS的信息，因为存在B帧的情况，这里我们分析Ctts Box.，但是对于音频Track而言是没有Ctts Box的。

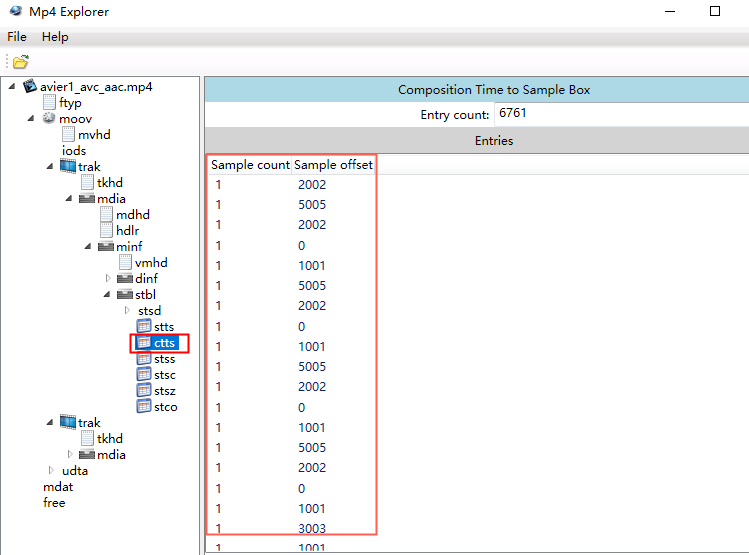
**Ctts Box**

**简介：**

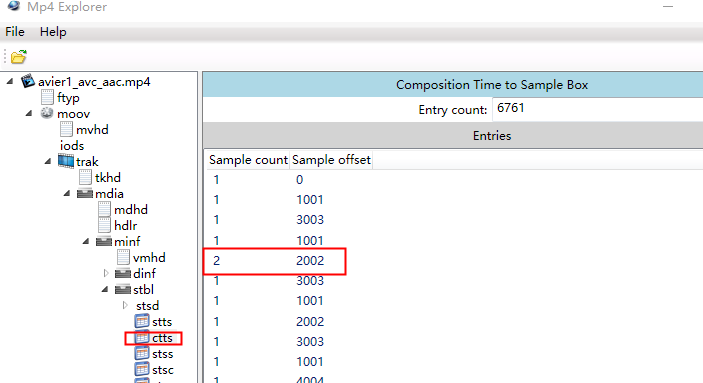
该Box保存了每个sample的composition time和decode time之间的差值，这里通过Composition Time就可以计算出Sample的PTS。

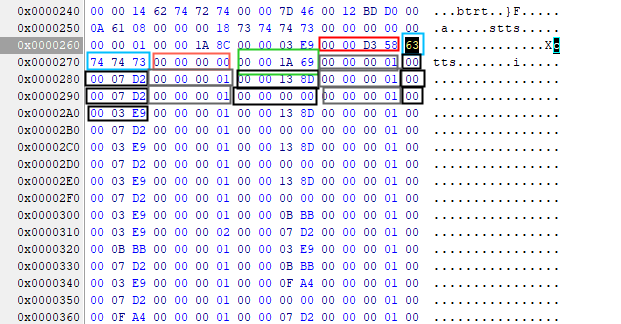
当然在这篇文章中，不存在B帧情况下PTS是等于DTS的，则不含B帧的视频文件是没有Ctts Box的，同样音频也没有Ctts Box。

**实例分析：**



这里我们看到实际的二元组信息为6761，也就是说具有相同Sample offset值的sample个数存在大于1的情况，所以这里的6761小于stts里面分析出来的帧格式6796,这里需要理解好。





**Box Header:**

该Box同样是Full Box，所以不做分析，主要分析下Box Data。

**Box Data：**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| entry count | 4 | 描述了下面sample conuth和sample delta组成的二元组信息个数。 |
| sample count | 4 | 连续相同offset的sample个数 |
| sample offset | 4 | CT和DT之间的offset。 |

entry count:

十六进制：0x00 00 1A 69

十进制：6761

说明下面的sample count和sample offset组成的二元组信息有6791个，所以我们分析前五组即可。

sample count:

十六进制：0x00 00 00 01

十进制：1

具有相同的sample offset连续sample个数为1

sample offset:

十六进制：0x00 00 07 D2

十进制：2002

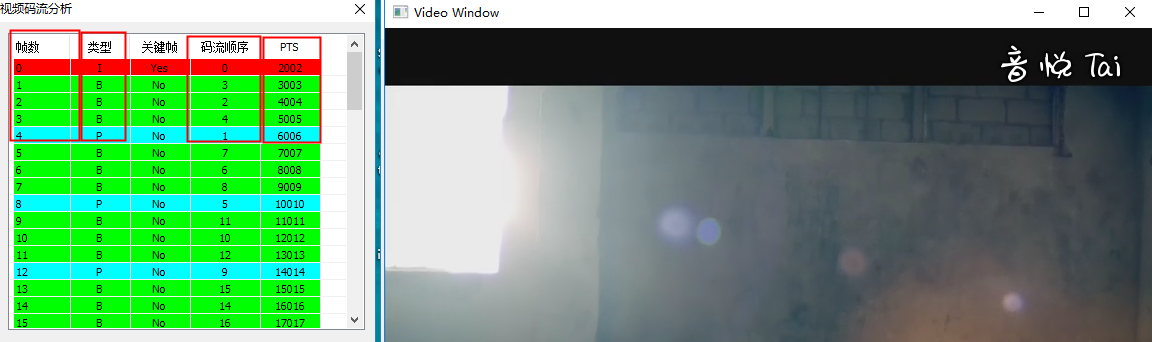
这样我们结合stts进行分析即可以得到前面5个sample的DTS和PTS:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sample num | 0 | 1 | 2 | 3 | 4 |
| Sample detla | 1001 | 1001 | 1001 | 1001 | 1001 |
| Sample offset | 2002 | 5005 | 2002 | 0 | 1001 |
| DTS | 0 | 1001 | 2002 | 3003 | 4004 |
| PTS | 2002 | 6006 | 4004 | 3003 | 5005 |

这样我们就知道对应sample的解码和显示顺序，为了验证猜想我用码流分析工具实际解码播放看了下解码PTS的大小关系：

解码：0-1-2-3-4

显示：0-3-2-4-1



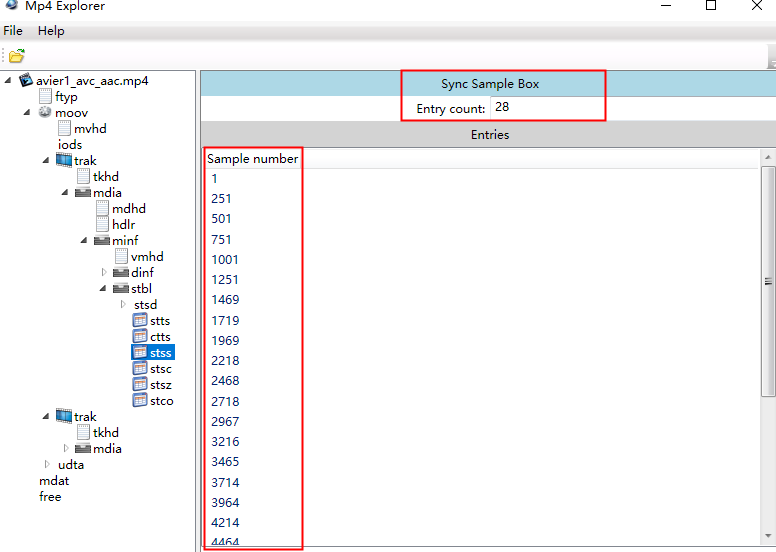
只有真正了理解这块才真正理解了stts和ctts对于DTS和PTS的计算方法，这是实现按顺序解码和播放关键也是音视频同步的核心内容。

**Stss Box**

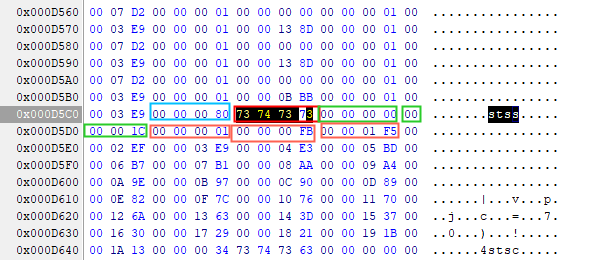
以前的文章已经讲过I帧是播放的起始位置，只有编码器拿到第一个I帧才能渲染出第一幅画面。所以后续的一些特殊随机操作，高标清切换时都需要找I帧，只有随机找到I帧才能完成这些特殊操作。

其中该Box就是存储了那些Sample是I关键帧，很显然音频Track也是不存在这个Box的。

**实例分析：**



通过Mp4Explor分析工具可以得出是I帧的Sample序号，下面看Box的实际数据：



该Box同样是Full Box，所以不做分析，主要分析下Box Data。

**Box Data：**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| **entry count** | 4 | 关键帧的sample个数 |
| **sample count** | 4 | 关键帧sample序号 |

entry count:

十六进制：0x00 00 00 1C

十进制：28

说明本文件大概有28个关键帧

sample count:

十六进制：0x00 00 00 01

十进制：1

继续分析则关键帧的Sample序号如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 是否关键帧 | Yes | Yes | Yes | Yes | Yes |
| 十六进制 | 0x00 00 00 01 | 0x00 00 00 FB | 0x00 00 01 F5 | 0x00 00 02 EF | 0x00 00 03 E9 |
| 十进制 | 1 | 251 | 501 | 751 | 1001 |

**Stsc Box**

**简介：**

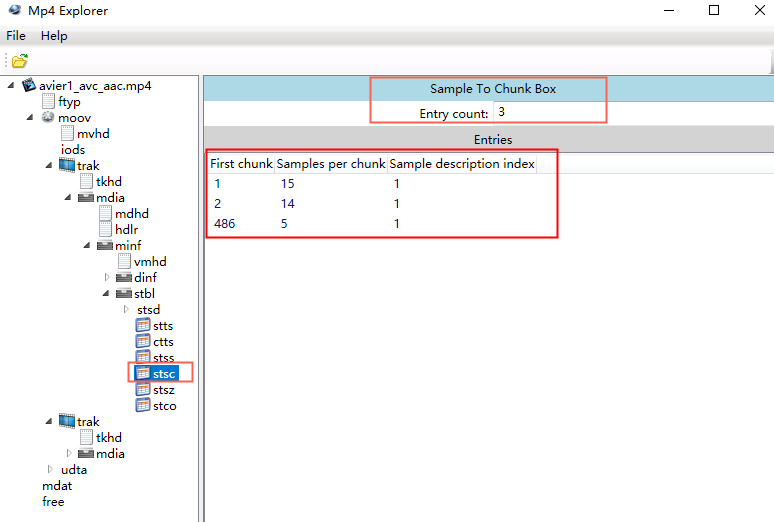
上篇文章讲解了MP4中存在Track-Trunk-Sample概念，讲到现在Track和Sample已经都讲到了，但是还没有讲解Trunk和Sample的映射关系，这个Box就是说明那些Sample可以划分为一个Trunk。

媒体数据被分为若干个chunk, chunk可以有不同的大小，同一个chunk中的样点sample也允许有不同的大小；通过本表可以定位一个样点的chunk位置,同时该Box里面的Box Data里面有三个字段，分别是first chunk、sample per chunk、sample description index。

1. fist chunk: 具有相同采样点sample和sample\_description\_index的chunk中，第一个chunk的索引值,也就是说该chunk索引值一直到下一个索引值之间的所有chunk都具有相同的sample个数，同时这些sample的描述description也一样；
2. samples per chunk: 上面所有chunk的sample个数
3. sample description index: 描述采样点的采样描述项的索引值，范围为1到样本描述表中的表项数目；

这 3 个字段实际上决定了一个 MP4 中有多少个 chunks，每个 chunks 有多少个 samples。

**实例分析：**



这说明三元组信息（first chunk、sample per chunk、sample desciption index）的个数只有3个，其中的意义：

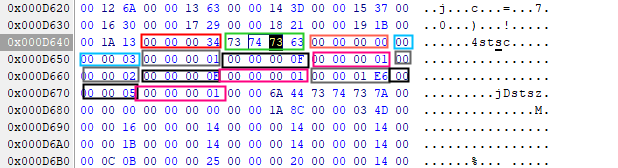
第一个chunk（1），每个chunk的sample的sample个数有15个，同时这些sample description index都是1；

第二个到四百八十六（2-485）这里面的所有chunk，每个chunk的sample个数有14个，同时这些sample description index都是1；

第486即最后一个chunk的sample个数有5个，同时这些sample description index都是1；

从上面的表中我们可以得出下面信息：

1. Chunk总数是486个；
2. Sample的总个数是1\*15 + （486-2）\*14 + 1\*5 即 6796个sample，这和Stts Box分析出来的结果是一致的；
3. 如果只有first chunk，这种情况，通过stts得出sample的总数，通过total / sample per chunk = num of chunk



**Box Data：**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| entry count | 4 | 三元组信息的个数 |
| first chunk | 4 | 每一个 entry 开始的 chunk 位置 |
| sample per chunk | 4 | 每一个 chunk 里面包含多少的 sample |
| sample description index | 4 | 每一个 sample 的描述。一般可以默认设置为 1。 |

entry count:

十六进制：0x00 00 00 03

十进制：3

说明本文件三元组信息为3个，循环解析三次即可；

first chunk:

十六进制：0x00 00 00 01

十进制：1

具有相同sample个数和description的chunk起始索引是1；

sample per chunk：

十六进制：0x00 00 00 0F

十进制：15

每个chunk具有的sample的个数是15；

sample description index：

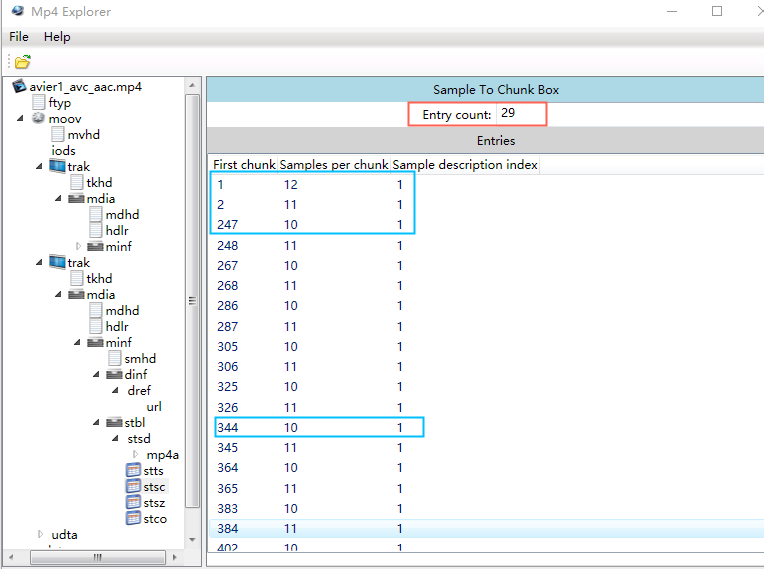
十六进制：0x00 00 00 01

十进制：1

这些sample的description index都是1，一般用默认值即可；

其它三元组信息分析大同小异，这里不再赘述。

顺便看下音频Track的各个音频Sample和chunk的映射关系：

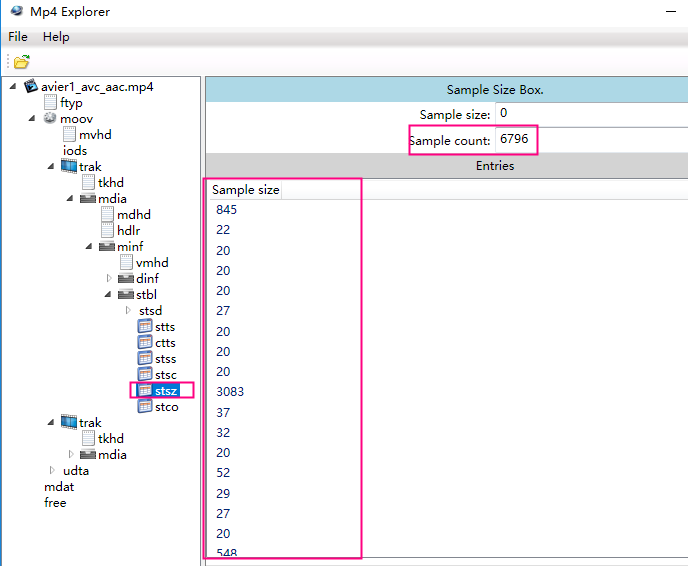


**Stsz Box:**

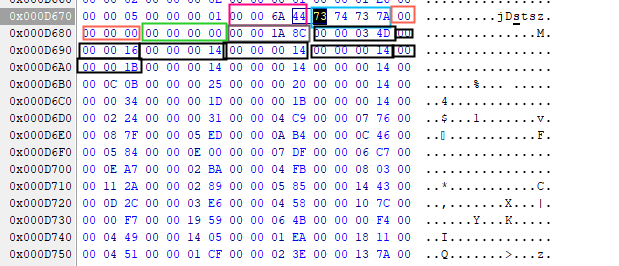
前面分析了sample的PTS、DTS等，也分析了chunk里面sample的信息，但是没有分析sample的大小，这是我们在文件读取和解析Sample的关键。这里给出每个Sample的Size即包含的字节数。

包含了媒体中全部sample的数目和一张给出每个sample大小的表。这个box相对来说体积是比较大的。

**实例分析：**



从上到下就是6796个Sample的大小。



**Box Header:**

此Box是Full Box，同样含有version和flag字段，其中flag为0。

**Box Data:**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| sample size | 4 | 指定默认的sample大小，如果每个sample大小不相等，则这个字段值为0，每个sample大小存在下表中。 |
| sample count | 4 | 该track中所有sample的数量 |
| entry size | 4 | 每个sample的大小 |

sample size:

十六进制：0x00 00 00 00

十进制：0

说明sample的具体大小在表后面项目中entry size；

sample count:

十六进制：0x00 00 1A 8C

十进制：6796

说明本Track的大小为6789个

entry size:

十六进制：0x00 00 03 4D

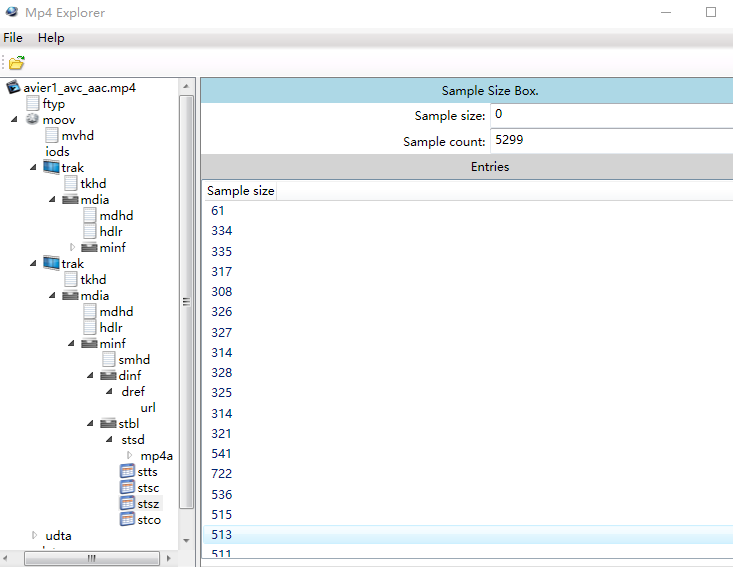
十进制：845

说明第一个sample大小为845字节

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sample num | 1 | 2 | 3 | 4 |
| entry size | 0x00 00 03 4D(845) | 0x00 00 00 16(22) | 0x00 00 00 14(20) | 0x00 00 00 14(20) |

现在我们分析了sample的大小，也分析了sample和chunk的映射关系，现在分析最后一个Stbl的子Box，该Box将给出每个Trunk在文件的偏移量。

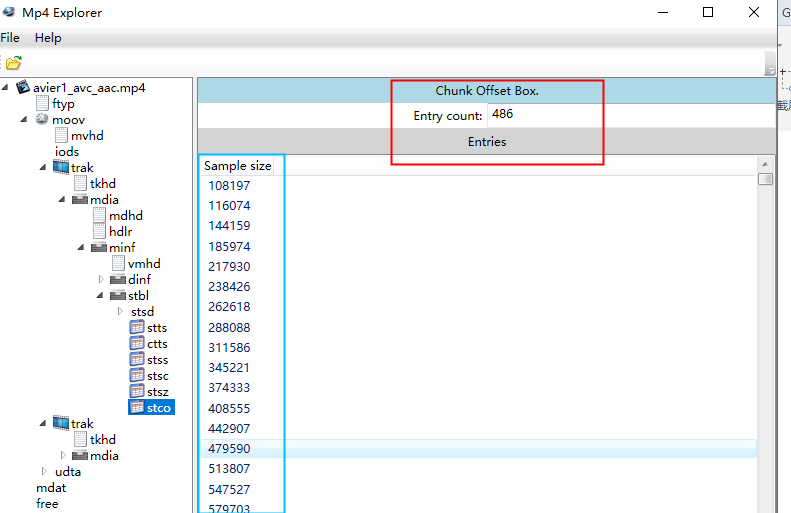
分析下音频Track 的Stsz Box各个Sample的大小：



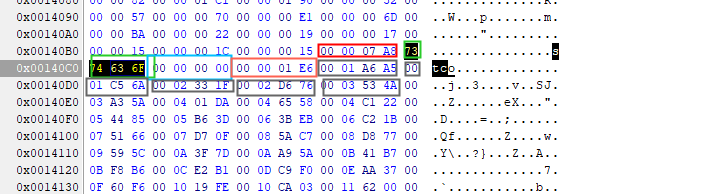
**Stco Box**

**简介:**

该Box存储了Chunk Offset，表示了每个Chunk在文件中的位置，这样我们就能找到了chunk在文件的偏移量，然后根据其它表的关联关系就可以读取每个Sample的大小。



这张表中我们发现了有chunk的数量是486个，这和Stsc Box分析得出的结论是一致的。同时给出了每个chunk的在文件的偏移量。



**Box Header:**

此Box是Full Box，同样含有version和flag字段，其中flag为0。

**Box Data：**

|  |  |  |
| --- | --- | --- |
| 字段 | 字节数 | 意义 |
| entry count | 4 | chunk的总个数 |
| chunk offset | 4 | 每个chunk在文件中的位置 |

entry count:

十六进制：0x00 00 01 E6

十进制：486

说明本文件的chunk数量总和是486。

Chunk offset:

十六进制：0x00 01 A6 A5

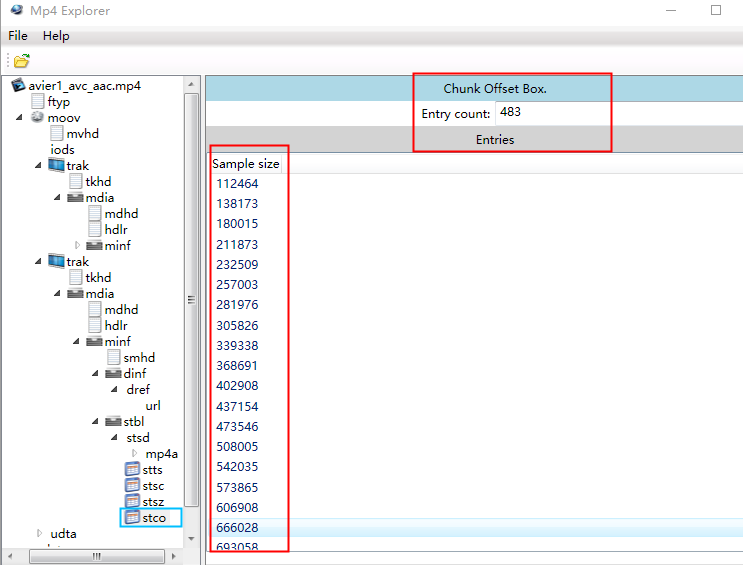
十进制：108197

说明本文件第一个chunk的偏移量是6575字节,我们再分析了后面四个chunk的偏移量：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| chunk index | 1 | 2 | 3 | 4 |
| chunk size | 0x00 01 A6 A5**(108197)** | 0x00 01 C5 A6**(116134)** | 0x00 02 33 1F**(144159)** | 0x00 02 D6 76**(185974)** |

我们发现是和上面工具分析的结果是一致的。

音频Track的Stco Box顺便分析下：



stco 有两种形式，如果你的视频过大的话，就有可能造成 chunk offset 超过 32bit 的限制。所以，这里针对大 Video 额外创建了一个 co64 的 Box。它的功效等价于 stco，也是用来表示 sample 在 mdat box 中的位置。只是，里面 chunk\_offset 是 64bit 的。

**H.264数据打包MP4文件步骤：**

1. 构造Ftyp Box，这是MP4文件的基本标识，基本按照规范构造即可；
2. 构造Mdat Box，这里面都是音视频媒体数据，将NALU数据封装成一个个Sample，从上到下排列起来即可；
3. 构造Moov Box,这个Box，先构造除了stbl的其它Box,这样从子Box，一直按照层次构造上来，这些Box的基本信息是明确的，个别信息通过解析SPS、PPS是可以明确知道的，所以上文已经明确了这些Box的字段，所以构造起来并不难；
4. 构造Moov Box的Stbl部分，这部分要封装：

SPS、PPS的NALU数据

也要构造时间戳PTS和DTS信息

Sample的大小和数量

Chunk的数量和在文件偏移量

Sample和Chunk的映射关系等，

需要仔细处理，一旦构造错误都会播放失败；

1. 其中Moov到底是放到Mdat Box前面还是后面，需要在封装打包前确定后，因为会牵一发动全身，里面的信息都依赖在具体文件的位置，至于前面后面区别见下篇文章；
2. 再填充一些你感兴趣的非必要信息Box，至此组装成整个MP4文件即可；

MP4的解封装代码比较长，看了几个开源项目都有上万行的代码量，不过只要严格按照ISO规范进行书写，问题不会太大，当然该关心H265、Av1等编码格式的打包，其实流程基本差不多，后面会慢慢介绍给大家，但是这篇是基础。

**总结：**

截止到这里我们分析完了所有核心Box的内容，包含了Box的作用，语法结构，各个字段值。同时用多种工具进行了交叉验证分析的结果。当然Box的总数量要达到70多个，其它都不是必须Box,如果后面用到会继续分享出来。