# MP4文件格式

### 综述

在MP4文件格式中，整个视频容器都是由多个box和子box组成，根据box类型主要分为3大类：视频类型（ftyp）、视频数据（mdat）、视频信息（moov）。视频信息（moov）用来描述视频数据（mdat）。（注：还有一个主要box为moof box，因这里仅解释普通MP4格式数据，moof box仅在流式Mp4中使用。在流式MP4格式中，box排序、相同box的box body内容格式与普通MP4不一样，详情参见[扩展](#_流式MP4)）

视频参数（moov）中主要的子box 为track，每个track都是一个随时间变化的媒体序列，时间单位为一个sample，可以是一帧数据，或者音频（注意，一帧音频可以分解成多个音频sample，所以音频一般用sample作为单位，而不用帧）。Sample按照事件顺序排列。track里面的每个sample通过引用关联到一个sample description。这个sample descriptios定义了怎样解码这个sample，例如使用的压缩算法。（注：在目前的使用中，该值为1）

注：该文主要介绍普通mp4文件类型

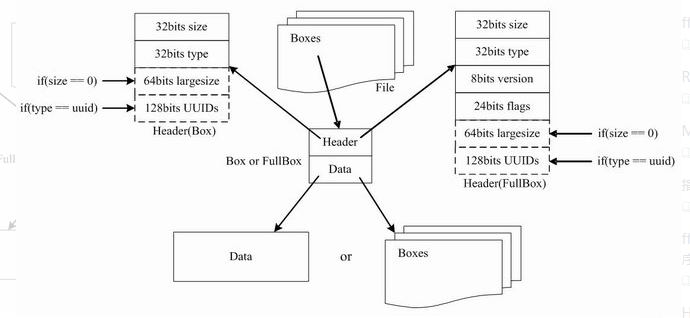
### MP4.box

在Javascript中，Mp4 的所有box全部由通过new Uint8Array() 实现。

box前8位为预留位，这8位中前4位为数据size，当size值为0时，表示该box为文件的最后一个box（仅存在于[mdat box](#_Mdat box)中），当size值为1时，表示该box的size为large size（8位），真正的box size要在largesize中得到（同样仅存在于[mdat box](#_Mdat box)中）。后4位为前面box type的Unicode 编码。当type是uuid时，代表Box中的数据是用户自定义扩展类型。

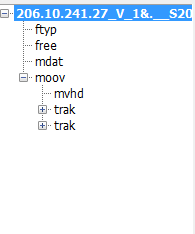
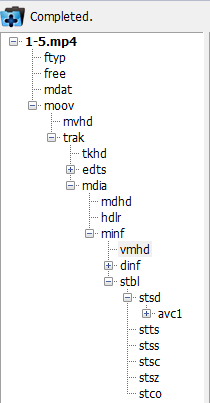
Box由header和body组成，以32位的4字节整数存储方式存储到内存，开头4个字节（32位）为box size，后面紧跟的4位为box的类型。Box body可以由数据组成，也可以由子box组成。

一个box的结构如下：



视频与音频的参数不一样，一般情况下一个MP4文件区分为2个trak，一个为video trak，另一个是audio trak。每个track都有trakId，视频的trakId为1，音频的trakId为2。

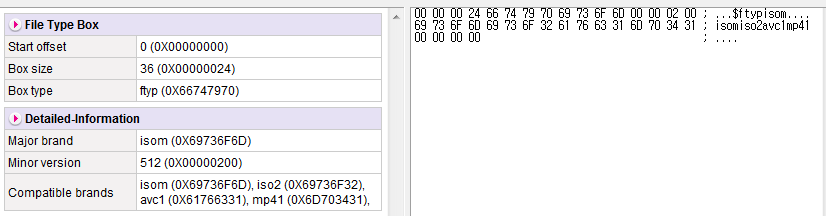
整个MP4文件格式如下图



### FTYP box

Ftypbox 是一个由四个字符组成的码字，用来表示编码类型、兼容协议或者媒体文件的用途。

在普通MP4文件中，ftyp box有且仅有一个，在文件的开始位置。

通过MP4reader工具，可以看出ftyp box 的结构

Box size（4字节）：0x00000024：box的长度是36字节；

Boxt type（4字节）：0x66747970：“ftyp”的ASCII码，box的类型；

major\_brand（4字节）：0x69736f6d：“isom“的ASCII码；

minor\_version（4字节）：0x00000200：isom的版本号；

compatible\_brands（12字节）：说明该文件兼容isom, iso2, avc1, mp41 四种协议。

Ftyp更多兼容协议 ： <http://www.ftyps.com/>

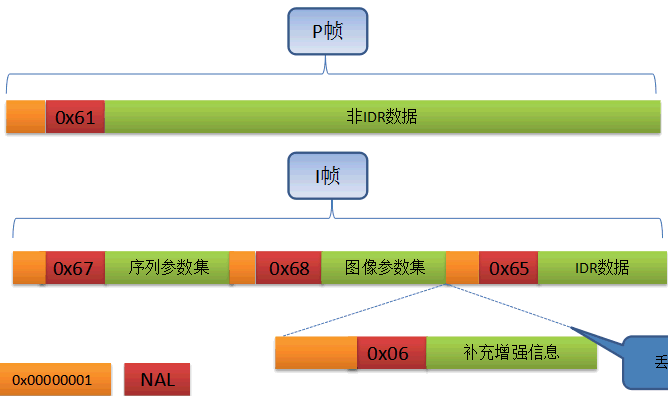
### Mdat box

Mdat box中包含了MP4文件的媒体数据，在文件中的位置可以在moov的前面，也可以在moov的后面，因我们这里用到mp4文件格式用来写mp4文件，需要计算每一帧媒体数据在文件中的偏移量，为了方便计算，mdat放置moov前面。

Mdat box数据格式单一，无子box。主要分为box header 和box body，box header中存放box size 和box type（mdat），box body中存放所有媒体数据，媒体数据以sample为数据单元。

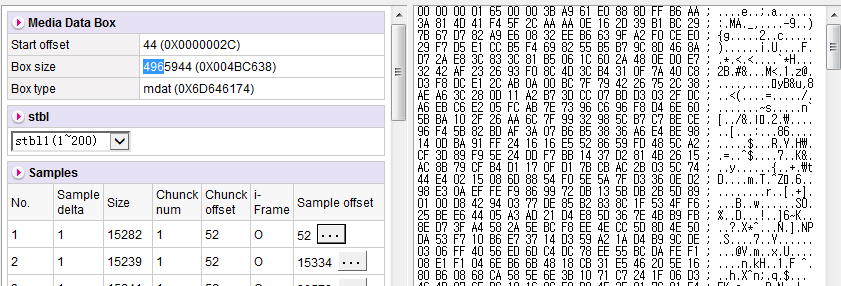
这里使用时，视频数据中，每一个sample是一个视频帧，存放sample时，需要根据帧数据类型进行拼帧处理后存放。

H.264视频帧数据类型如下：



注：1、在目前实现中，I帧数据中暂不包含序列参数集（sps）和图像参数集（pps）。

2、以上帧数据仅针对视频帧数据。

在普通mp4中，在获取数据之前，需要解析每个帧数据所在位置，每个帧数据都存放在mdat中，而这些帧的信息全部存放在stbl box 中，所以，若要mp4文件能够正常播放，需要在写mp4文件时，将所有的帧数据信息写入 stbl box中。

Mdat box中，可能会使用到box的large size，当数据足够大，无法用4个字节来描述时，便会使用到large size。在读取MP4文件时，当mdat box的size位为1时，真正的box size在large size中，同样在写mp4文件时，若需要large size，需要将box size位配置为1。

### Moov box

Moov box中存放着媒体信息，上面提到的stbl里存放帧信息，属于媒体信息，也在moov box里。Moov box 用来描述媒体数据。

Moov box 主要包含 mvhd、trak、mvex三种子box。

### Mvhd box

Mvhd box定义了整个文件的特性

|  |  |  |
| --- | --- | --- |
| 字段 | 长度(字节) | 描述 |
| 尺寸 | 4 | 这个movie header atom的字节数 |
| 类型 | 4 | Mvhd |
| 版本 | 1 | 这个movie header atom的版本 |
| 标志 | 3 | 扩展的movie header标志，这里为0 |
| 生成时间 | 4 | Movie atom的起始时间。基准时间是1904-1-1 0:00 AM |
| 修订时间 | 4 | Movie atom的修订时间。基准时间是1904-1-1 0:00 AM |
| Time scale | 4 | 本文件的所有时间描述所采用的单位 |
| Duration | 4 | 媒体可播放时长 |
| 播放速度 | 4 | 播放此movie的速度。1.0为正常播放速度 |
| 播放音量 | 2 | 播放此movie的音量。1.0为最大音量 |
| 保留 | 10 | 这里为0 |
| 矩阵结构 | 36 | 该矩阵定义了此movie中两个坐标空间的映射关系 |
| 预览时间 | 4 | 开始预览此movie的时间，写文件时该值为0 |
| 预览duration | 4 | 以movie的time scale为单位，预览的duration，写文件时该值为0 |
| Poster time | 4 | The time value of the time of the movie poster. |
| Selection time | 4 | The time value for the start time of the current selection. |
| Selection duration | 4 | The duration of the current selection in movie time scale units. |
| 当前时间 | 4 | 当前时间 |
| 下一个track ID | 4 | 下一个待添加track的ID值。0不是一个有效的ID值。 |

这里写mp4时需要传入的参数为Time scale 和 Duration，其他的使用默认值即可。

### Trak box

一个Track box定义了movie中的一个track。一部movie可以包含一个或多个tracks，它们之间相互独立，各自有各自的时间和空间信息。每个track box 都有与之关联的mdat box。

Track主要有以下目的：

1. 包含媒体数据引用和描述
2. 包含modifier track
3. 流媒体协议的打包信息（hint trak），引用或者复用对应的媒体sample data。

Hint tracks和modifier tracks必须保证完整性，同时和至少一个media track一起存在。换句话说，即使hint tracks复制了对应的媒体sample data，media tracks 也不能从一部hinted movie中删除。

写mp4时仅用到第一个目的，所以这里只介绍媒体数据的引用和描述。

一个trak box一般主要包含了tkhd box、 edts box 、mdia box

### Tkhd box

用来描述trak box的header 信息，定义了一个trak的时间、空间、音量信息。

|  |  |  |
| --- | --- | --- |
| 字段 | 长度(字节) | 描述 |
| 尺寸 | 4 | 这个atom的字节数 |
| 类型 | 4 | tkhd |
| 版本 | 1 | 这个atom的版本 |
| 标志 | 3 | 有效的标志是  ·                     0x0001 - the track is enabled  ·                     0x0002 - the track is used in the movie  ·                     0x0004 - the track is used in the movie’s preview  ·                     0x0008 - the track is used in the movie’s poster |
| 生成时间 | 4 | Movie atom的起始时间。基准时间是1904-1-1 0:00 AM |
| 修订时间 | 4 | Movie atom的修订时间。基准时间是1904-1-1 0:00 AM |
| Track ID | 4 | 唯一标志该track的一个非零值。 |
| 保留 | 4 | 这里为0 |
| Duration | 4 | 该track的时长，若该trak为videotrak，其时长来源于elst，若无elst，则取mvhd的时长 |
| 保留 | 8 | 这里为0 |
| Layer | 2 | The track’s spatial priority in its movie. The QuickTime Movie Toolbox uses this value to determine how tracks overlay one another. Tracks with lower layer values are displayed in front of tracks with higher layer values. |
| Alternate group | 2 | A collection of movie tracks that contain alternate data for one  another. QuickTime chooses one track from the group to be used when the movie is played.  The choice may be based on such considerations as playback quality, language, or the capabilities of the computer. |
| 音量 | 2 | 播放此track的音量。1.0为正常音量 |
| 保留 | 2 | 这里为0 |
| 矩阵结构 | 36 | 该矩阵定义了此track中两个坐标空间的映射关系 |
| 宽度 | 4 | 如果该track是video track，此值为图像的宽度，若为audio，为0 |
| 高度 | 4 | 如果该track是video track，此值为图像的高度，若为audio，为0 |

### Elst box

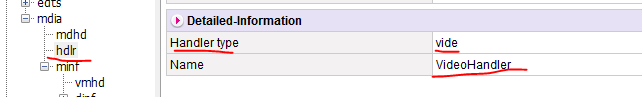
该box为edst box的唯一子box，不是所有的MP4文件都有edst box，这个box是使其对应的trak box的时间戳产生偏移。暂时未发现需要该偏移量的地方，编码时也未对该box进行编码。

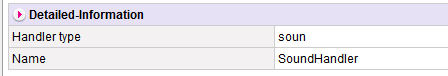
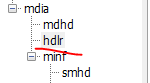
### Mdia box

该box定义了trakbox的类型和sample的信息。

其header box---**mdhd box**定义了该box的timescale 和duration（注：这里的这两个参数与前面说的mvhd有区别，这里的这两个参数都是以一个sample为时间单位的，例：在只有一个视频trak的情况下，mvhd的timescale为1000，一个sample的duration为40 ，那么这里的timescale为1000/40，同理这里的duration算法与之一样理解。）

**Hdlr box** 定义了这段trak的媒体处理组件，以下图会更清晰的解释这个box





### Minf box

该box也是上面的mdia box的子box，其主要用来描述该trak的具体的媒体处理组件内容的。

其header box根据trak的类型有2种，vmhd 和smhd，两者没有什么特殊的数据，只是为了定义headle的类型。

其子box---dinf box 用来定义媒体处理组件如何获取媒体数据的，dinf box的子box --- dref box用来定义数据引用方式，这里使用时无需使用该box，因此这里不做详细解释，虽然不使用该box，但是在编码mp4 文件时，该box 为必选项，只不过不使用时将dref中的引用方式的数量默认为0，其引用的信息默认为url且为空即可。

### Stbl box

Sample Table Box（stbl）是上面minf的子box之一，用来定义存放时间/偏移的映射关系，数据信息都在以下子box中

**stts**: Time to Sample Box 时间戳和Sample序号映射表

**stsd**: Sample Description Box用来描述数据的格式，比如视频格式为avc，比如音频格式为aac



**stsz, stz2**: Sample Size Boxes 每个Sample大小的表。Stz2是另一种sample size的存储算法，更节省空间，使用时使用其中一种即可，这里使用stsz。原因简单，因为算法容易。

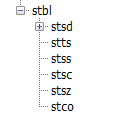
**stsc**: Sample to chunk 的映射表。这个算法比较巧妙，在多个chunk时，该算法较为复杂。在本次使用中未考虑多个chunk的状态，仅考虑整个文件单个chunk的情况。

**stco, co64**: 每个Chunk位置偏移表，sample的偏移可根据其他box推算出来，co64是指64位的chunk偏移，暂时只使用到32位的，因此这里使用stco即可。

**stss**：关键帧序号，该box存在于video trak，因为audio trak 中以sample为单位，但多个sample才组成一帧音频，所以在audio trak中无需该box。

以上子box在MP4编码中尤为重要，具体介绍在[实例](#_实例：)中解释

结构图如下：

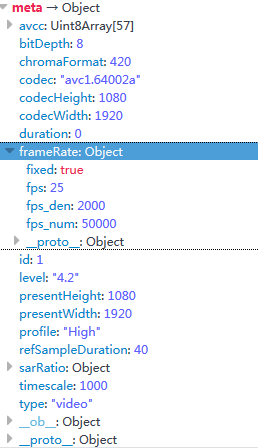


## 实例：

以从url中接收到的一段经过解封装后的视频数据的分析

解封装的方法 \_parseChunks

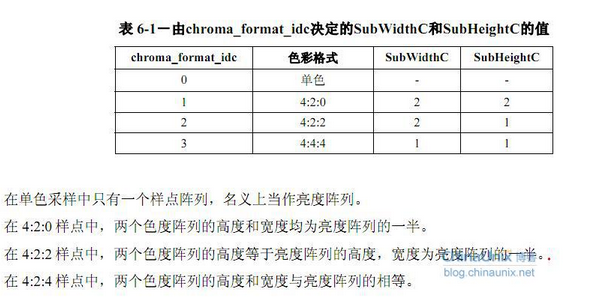
解封装后的数据如下



以上数据为视频数据，大部分来源于flv视频流数据中的sps。

**Id：** 这里的id是在解码时写死的，当是视频段数据，id=1，音频，id=2

**chromaFormat** ：色彩采样格式



**bitDepth**：图像灰度

8 : 256色位图

24 ： 真彩色

**Level** ： leve\_idc 比特流所遵守的级别

**profile**： profile\_idc 比特流所遵守的配置

MP41.types = {  
 avc1: [], avcC: [], btrt: [], dinf: [],  
 dref: [], esds: [], ftyp: [], hdlr: [],  
 mdat: [], mdhd: [], mdia: [], mfhd: [],  
 minf: [], moof: [], moov: [], mp4a: [],  
 mvex: [], mvhd: [], sdtp: [], stbl: [],  
 stco: [], stsc: [], stsd: [], stsz: [],  
 stts: [], tfdt: [], tfhd: [], traf: [],  
 trak: [], trun: [], trex: [], tkhd: [],  
 vmhd: [], smhd: [], '.mp3': [], free: [],  
 edts: [], elst: [], stss: []  
};

一个MP4文件中有以上种类型，MP4.types中的每个type都是一个将type的每个字符转成 Unicode 编码的值，供后续重封装时使用。关于box详见[MP4.box](#_MP4.box)

注：因为这里的解封装和重封装都是对flv的一个tag进行操作，所以音频和视频的数据时分开操作的。

通过flv解析后的一个sample数据如下：

{  
 dts: dts,  
 pts: pts,  
 cts: cts,  
 units: units,  
 size: sample.length,  
 isKeyframe: isKeyframe,  
 duration: sampleDuration,  
 originalDts: originalDts,  
 flags: {  
 isLeading: 0,  
 dependsOn: isKeyframe ? 2 : 1,  
 isDependedOn: isKeyframe ? 1 : 0,  
 hasRedundancy: 0,  
 isNonSync: isKeyframe ? 0 : 1  
 }

里面在编码mp4文件时重点使用的参数有units，isKeyframe，写入mdat的数据来源于每个sample数据中的units，在存储sample数据时需要注意对象的浅拷贝，因为若是使用了浅拷贝，units数据在停止录像时会被置空，这里使用了es6的深拷贝方法

Object.assign({}, sample.units[i])

Units是一个数组，所以对其使用遍历深拷贝。

在拷贝数据前需要对unit数据做拼帧处理

let DRFlag = new Uint8Array(5);  
if (singleSample.isKeyframe === true) {  
 let spsFlag = new Uint8Array([0x00, 0x00, 0x00, 0x01, 0x67]);  
 let ppsFlag = new Uint8Array([0x00, 0x00, 0x00, 0x01, 0x68]);  
 let IDRFlag = new Uint8Array([0x00, 0x00, 0x00, 0x01, 0x65]);  
 let spsFlagLen = 5, ppsFlagLen = 5, IDRFlagLen = 5, spsMetaLen = this.spsMeta.byteLength, ppsMetaLen = this.ppsMeta.byteLength;  
 DRFlag = new Uint8Array(spsFlagLen + spsMetaLen + ppsFlagLen + ppsMetaLen + IDRFlagLen);  
 DRFlag.set(spsFlag, 0);  
 DRFlag.set(this.spsMeta, spsFlagLen);  
 DRFlag.set(ppsFlag, spsFlagLen + spsMetaLen);  
 DRFlag.set(this.ppsMeta, spsFlagLen + spsMetaLen + ppsFlagLen);  
 DRFlag.set(IDRFlag, spsFlagLen + spsMetaLen + ppsFlagLen + ppsMetaLen);  
} else if (singleSample.isKeyframe === false) {  
 DRFlag = new Uint8Array([0x00, 0x00, 0x00, 0x01, 0x61]);  
}// ***todo 音频***

let unitData = new Uint8Array(units[i].data.byteLength + 5);  
unitData.set(DRFlag, 0);  
unitData.set(units[i].data, 5);  
units[i].data = new Uint8Array(unitData.byteLength);  
units[i].data.set(unitData, 0);

最后使用编码mp4文件时需要将这些数据全部通过box方法转化成4位32进制存储，其中需要传入的参数有两个，一个是上面的视频参数，另一个是sample列表。因为在js中写数据时需要先写数据长度，那么还需要传一个拼帧后的sample中unit data的总长度，这个长度也是在存储sample列表时同时进行处理的。

let mdatbox = new Uint8Array(*mdatBytes* + 8);

所以传参有3个：

*meta*, *mdatDataList*, *mdatBytes*

Box的写法：

static box(*type*) {  
 let size = 8;  
 let result = null;  
 let datas = Array.prototype.slice.call(arguments, 1);  
 let arrayCount = datas.length;  
  
 for (let i = 0; i < arrayCount; i++) {  
 size += datas[i].byteLength;  
 }  
 result = new Uint8Array(size);  
 result[0] = (size >>> 24) & 0xFF; // size  
 result[1] = (size >>> 16) & 0xFF;  
 result[2] = (size >>> 8) & 0xFF;  
 result[3] = (size) & 0xFF;  
  
 result.set(*type*, 4); // type  
  
 let offset = 8;  
 for (let i = 0; i < arrayCount; i++) { // data body  
 result.set(datas[i], offset);  
 offset += datas[i].byteLength;  
 }  
  
 return result;  
}

Type是box的类型，方法中的第三行表示获取参数中除去第一个参数的其他参数，box的参数除了第一个为类型，其他参数都需要是二进制的arraybuffer类型。

编写mp4文件blob数据的方法：

static generateInitSegment(*meta*, *mdatDataList*, *mdatBytes*) {  
  
 let ftyp = MP41.box(MP41.types.ftyp, MP41.constants.FTYP);  
 let free = MP41.box(MP41.types.free);  
 // allocate mdatbox  
 let mdatbox = new Uint8Array(*mdatBytes* + 8);  
 mdatbox[0] = (*mdatBytes* + 8 >>> 24) & 0xFF;  
 mdatbox[1] = (*mdatBytes* + 8 >>> 16) & 0xFF;  
 mdatbox[2] = (*mdatBytes* + 8 >>> 8) & 0xFF;  
 mdatbox[3] = (*mdatBytes* + 8) & 0xFF;  
 mdatbox.set(MP41.types.mdat, 4);  
 let offset = 8;  
 // Write samples into mdatbox  
 for (let i = 0; i < *mdatDataList*.length; i++) {  
 *mdatDataList*[i].chunkOffset = ftyp.byteLength + free.byteLength + offset;  
 let units = [], unitLen = *mdatDataList*[i].units.length;  
 for (let j = 0; j < unitLen; j ++) {  
 units[j] = Object.assign({}, *mdatDataList*[i].units[j]);  
 }  
 while (units.length) {  
 let unit = units.shift();  
 let data = unit.data;  
 mdatbox.set(data, offset);  
 offset += data.byteLength;  
 }  
 }  
 let moov = MP41.moov(*meta*, *mdatDataList*);  
 let result = new Uint8Array(ftyp.byteLength + moov.byteLength + mdatbox.byteLength + free.byteLength);  
 result.set(ftyp, 0);  
 result.set(free, ftyp.byteLength);  
 result.set(mdatbox, ftyp.byteLength + free.byteLength);  
 result.set(moov, ftyp.byteLength + mdatbox.byteLength + free.byteLength);  
 return result;  
}

通过以上方法便可编写出mp4文件的blob数据了，接下来说明怎么讲blob数据存储为mp4文件，这里关键点为html5 a标签的一个download属性(ie不支持)和window的内置事件（[event.initMouseEvent](https://developer.mozilla.org/zh-CN/docs/Web/API/MouseEvent/initMouseEvent)）：

\_finishRecord(*recordMate*) {  
 let blob = new Blob([*recordMate*.recordBuffer], {'type': 'application/octet-stream'});  
 let url = window.URL.createObjectURL(blob);  
 let aLink = window.document.createElement('a');  
 aLink.download = *recordMate*.*filename*;  
 aLink.href = url;  
 //创建内置事件并触发  
 let evt = window.document.createEvent('MouseEvents');  
 evt.initMouseEvent('click', true, true, window, 0, 0, 0, 0, 0, false, false, false, false, 0, null);  
 aLink.dispatchEvent(evt);  
}

以上，整个MP4文件就完成了。

关于MP4.moov方法，是根据以上MP4文件格式拼接起来的，若需要详细了解，可看下方的moov方法，因flv视频流暂无音频数据流，编写该封装方法时仅对视频数据进行了编码，音频部分待有音频数据流时开始。



### 存在问题：

1. 目前仅支持视频编码

## 扩展：

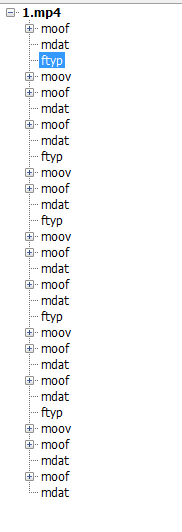
### 流式MP4

流式Mp4文件又称fmp4文件（fragment MP4），与普通MP4文件相比，fmp4文件有以下特点：

1. 内容与metadata分开保存
2. Track之间相互独立
3. Video与audio可以被单独请求
4. 视频质量可不断变化
5. Tracks可多种语言
6. 无需文件全部加载完成便可进行传输

流式Mp4文件中每一个fragment都是一个完整的MP4数据，ftyp box与Moov box绑定，描述数据的类型、兼容协议以及视频参数。在视频参数发生变更时，会再次出现ftyp box和moov box。mdat box用来存储视频碎片数据，moof用来描述mdat，在fmp4中，mdat box与moof绑定存在。

流式MP4文件格式如下：



# 附录：

MP4文件格式资料：<http://www.52rd.com/Blog/wqyuwss/559/>

MP4结构分析工具（Mp4Reader）：

<http://jchblog.u.qiniudn.com/software/MP4Reader_v0.9.0.6.zip>