```
/**
 * @fileoverview MP4结构解析, 仅用于测试学习使用
 *
 * @author chyingp
 */
const fs = require('fs');

function getBox(buffer, offset = 0) {
        let size = buffer.readUInt32BE(offset); // 4个字节
        let type = buffer.slice(offset + 4, offset + 8).toString(); // 4个字节

        if (size === 1) {
                size = buffer.readUIntBE(offset + 8, 8); // 8个字节, largeSize
        } else if (size === 0) {
                // last box
        }

        let boxBuffer = buffer.slice(offset, offset + size);

        return {
        s   i   z   e,
        t   y   p   e,
                buffer: boxBuffer
        };
}

function getInnerBoxes(buffer) {
        let boxes = [];
        let offset = 0;
        let totalByteLen = buffer.byteLength;

        do {
                // let size = buffer.readUInt32BE(offset); // 4个字节
                // let type = buffer.slice(offset + 4, offset + 8).toString(); // 4个字节

                // if (size === 1) {
                //  size = buffer.readUIntBE(offset + 8, 8); // 8个字节, largeSize
                // } else if (size === 0) {
                //  // last box
                // }

                // let boxBuffer = buffer.slice(offset, offset + size);

                // boxes.push({
```

```
                //  type: type,
                //  size: size,
                //  buffer: boxBuffer
                // });

                let box = getBox(buffer, offset);
                boxes.push(box);

                offset += box.size;

                // console.log(`size: ${size}, type: ${type}, offset: ${offset}, totalByteLen: ${totalByteLen}`);

        } while(offset < totalByteLen);

        // console.log(boxes);

        return boxes;
}

/*
 aligned(8) class Box (unsigned int(32) boxtype, optional unsigned int(8)[16] extended_type) {
   unsigned int(32) size;
   unsigned int(32) type = boxtype;
    if (size==1) {
     unsigned int(64) largesize;
    } else if (size==0) {
     // box extends to end of file
                }
    if (boxtype==‘uuid’) {
     unsigned int(8)[16] usertype = extended_type;
                }
        }
*/
class Box {
        constructor(boxType, extendedType, buffer) {
                this.type = boxType; // 必选, 字符串, 4个字节, box类型
                this.size = 0; // 必选, 整数, 4个字节, box的大小, 单位是字节
                this.headerSize = 8; //
                this.boxes = [];

                // this.largeSize = 0; // 可选, 8个字节
                // this.extendedType = extendedType || boxType; // 可选, 16个字节
                this._initialize(buffer);
        }

        _initialize(buffer) {
                this.size = buffer.readUInt32BE(0); // 4个字节
                this.type = buffer.slice(4, 8).toString(); // 4个字节

                let offset = 8;

                if (this.size === 1) {
                        this.size = buffer.readUIntBE(8, 8); // 8个字节, largeSize
                        this.headerSize += 8;
                        offset = 16;
                } else if (this.size === 1) {
                        // last box
                }

                if (this.type === 'uuid') {
                        this.type = buffer.slice(offset, 16); // 16个字节
                        this.headerSize += 16;
                }
        }

        setInnerBoxes(buffer, offset = 0) {
                const innerBoxes = getInnerBoxes(buffer.slice(this.headerSize + offset, this.size));

                innerBoxes.forEach(item => {
                        let { type, buffer } = item;

                        type = type.trim(); // 备注, 有些box类型不一定四个字母, 比如 url、urn

                        if (this[type]) {
                                const box = this[type](buffer);
                                this.boxes.push(box);
                        } else {
                                this.boxes.push('TODO 待实现');
                                // console.log(`unknowed type: ${type}`);
                        }
                });
        }

        // _setInnerBoxes(buffer, offset = 0) {
        //  const innerBoxes = getInnerBoxes(buffer.slice(this.headerSize + offset, this.size));

        //  const innerBoxes = getInnerBoxes(buffer);
        //   const boxes = [];

        //  innerBoxes.forEach(item => {
        //    let { type, buffer } = item;

        //    type = type.trim(); // 备注, 有些box类型不一定四个字母, 比如 url、urn

        //    if (this[type]) {
        //     const box = this[type](buffer);
        //         this.boxes.push(box);
        //     } else {
        //     // console.log(`unknowed type: ${type}`);
        //         }
        //   });
        // }
}

/*
 aligned(8) class FullBox(unsigned int(32) boxtype, unsigned int(8) v, bit(24) f) extends Box(boxtype) {
   unsigned int(8) version = v;
    bit(24) flags = f;
        }
*/
class FullBox extends Box {
        constructor(boxType, buffer) {
                super(boxType, '', buffer);

                const headerSize = this.headerSize;

                this.version = buffer.readUInt8(headerSize); // 必选, 1个字节
                this.flags = buffer.readUIntBE(headerSize + 1, 3); // 必选, 3个字节

                this.headerSize = headerSize + 4;
        }
}

class Movie {
        constructor(buffer) {

                this.boxes = [];
                this.bytesConsumed = 0;

                const innerBoxes = getInnerBoxes(buffer);

                innerBoxes.forEach(item => {
                        const { type, buffer, size } = item;
                        if (this[type]) {
                                const box = this[type](buffer);
                                this.boxes.push(box);
```

```
                        } else {
                                // console.log(`unknowed type: ${type}`);
                                this.boxes.push('TODO 待实现');
                        }
                        this.bytesConsumed += size;
                });
        }

        ftyp(buffer) {
                return new FileTypeBox(buffer);
        }

        pdin() {
                return 'TODO pdin';
        }

        moov(buffer) {
                return new MovieBox(buffer);
        }

        mdat(buffer) {
                return new MediaDataBox(buffer);
        }
}

/*
 aligned(8) class FileTypeBox extends Box(‘ftyp’) {
  unsigned int(32) major_brand;
  unsigned int(32) minor_version;
  unsigned int(32) compatible_brands[];
        }
*/
class FileTypeBox extends Box {
        constructor(buffer) {
                super('ftyp', '', buffer);

                const headerSize = this.headerSize;

                this.majorBrand = buffer.slice(headerSize, headerSize + 4).toString(); // 必选, 字符串, 4个字节
                this.minorVersion = buffer.readUInt32BE(headerSize + 4); // 必选, 整数, 4个字节
                this.compatibleBrands = []; // 必选, 数组, 每个元素4个字节, 填充至box末尾

                for(let i = headerSize + 8; i < this.size; i = i + 4) {
                        const compatibleBrand = buffer.slice(i, i + 4).toString();
                        this.compatibleBrands.push(compatibleBrand);
                }
                // console.log(this);
        }
}

/*
 aligned(8) class MediaDataBox extends Box(‘mdat’) {
   bit(8)  data[];
        }
*/
class MediaDataBox extends Box {
        constructor(buffer) {
                super('mdat', '', buffer);
                this.data = buffer.slice(this.headerSize);
        }
}


/*
 aligned(8) class MovieBox extends Box(‘moov’){ }
*/
class MovieBox extends Box {
        constructor(buffer) {
                super('moov', '', buffer);
                this.setInnerBoxes(buffer);
        }

        mvhd(buffer) {
                return new MovieHeaderBox(buffer);
        }

        trak(buffer) {
                return new TrackBox(buffer);
        }
}

/*

 aligned(8) class MovieHeaderBox extends FullBox(‘mvhd’, version, 0) {
   if (version==1) {
    unsigned int(64)  creation_time;
    unsigned int(64)  modification_time;
    unsigned int(32)  timescale;
    unsigned int(64)  duration;
   } else { // version==0
    unsigned int(32)  creation_time;
    unsigned int(32)  modification_time;
    unsigned int(32)  timescale;
    unsigned int(32)  duration;
                }
  template int(32) rate = 0x00010000; // typically 1.0
  template int(16) volume = 0x0100; // typically, full volume const bit(16) reserved = 0;
  const unsigned int(32)[2] reserved = 0;
  template int(32)[9] matrix = { 0x00010000,0,0,0,0x00010000,0,0,0,0x40000000 }; // Unity matrix
   bit(32)[6]  pre_defined = 0;
  unsigned int(32)  next_track_ID;
        }
*/
class MovieHeaderBox extends FullBox {
        constructor(buffer) {
                super('mvhd', buffer);

                const headerSize = this.headerSize;
                let offset = 0;

                if (this.version === 1) {
                        this.creation_time = buffer.readUIntBE(headerSize, 8);
                        this.modification_time = buffer.readUIntBE(headerSize + 8, 8);
                        this.timescale = buffer.readUInt32BE(headerSize + 16, 4);
                        this.duration = buffer.readUIntBE(headerSize + 20, 8);
                        offset = headerSize + 28;
                } else {
                        this.creation_time = buffer.readUInt32BE(headerSize);
                        this.modification_time = buffer.readUInt32BE(headerSize + 4);
                        this.timescale = buffer.readUInt32BE(headerSize + 8);
                        this.duration = buffer.readUInt32BE(headerSize + 12);
                        offset = headerSize + 16;
                }

                this.rate = buffer.readUInt16BE(offset) + buffer.readUInt16BE(offset + 2)/10; // 4个字节, 按照 16.16 来解析, 默认 0x00010000
                this.volume = buffer.readUInt8(offset + 4) + buffer.readUInt8(offset + 5); // 2个字节, 按照 8.8 来解析, 默认0x0100

                // 接下来6个字节是保留用途
                // const bit(16) reserved = 0;
                // const unsigned int(32)[2] reserved = 0;

                this.matrix = [
                        // buffer.readUInt32BE(offset + 12),
                        buffer.readUInt32BE(offset + 16),
                        buffer.readUInt32BE(offset + 20),
```

```
                        buffer.readUInt32BE(offset + 24),
                        buffer.readUInt32BE(offset + 28),
                        buffer.readUInt32BE(offset + 32),
                        buffer.readUInt32BE(offset + 36),
                        buffer.readUInt32BE(offset + 40),
                        buffer.readUInt32BE(offset + 44),
                        buffer.readUInt32BE(offset + 48),
                ];

                const preDefinedBytes = 32 * 6 / 8;
                this.pre_defined = buffer.slice(offset + 52, offset + 52 + preDefinedBytes); //
                this.next_track_ID = buffer.readUInt32BE(offset + 52 + preDefinedBytes);
        }
}


/*
 aligned(8) class TrackBox extends Box('trak') { }
*/
class TrackBox extends Box {
        constructor(buffer) {
                super('trak', '', buffer);

                this.setInnerBoxes(buffer);
                // console.log(this.boxes);
        }

        tkhd(buffer) {
                return new TrackHeaderBox(buffer);
        }

        edts(buffer) {
                return 'TODO edts';
        }

        mdia(buffer) {
                return new MediaBox(buffer);
        }
}

/*
 aligned(8) class TrackHeaderBox extends FullBox('tkhd', version, flags){
   if  (version==1)  {
    unsigned int(64)   creation_time;
    unsigned int(64)   modification_time;
    unsigned  int(32)   track_ID;
    const unsigned int(32)   reserved = 0;
    unsigned  int(64)   duration;
     } else { // version==0
    unsigned int(32)   creation_time;
    unsigned int(32)   modification_time;
    unsigned  int(32)   track_ID;
    const unsigned int(32)   reserved = 0;
    unsigned  int(32)   duration;
                }
  const unsigned int(32)[2] reserved = 0;
   template int(16) layer = 0;
   template int(16) alternate_group = 0;
  template int(16) volume = {if track_is_audio 0x0100 else 0}; const unsigned int(16) reserved = 0;
  template int(32)[9] matrix= { 0x00010000,0,0,0,0x00010000,0,0,0,0x40000000 }; // unity matrix
      unsigned int(32) width;
      unsigned int(32) height;
       }
*/
class TrackHeaderBox extends FullBox {
       constructor(buffer) {
                super('tkhd', buffer);

                const headerSize = this.headerSize;
                let offset = 0;

                if (this.version === 1) {
                        this.creation_time = buffer.readUIntBE(headerSize, 8); // 8个字节
                        this.modification_time = buffer.readUIntBE(headerSize + 8, 8); // 8个字节
                        this.track_ID = buffer.readUInt32BE(headerSize + 16);
                        // const unsigned int(32)  reserved = 0; // 预留4个字节
                        this.duration = buffer.readUIntBE(headerSize + 24, 8); // 4个字节
                        offset = headerSize + 32;
                } else {
                        this.creation_time = buffer.readUInt32BE(headerSize); // 4个字节
                        this.modification_time = buffer.readUInt32BE(headerSize + 4); // 4个字节
                        this.track_ID = buffer.readUInt32BE(headerSize + 8);
                        // const unsigned int(32)  reserved = 0; // 预留4个字节
                        this.duration = buffer.readUInt32BE(headerSize + 16); // 4个字节
                        offset = headerSize + 20;
                }

                // 请勿删, 接下来8个字节是保留用途
                // const unsigned int(32)[2] reserved = 0; // 8个字节

                this.layer = buffer.readUInt16BE(offset + 8); // 2个字节
                this.alternate_group = buffer.readUInt16BE(offset + 10); // 2个字节
                this.volume = 'TODO'; // TODO 2个字节, 根据是视频轨道, 还是音频轨道, 赋不同的值(需要先处理 hdlr, 然后再回来处理这个值)

                // 请勿删, 接下来2个字节是保留用途
                // const unsigned int(16) reserved = 0;

                offset += 16; // 加上

                // 36个字节:9个元素, 每个元素4个字节
                this.matrix = [
                        buffer.readInt32BE(offset),
                        buffer.readInt32BE(offset + 4),
                        buffer.readInt32BE(offset + 8),
                        buffer.readInt32BE(offset + 12),
                        buffer.readInt32BE(offset + 16),
                        buffer.readInt32BE(offset + 20),
                        buffer.readInt32BE(offset + 24),
                        buffer.readInt32BE(offset + 28),
                        buffer.readInt32BE(offset + 32)
                ];
                this.width = buffer.readUInt16BE(offset + 36); // 4个字节, 格式为 16.16
                this.height = buffer.readUInt16BE(offset + 40); // 4个字节, 格式为 16.16
        }
}

/*
 aligned(8) class MediaBox extends Box('mdia') { }
*/
class MediaBox extends Box {
        constructor(buffer) {
                super('mdia', '', buffer);
                this._handler_type = '';

                // TODO 需要确保 hdlr 比 minf 先解析
                this.setInnerBoxes(buffer);
        }

        mdhd(buffer) {
                return new MediaHeaderBox(buffer);
        }

        hdlr(buffer) {
                let hdlr = new HandlerBox(buffer);
                this._handler_type = hdlr.handler_type;

```

```
                return hdlr;
        }

        minf(buffer) {
                return new MediaInformationBox(buffer, this._handler_type);
        }
}


/*
 aligned(8) class MediaHeaderBox extends FullBox('mdhd', version, 0) {
   if (version==1) {
        unsigned int(64)  creation_time;
        unsigned int(64)  modification_time;
        unsigned int(32)  timescale;
        unsigned int(64)  duration;
   } else { // version==0
     unsigned int(32)  creation_time;
     unsigned int(32)  modification_time;
     unsigned int(32)  timescale;
     unsigned int(32)  duration;
                }
    bit(1)  pad=0;
   unsigned int(5)[3] language; // ISO-639-2/T language code unsigned
    int(16) pre_defined = 0;
        }
*/
class MediaHeaderBox extends FullBox {
        constructor(buffer) {
                super('mdhd', buffer);

                const headerSize = this.headerSize;
                let offset = 0;

                if (this.version === 1) {
                        this.creation_time = buffer.readUIntBE(headerSize, 8); // 8个字节, 单位是秒
                        this.modification_time = buffer.readUIntBE(headerSize + 8, 8); // 8个字节, 单位是秒
                        this.timescale = buffer.readUInt32BE(headerSize + 16); // 4个字节, 每秒包含的时间单位(time units), 比如1000
                        this.duration = buffer.readUIntBE(headerSize + 20, 8); // 8个字节, duration/timescale 得到实际的时长(秒)
                        offset = headerSize + 28;
                } else {
                        this.creation_time = buffer.readUInt32BE(headerSize); // 8个字节
                        this.modification_time = buffer.readUInt32BE(headerSize + 4); // 8个字节
                        this.timescale = buffer.readUInt32BE(headerSize + 8);
                        this.duration = buffer.readUInt32BE(headerSize + 12); // 4个字节
                        offset = headerSize + 16;
                }

                this.pad = 0; // 1 bit

                const codeDifferences = [ // 15 bits, 3个元素, 每个元素 5 bits. ISO-639-2/T language code unsigned
                        buffer.readUInt8(offset) >> 2,
                        buffer.readUInt16BE(offset) >> 5 & 0b0000000000011111,
                        buffer.readUInt8(offset + 1) & 0b00011111,
                ];

                // Each character is packed as the difference between its ASCII value and 0x60.
                // 举例, codeDifferences 为 [21, 14, 4]. 则 language 为 ['u', 'n', 'd']
                this.language = codeDifferences.map(code => {
                        return String.fromCharCode(code + 0x60);
                });

                // pad + language 共2个字节
                this.pre_defined = buffer.readUInt16BE(offset + 2); // 2个字节, 预留

                // console.log(this);
        }
}

/*
 aligned(8) class HandlerBox extends FullBox('hdlr', version = 0, 0) {
   unsigned int(32) pre_defined = 0;
   unsigned int(32) handler_type;
   const unsigned int(32)[3] reserved = 0;
        string name;
        }

    //  视频轨道
  HandlerBox {
    type:  'hdlr',
     size:  54,
    headerSize:  12,
     boxes:  [],
     version:  0,
     flags:  0,
    pre_defined:  0,
    handler_type:  'vide',
   name: 'L-SMASH Video Handler\u0000' }

    //  音频轨道
  HandlerBox {
    type:  'hdlr',
     size:  54,
    headerSize:  12,
     boxes:  [],
     version:  0,
     flags:  0,
    pre_defined:  0,
    handler_type:  'soun',
   name: 'L-SMASH Audio Handler\u0000' }
*/
class HandlerBox extends FullBox {
        constructor(buffer) {
                super('hdlr', buffer);

                let offset = this.headerSize;

                this.pre_defined = buffer.readUInt32BE(offset); // 4个字节, 预留

                // vide(video track), soun(audio track), hint(hint track)
                // 4个字节, 表明是 video track、audio track 还是 hint track
                this.handler_type = buffer.slice(offset + 4, offset + 8).toString();

                // 12个字节, 预留
                // const unsigned int(32)[3] reserved = 0;

                this.name = buffer.slice(offset + 20).toString();

                // console.log(this);
        }
}

/*
 aligned(8) class MediaInformationBox extends Box('minf') { }

 video media header, overall information (video track only)
*/
class MediaInformationBox extends Box {
        constructor(buffer, handler_type) {
                super('minf', '', buffer);
                this._handler_type = handler_type;
                this.setInnerBoxes(buffer);
        }

        vmhd(buffer) {
                return new VideoMediaHeaderBox(buffer);
```

```
        }

        smhd(buffer) {
                return new SoundMediaHeaderBox(buffer);
        }

        hmhd(buffer) {
                return 'TODO hmhd';
        }

        dinf(buffer) {
                return new DataInformationBox(buffer);
        }

        stbl(buffer) {
                return new SampleTableBox(buffer, this._handler_type);
        }
}

/*
 aligned(8) class VideoMediaHeaderBox extends FullBox('vmhd', version = 0, 1) {
  template unsigned int(16) graphicsmode = 0; // copy, see below template
  unsigned int(16)[3] opcolor = {0, 0, 0};
        }

  /  /   例子
 VideoMediaHeaderBox {
      type: 'vmhd',
       size: 20,
      headerSize: 12,
       boxes: [],
       version: 0,
       flags: 1,
      graphicsmode: 0,
     opcolor: [ 0, 0, 0 ] },
*/
class VideoMediaHeaderBox extends FullBox {
        constructor(buffer) {
                super('vmhd', buffer);

                this.version = 0;
                this.flags = 1;

                const offset = this.headerSize;

                this.graphicsmode = buffer.readUInt16BE(offset); // 2个字节, 是枚举的值, 目前只有0, 表示直接对图像进行拷贝(相当于不做处理)

                this.opcolor = [ // 6个字节, TODO 这个字段干嘛的?
                        buffer.readUInt16BE(offset + 2),
                        buffer.readUInt16BE(offset + 4),
                        buffer.readUInt16BE(offset + 6)
                ];
        }
}


/*
 aligned(8) class SoundMediaHeaderBox extends FullBox('smhd', version = 0, 0) {
    template int(16) balance = 0;
    const unsigned int(16)  reserved = 0;
        }

   /  /   例子:
 SoundMediaHeaderBox {
      type: 'smhd',
       size: 16,
      headerSize: 12,
       boxes: [],
       version: 0,
       flags: 0,
      balance: 0 }
*/
class SoundMediaHeaderBox extends FullBox {
        constructor(buffer) {
                super('smhd', buffer);

                this.version = 0;
                this.flags = 0;

                const offset = this.headerSize;

                // is a fixed-point 8.8 number that places mono audio tracks in a stereo space;
                // 0 is center (thenormal value); full left is -1.0 and full right is 1.0.
                this.balance = buffer.readInt8(offset) + buffer.readInt8(offset + 1) / 10; // 2个字节

                // 预留, 2个字节
                // const unsigned int(16)  reserved = 0;
        }
}

/*
 sample table box, container for the time/space map

 aligned(8) class SampleTableBox extends Box('stbl') { }
*/
class SampleTableBox extends Box {
        constructor(buffer, handler_type) {
                super('stbl', '', buffer);
                this._handler_type = handler_type;
                this.setInnerBoxes(buffer);
        }

        stsd(buffer) {
                return new SampleDescriptionBox(buffer, this._handler_type);
        }

        stco(buffer) {
                return new ChunkOffsetBox(buffer);
        }

        stsc(buffer) {
                return new SampleToChunkBox(buffer);
        }

        stsz(buffer) {
                return new SampleSizeBox('stsz', buffer);
        }

        stz2(buffer) {
                return new SampleSizeBox('stz2', buffer);
        }

        stts(buffer) {
                return new TimeToSampleBox(buffer);
        }

        stss(buffer) {
                return new SyncSampleBox(buffer);
        }

        ctts(buffer) {
                return new CompositionOffsetBox(buffer);
        }
}

```

```
/*
 sample descriptions (codec types, initialization etc.)

 aligned(8) abstract class SampleEntry (unsigned int(32) format) extends Box(format){
  const unsigned int(8)[6] reserved = 0;
  unsigned int(16) data_reference_index;
       }

 class HintSampleEntry() extends SampleEntry (protocol) {
  unsigned int(8) data [];
       }

 // Visual Sequences
 class VisualSampleEntry(codingname) extends SampleEntry (codingname){
  unsigned int(16) pre_defined = 0;
  const unsigned int(16) reserved = 0;
  unsigned int(32)[3] pre_defined = 0;
  unsigned int(16) width;
  unsigned int(16) height;
  template unsigned int(32) horizresolution = 0x00480000; // 72 dpi
  template unsigned int(32) vertresolution = 0x00480000; // 72 dpi const
  unsigned int(32) reserved = 0;
  template unsigned int(16) frame_count = 1;
  string[32] compressorname;
  template unsigned int(16) depth = 0x0018;
  int(16) pre_defined = -1;
       }

 // Audio Sequences
 class AudioSampleEntry(codingname) extends SampleEntry (codingname){
  const unsigned int(32)[2] reserved = 0;
  template unsigned int(16) channelcount = 2;
  template unsigned int(16) samplesize = 16;
  unsigned int(16) pre_defined = 0;
  const unsigned int(16) reserved = 0 ;
  template unsigned int(32) samplerate = {timescale of media}<<16;
       }

 aligned(8) class SampleDescriptionBox (unsigned int(32) handler_type) extends FullBox('stsd', 0, 0){
     int    i ;
  unsigned int(32) entry_count;
  for (i = 1 ; i u entry_count ; i++){
    switch (handler_type){
      case ‘soun’: // for audio tracks
                  }
             }
       }

aligned(8) class SampleDescriptionBox (unsigned int(32) handler_type) extends FullBox('stsd', 0, 0){
 int i ;
 unsigned int(32) entry_count;
 for (i = 1 ; i u entry_count ; i++){
  switch (handler_type){
    case ‘soun’: // for audio tracks
       AudioSampleEntry();
             break;
    case ‘vide’: // for video tracks
       VisualSampleEntry();
             break;
    case ‘hint’: // Hint track
       HintSampleEntry();
                 break;
             }
       }
}
*/
class SampleEntry extends Box {
       constructor(format = '', buffer) {
              super(format, '', buffer);

              // 预留 8*6 = 48位 => 6个字节
              // const unsigned int(8)[6] reserved = 0;

              this.data_reference_index = buffer.readUInt16BE(this.headerSize + 6); // 2个字节
       }
}

/*
 AVC decoder configuration record
       参考 mpeg-4 part 15.5.2.4.1节

 aligned(8) class AVCDecoderConfigurationRecord {
  unsigned int(8) configurationVersion = 1;
  unsigned int(8) AVCProfileIndication;
  unsigned int(8) profile_compatibility;
  unsigned int(8) AVCLevelIndication;
  bit(6) reserved = ‘111111’b;
  unsigned int(2) lengthSizeMinusOne;
  bit(3) reserved = ‘111’b;
  unsigned int(5) numOfSequenceParameterSets;

  for (i=0; i< numOfSequenceParameterSets; i++) {
   unsigned int(16) sequenceParameterSetLength ;
   bit(8*sequenceParameterSetLength) sequenceParameterSetNALUnit;
             }

  unsigned int(8) numOfPictureParameterSets;

  for (i=0; i< numOfPictureParameterSets; i++) {
   unsigned int(16) pictureParameterSetLength;
   bit(8*pictureParameterSetLength) pictureParameterSetNALUnit;
             }

  if( profile_idc  ==  100  ||  profile_idc  ==  110  ||
   profile_idc  ==  122  ||  profile_idc  ==  144 )
             {
    bit(6) reserved = ‘111111’b;
    unsigned int(2) chroma_format;
    bit(5) reserved = ‘11111’b;
   unsigned int(3) bit_depth_luma_minus8;
    bit(5) reserved = ‘11111’b;
   unsigned int(3) bit_depth_chroma_minus8;
   unsigned int(8) numOfSequenceParameterSetExt;

   for (i=0; i< numOfSequenceParameterSetExt; i++) {
     unsigned int(16) sequenceParameterSetExtLength;
    bit(8*sequenceParameterSetExtLength) sequenceParameterSetExtNALUnit;
                 }
             }
       }
*/
class AVCDecoderConfigurationRecord{
       constructor(buffer) {
              // super(boxType, '', buffer);

              // const offset = this.headerSize;
              let offset = 0;

              this.configurationVersion = buffer.readUInt8(offset++); // 1个字节

              // AVCProfileIndication contains the profile code as defined in ISO/IEC 14496-10.
              this.AVCProfileIndication = buffer.readUInt8(offset++); // 1个字节

              this.profile_compatibility = buffer.readUInt8(offset++); // 1个字节

              // AVCLevelIndication contains the level code as defined in ISO/IEC 14496-10.
```

```
                this.AVCLevelIndication = buffer.readUInt8(offset++); // 1个字节

                // bit(6) reserved = ‘111111’b; // 高6位，保留
                this.lengthSizeMinusOne = buffer[offset++] & 0b00000011; // 高2位

                // bit(3) reserved = ‘111’b; // 高3位，保留
                this.numOfSequenceParameterSets = buffer[offset++] & 0b00011111; // 低5位
                this.sequenceParameterSets = [];
                for(let i = 0; i< this.numOfSequenceParameterSets; i++) {
                        const sequenceParameterSetLength = buffer.readUInt16BE(offset); // 2个字节
                        offset += 2;
                        const sequenceParameterSetNALUnit = buffer.slice(offset, offset += sequenceParameterSetLength); // sequenceParameterSetLength 个字节
                        this.sequenceParameterSets.push({ sequenceParameterSetLength, sequenceParameterSetNALUnit });
                }

                this.numOfPictureParameterSets = buffer[offset++]; // 1个字节
                this.pictureParameterSets = [];
                for (let i = 0; i < this.numOfPictureParameterSets; i++) {
                        const pictureParameterSetLength = buffer.readUInt16BE(offset); // 2个字节
                        offset += 2;
                        const pictureParameterSetNALUnit = buffer.slice(offset, offset += pictureParameterSetLength); // pictureParameterSetLength 个字节
                        this.pictureParameterSets.push({ pictureParameterSetLength, pictureParameterSetNALUnit });
                }

                const profile_idc = this.AVCProfileIndication;
                if( profile_idc  ==  100  ||  profile_idc  ==  110  ||
                        profile_idc  ==  122  ||  profile_idc  ==  144 )
                {
                        // bit(6) reserved = ‘111111’b; // 高6位，保留
                        this.chroma_format = buffer[offset++] & 0b00000011; // 低2位

                        // bit(5) reserved = ‘11111’b; // 高5位，保留
                        this.bit_depth_luma_minus8 = buffer[offset++] & 0b00000111; // 低3位

                        // bit(5) reserved = ‘11111’b; // 高5位，保留
                        this.bit_depth_chroma_minus8 = buffer[offset++] & 0b00000111; // 3位

                        this.numOfSequenceParameterSetExt = buffer.readUInt8(offset++); // 1个字节
                        this.sequenceParameterSetExt = [];
                        for (let i = 0; i < this.numOfSequenceParameterSetExt; i++) {
                                const sequenceParameterSetExtLength = buffer.readUInt16BE(offset); // 2个字节
                                const sequenceParameterSetExtNALUnit = buffer.slice(offset += 2, offset += sequenceParameterSetExtLength); // sequenceParameterSetExt
                                this.sequenceParameterSetExt.push({ sequenceParameterSetExtLength, sequenceParameterSetExtNALUnit });
                        }
                }
        }
}
/*
 // Visual Sequences
 class AVCConfigurationBox extends Box(‘avcC’) {
  AVCDecoderConfigurationRecord() AVCConfig;
        }

        例子:
 AVCConfigurationBox {
   type:  'avcC',
    size:  58,
   headerSize:  8,
    boxes:  [],
  AVCConfig: AVCDecoderConfigurationRecord {
    configurationVersion:  1,
    AVCProfileIndication:  100,
    profile_compatibility:  0,
    AVCLevelIndication:  31,
    lengthSizeMinusOne:  3,
    numOfSequenceParameterSets:  1,
    sequenceParameterSets:  [ [Object] ],
    numOfPictureParameterSets:  1,
    pictureParameterSets:  [ [Object] ],
     chroma_format:  1,
    bit_depth_luma_minus8:  0,
    bit_depth_chroma_minus8:  0,
    numOfSequenceParameterSetExt:  0,
    sequenceParameterSetExt:  []
                }
        }
*/
class AVCConfigurationBox extends Box {
        constructor(buffer) {
                super('avcC', '', buffer);

                this.AVCConfig = new AVCDecoderConfigurationRecord(buffer.slice(this.headerSize));

                // console.log(this);
        }
}

/*
 Color Parameter Atoms ('colr')
        参考:https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFChap3/qtff3.html#//apple_ref/doc/uid/TP40000939-CH205-125526

 Color parameter type:4字节
 Primaries index:2字节
 Transfer function index:2字节
 Matrix index:2字节

        例子:
 Colr { type: 'colr', size: 19, headerSize: 8, boxes: Array(0), colorParameterType: 'nclx'}
*/

class Colr extends Box {
        constructor(buffer) {
                super('colr', '', buffer);
                const offset = this.headerSize;

                // A 32-bit field containing a four-character code for the color parameter type. The currently defined types are 'nclc' for video, and 'prof' for pri
                this.colorParameterType = buffer.slice(offset, offset + 4).toString().trim(); // 4字节
                // A 16-bit unsigned integer containing an index into a table specifying the CIE 1931 xy chromaticity coordinates of the white point and the red, gre
                this.primariesIndex = buffer.readUInt32BE(offset + 4); // 2字节
                // A 16-bit unsigned integer containing an index into a table specifying the nonlinear transfer function coefficients used to translate between RGB c
                this.transferFunctionIndex = buffer.readUInt16BE(offset + 2); // 2字节
                // A 16-bit unsigned integer containing an index into a table specifying the transformation matrix coefficients used to translate between RGB color s
                this.matrix = buffer.readUInt16BE(offset + 2); // 2字节
        }
}
/*
 This extension specifies the height-to-width ratio of pixels found in the video sample.
 This is a required extension for MPEG-4 and uncompressed Y´CbCr video formats when non-square pixels are used.
 It is optional when square pixels are used.

        参考这里:https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFChap3/qtff3.html#//apple_ref/doc/uid/TP40000939-CH205-124550

  hSpacing:4字节. An unsigned 32-bit integer specifying the horizontal spacing of pixels, such as luma sampling instants for Y´CbCr or YUV video.
  vSpacing:4字节. An unsigned 32-bit integer specifying the vertical spacing of pixels, such as video picture lines

        例子:
 PixelAspectRatio {type: 'pasp', size: 16, headerSize: 8, boxes: Array(0), hSpacing: 1}
*/
class PixelAspectRatio extends Box {
        constructor(buffer) {
                super('pasp', '', buffer);
                let offset = this.headerSize;

                this.hSpacing = buffer.readUInt32BE(offset);
                this.vSpacing = buffer.readUInt32BE(offset + 4);
        }
```

```
}

/*

  The ‘protocol’ and ‘codingname’ fields are registered identifiers that uniquely identify the streaming protocol or compression format decoder to be used.
  A given protocol or codingname may have optional or required extensions to the sample description (e.g. codec initialization parameters).
  All such extensions shall be within boxes; these boxes occur after the required fields.
  Unrecognized boxes shall be ignored.

        上面这段话的意思，比如 'codingname' 是 'AVC'，那么，VisualSampleEntry 里还可以包含其他扩展参数，比如编码初始化参数(codec initialization parameters)
        这些扩展参数是可选的，作为 VisualSampleEntry 内部的box存在。如果这些内部box是不认识的，那么需要直接忽略。
        也就是说，这些内部box是针对特定编码自定义的，需要查看编码相关的规范(MP4规范本身没有定义)

        举例:codingname 为 avc1，内部box可能包含 avcC、colr、pasp 等内部box，其中，avcC 这个box内存储了SPS、PPS信息

  The AVC file format (Advanced Video Coding) is the video file format defined in Part 15 of the MPEG-4 standard. It uses ISO Base Media File Format (MPEG-4 Part
   a v c / a v c 1  在'mpeg-4 part 15'中 定义，采用 AVC 编码，并采用 ISOM 存储，可以认为是 MP4 的扩展。

  Part 15: Carriage of network abstraction layer (NAL) unit structured video in the ISO base media file format
  For storage of Part 10 video. File format is based on Part 12, but also allows storage in other file formats.

        例子:
  VisualSampleEntry {
    t y p e :   ' a v c 1 ' ,
     s i z e :    1 7 9 ,
    h e a d e r S i z e :   8 ,
     b o x e s :    [
      A V C C o n f i g u r a t i o n B o x   {
          t y p e :     ' a v c C ' ,
            s i z e :     5 8 ,
          h e a d e r S i z e :    8 ,
            b o x e s :     [ ] ,
       A V C C o n f i g :  [ A V C D e c o d e r C o n f i g u r a t i o n R e c o r d ]
                }     ,
       ' T O D O    c o l r ' ,
       ' T O D O    p a s p '
          ]     ,
    d a t a _ r e f e r e n c e _ i n d e x :  1 ,
     w i d t h :    9 6 0 ,
     h e i g h t :    5 4 0 ,
    h o r i z r e s o l u t i o n :  4 7 1 8 5 9 2 ,
    v e r t r e s o l u t i o n :  4 7 1 8 5 9 2 ,
     f r a m e _ c o u n t :   1 ,
    c o m p r e s s o r n a m e :  ' A V C   C o d i n g ' ,
     d e p t h :    2 4
        }
*/
class VisualSampleEntry extends SampleEntry {
        constructor(codingname = '', buffer) {
                super(codingname, buffer);

                let offset = this.headerSize + 6 + 2; // SampleEntry 额外占据了 6+2 个字节

                // 预留16个字节
                // unsigned int(16) pre_defined = 0; // 2个字节
                // const unsigned int(16) reserved = 0; // 2个字节
                // unsigned int(32)[3] pre_defined = 0; // 12个字节

                this.width = buffer.readUInt16BE(offset += 16);  // 2个字节
                this.height = buffer.readUInt16BE(offset += 2); // 2个字节

                this.horizresolution = buffer.readUInt32BE(offset += 2) || 0x00480000; // 4个字节，默认值，72dpi
                this.vertresolution = buffer.readUInt32BE(offset += 4) || 0x00480000; // 4个字节，默认值，72dpi

                // unsigned int(32) reserved = 0; // 4个字节

                this.frame_count =  buffer.readUInt16BE(offset += 8) || 1; // 2个字节

                offset += 2;

                const bytesOfCompressorname = buffer.readUInt8(offset); // 1个字节，compressorname 的实际字节数
                this.compressorname =  buffer.slice(offset + 1, offset + 1 + bytesOfCompressorname).toString(); // 32个字节，比如，第1个字节是compressorname实际占据的字节数

                this.depth = buffer.readUInt16BE(offset += 32) || 0x0018; // 2个字节

                // 预留2个字节
                // int(16) pre_defined = -1; // 2个字节

                // 解析内部的box
                this.setInnerBoxes(buffer, offset + 2 + 2 - this.headerSize, 71);
        }

        avcC(buffer) {
                return new AVCConfigurationBox(buffer);
        }

        // 参考:https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFChap3/qtff3.html#//apple_ref/doc/uid/TP40000939-CH205-125526
        colr(buffer) {
                return new Colr(buffer);
        }

        pasp(buffer) {
                return new PixelAspectRatio(buffer);
        }
}


class SampleDescriptionBox extends FullBox {
        constructor(buffer, handler_type) {
                super('stsd', buffer);

                let offset = this.headerSize;

                this.entry_count = buffer.readUInt32BE(offset); // 4个字节，条目数
                offset += 4;

                this.sampleDescriptionEntries = [];

                for (let i = 0; i < this.entry_count; i++) {
                        let box = getBox(buffer, offset); // { size: xx, type: yy, buffer: zz }
                        switch (handler_type) {
                                case 'soun':
                                        // box = getBox(buffer, offset);
                                        // TODO
                                        break;
                                case 'vide':
                                        box = new VisualSampleEntry('', box.buffer);
                                        break;
                                case 'hint':
                                        // TODO
                                        break;
                        }
                        offset += box.size;
                        this.sampleDescriptionEntries.push(box);
                }
        }
}

/*
 aligned(8) class ChunkOffsetBox extends FullBox(‘stco’, version = 0, 0) {
   u n s i g n e d   i n t ( 3 2 )   e n t r y _ c o u n t ;
   f o r   ( i = 1 ;   i   u   e n t r y _ c o u n t ;   i ++ )   {
        u n s i g n e d   i n t ( 3 2 )    c h u n k _ o f f s e t ;
        }
        }
```

```
    /  /   例子
  ChunkOffsetBox {
    t y p e :  ' s t c o ' ,
     s i z e :   5 6 ,
    h e a d e r S i z e :  1 2 ,
     b o x e s :   [ ] ,
     v e r s i o n :   0 ,
     f l a g s :   0 ,
    e n t r y _ c o u n t :  1 0 ,
   chunk_offsets: [ 4286, 192256, 282028, 389838, 488519, 592669, 689195, 841434, 939661, 1047160 ]
     }
*/
class ChunkOffsetBox extends FullBox {
        constructor(buffer) {
                super('stco', buffer);

                this.version = 0;
                this.flags = 0;

                const offset = this.headerSize;

                this.entry_count = buffer.readUInt32BE(offset); // 4个字节, entry条目数

                this.chunk_offsets = [];

                for(let i = 1; i <= this.entry_count; i++) {
                        const chunk_offset = buffer.readUInt32BE(offset + i * 4); // 4个字节, chunk相对于文件的偏移量
                        this.chunk_offsets.push(chunk_offset);
                }

                // console.log(this);
        }
}

/*
    /  /   例子:视轨
  SampleToChunkBox {
    t y p e :  ' s t s c ' ,
     s i z e :   2 8 ,
    h e a d e r S i z e :  1 2 ,
     b o x e s :   [ ] ,
     v e r s i o n :   0 ,
     f l a g s :   0 ,
    e n t r y _ c o u n t :  1 ,
     e n t r i e s :   [
    { first_chunk: 1, samples_per_chunk: 15, sample_description_index: 1 }
                ]
        }

    /  /   例子:音轨
  SampleToChunkBox {
    t y p e :  ' s t s c ' ,
     s i z e :   4 0 ,
    h e a d e r S i z e :  1 2 ,
     b o x e s :   [ ] ,
     v e r s i o n :   0 ,
     f l a g s :   0 ,
    e n t r y _ c o u n t :  2 ,
     e n t r i e s :   [
    { first_chunk: 1,  samples_per_chunk: 24, sample_description_index: 1  },
     { first_chunk: 10, samples_per_chunk: 21, sample_description_index: 1 }
                 ]
        }

 aligned(8) class SampleToChunkBox extends FullBox('stsc', version = 0, 0) {
   unsigned int(32) entry_count;
   for (i=1; i u entry_count; i++) {
     unsigned int(32) first_chunk;
     unsigned int(32) samples_per_chunk;
     unsigned int(32) sample_description_index;
                }
    }
*/
class SampleToChunkBox extends FullBox {
        constructor(buffer) {
                super('stsc', buffer);

                this.version = 0;
                this.flags = 0;

                const offset = this.headerSize;

                this.entry_count = buffer.readUInt32BE(offset); // 4个字节, entry条目数

                this.entries = [];

                for(let i = 0; i < this.entry_count; i++) {
                        const first_chunk = buffer.readUInt32BE(offset + 4 + 12 * i); // 4个字节, 具有相同sample数的第一个chunk的序号, 从1开始
                        const samples_per_chunk = buffer.readUInt32BE(offset + 4 +  12 * i + 4); // 4个字节, 每个chunk里的sample数
                        const sample_description_index = buffer.readUInt32BE(offset + 4 + 12 * i + 8); // 4个字节, 条目的序号
                        this.entries.push({ first_chunk, samples_per_chunk, sample_description_index});
                }

                // console.log(this);
        }
}

/*

    /  /   例子:video track
  SampleSizeBox {
    t y p e :  ' s t s z ' ,
     s i z e :   6 2 0 ,
    h e a d e r S i z e :  1 2 ,
     b o x e s :   [ ] ,
     v e r s i o n :   0 ,
     f l a g s :   0 ,
    s a m p l e _ s i z e :  0 ,
    s a m p l e _ c o u n t :  1 5 0 ,
   entry_sizes: [ 58070,21324,6598,4720,4316,19998,3844,1749,1232,1615, ... ]
        }

    /  /   例子:sound track
  SampleSizeBox {
    t y p e :  ' s t s z ' ,
     s i z e :   9 6 8 ,
    h e a d e r S i z e :  1 2 ,
     b o x e s :   [ ] ,
     v e r s i o n :   0 ,
     f l a g s :   0 ,
    s a m p l e _ s i z e :  0 ,
    s a m p l e _ c o u n t :  2 3 7 ,
   entry_sizes: [ 683,682,683,683,682,683,683,682, ... ]
        }

 aligned(8) class SampleSizeBox extends FullBox('stsz', version = 0, 0) {
   unsigned int(32) sample_size;
   unsigned int(32) sample_count;
   if (sample_size==0) {
     for (i=1; i u sample_count; i++) {
         unsigned int(32)   entry_size;
                }
        }
      }
```

```
 aligned(8) class CompactSampleSizeBox extends FullBox(‘stz2’, version = 0, 0) {
   unsigned int(24) reserved = 0;
   unisgned int(8) field_size;
   unsigned int(32) sample_count;
   for (i=1; i u sample_count; i++) {
    unsigned int(field_size) entry_size;
                }
        }
*/
class SampleSizeBox extends FullBox {
        constructor(boxType = 'stsz', buffer) {
                super(boxType, buffer);

                this.version = 0;
                this.flags = 0;

                let offset = this.headerSize;

                this.sample_size = buffer.readUInt32BE(offset); // 4个字节, 每个sample的size(如果不为0, 则所有sample的size相等)
                this.sample_count = buffer.readUInt32BE(offset + 4); // 4个字节, sample的数目

                this.entry_sizes = [];

                offset += 8;

                if (this.sample_size === 0) {
                        for (let i = 0; i < this.sample_count; i++) {
                                const entry_size = buffer.readUInt32BE(offset + i * 4); // 4个字节, sample的size
                                this.entry_sizes.push(entry_size);
                        }
                }

                // console.log(this);
        }
}

/*

   /  /   例子:video track
 TimeToSampleBox {
   type:  'stts',
    size:  24,
   headerSize:  12,
    boxes:  [],
    version:  0,
    flags:  0,
   entry_count:  1,
    entries:  [
    { sample_count: 150, sample_delta: 1001 }
                ]
        }

   /  /   例子:audio track
 TimeToSampleBox {
   type:  'stts',
    size:  24,
   headerSize:  12,
    boxes:  [],
    version:  0,
    flags:  0,
   entry_count:  1,
    entries:  [
    { sample_count: 237, sample_delta: 1024 }
                ]
        }

 aligned(8) class TimeToSampleBox extends FullBox(‘stts’, version = 0, 0) {
   unsigned int(32)  entry_count;
        int i;
   for (i=0; i < entry_count; i++) {
    unsigned int(32)  sample_count;
    unsigned int(32)  sample_delta;
          }
        }
*/
class TimeToSampleBox extends FullBox {
        constructor(buffer) {
                super('stts', buffer);

                this.version = 0;
                this.flags = 0;

                let offset = this.headerSize;

                this.entry_count = buffer.readUInt32BE(offset); // 4个字节, entry条目数
                this.entries = [
                        // { sample_count: 1, sample_delta: 30 }
                ];

                offset += 4;

                for (let i = 0; i < this.entry_count; i++) {
                        const sample_count = buffer.readUInt32BE(offset + i * 8);
                        const sample_delta = buffer.readUInt32BE(offset + i * 8 + 4);
                        this.entries.push({ sample_count, sample_delta });
                }

                // console.log(this);
        }
}


/*

//
        例子:video track
 SyncSampleBox {
   type:  'stss',
    size:  24,
   headerSize:  12,
    boxes:  [],
    version:  0,
    flags:  0,
   entry_count:  2,
   sample_numbers: [ 1, 91 ]
        }

 aligned(8) class SyncSampleBox extends FullBox(‘stss’, version = 0, 0) {
   unsigned int(32)  entry_count;
     int    i;
   for (i=0; i < entry_count; i++) {
    unsigned int(32)  sample_number;
                }
        }
*/
class SyncSampleBox extends FullBox {
        constructor(buffer) {
                super('stss', buffer);

                this.version = 0;
                this.flags = 0;

                let offset = this.headerSize;

                this.entry_count = buffer.readUInt32BE(offset); // 4个字节, entry条目数
                this.sample_numbers = [];
```

```
                offset += 4;

                for (let i = 0; i < this.entry_count; i++) {
                        const sample_number = buffer.readUInt32BE(offset + i * 4); // 4个字节, sample序号, 从1开始
                        this.sample_numbers.push(sample_number);
                }

                // console.log(this);
        }
}

/*
    /  /   例子:video track
 CompositionOffsetBox {
    type:  'ctts',
    size:  1128,
    headerSize:  12,
    boxes:  [],
    version:  0,
    flags:  0,
    entry_count:  139,
    entries:  [
    { sample_count: 1, sample_offset: 2002 },
    { sample_count: 1, sample_offset: 5005 },
    { sample_count: 1, sample_offset: 2002 },
    { sample_count: 1, sample_offset: 0 },
    { sample_count: 1, sample_offset: 1001 },
                .   .   .
                ]

 aligned(8) class CompositionOffsetBox extends FullBox('ctts', version = 0, 0) {
   unsigned int(32) entry_count;
     int    i ;
   for (i=0; i < entry_count; i++) {
     unsigned int(32)   sample_count;
     unsigned int(32)   sample_offset;
                }
        }
*/
class CompositionOffsetBox extends FullBox {
        constructor(buffer) {
                super('ctts', buffer);

                this.version = 0;
                this.flags = 0;

                let offset = this.headerSize;

                this.entry_count = buffer.readUInt32BE(offset); // 4个字节, entry条目数
                this.entries = [];

                offset += 4;

                for (let i = 0; i < this.entry_count; i++) {
                        const sample_count = buffer.readUInt32BE(offset + i * 8); // 4个字节, 连续有多少个sample产生了偏移(dts,pts 之间)
                        const sample_offset = buffer.readUInt32BE(offset + i * 8 + 4); // 4个字节, 偏移量
                        this.entries.push({ sample_count, sample_offset });
                }
                // console.log(this);
        }
}

/*
 The data information box contains objects that declare the location of the media information in a track.

 aligned(8) class DataInformationBox extends Box('dinf') { }
*/
class DataInformationBox extends Box {
        constructor(buffer) {
                super('dinf', '', buffer);
                this.setInnerBoxes(buffer);
        }

        dref(buffer) {
                return new DataReferenceBox(buffer);
        }
}

/*
 aligned(8) class DataEntryUrlBox (bit(24) flags) extends FullBox('url ', version = 0, flags) {
   string  location;
        }

 aligned(8) class DataEntryUrnBox (bit(24) flags) extends FullBox('urn ', version = 0, flags) {
    string  name;
   string  location;
        }
 aligned(8) class DataReferenceBox extends FullBox('dref', version = 0, 0) {
   unsigned int(32) entry_count;
   for (i=1; i < entry_count; i++) {
    DataEntryBox(entry_version, entry_flags) data_entry;
                }
        }
*/
class DataEntryUrlBox extends FullBox {
        constructor(buffer) {
                super('url', buffer);
                if (this.flags !== 1) { // flag 为 1, 表示媒体数据包含在了当前movie文件里
                        this.location = '';
                } else {
                        this.location = buffer.slice(this.headerSize).toString();
                }
                // console.log(this.flags);
        }
}

class DataEntryUrnBox extends FullBox {
        constructor(buffer) {
                super('urn', buffer);
                if (this.flags !== 1) { // flag 为 1, 表示媒体数据包含在了当前movie文件里
                        this.name = '';
                        this.location = '';
                } else {
                        const nullIndex = buffer.slice(this.headerSize).indexOf(0x00);
                        this.name = buffer.slice(this.headerSize, nullIndex);
                        this.location = buffer.slice(nullIndex + 1);
                }
                // console.log(this.flags);
        }
}

class DataReferenceBox extends FullBox {
        constructor(buffer) {
                super('dref', buffer);

                let offset = this.headerSize;

                this.entry_count = buffer.readUInt32BE(offset); // 4个字节, entry条目数

                this.setInnerBoxes(buffer, 4);

                // console.log(this);
        }

```

```
        url(buffer) {
                return new DataEntryUrlBox(buffer);
        }

        urn(buffer) {
                return new DataEntryUrnBox(buffer);
        }
}

////////////////////////////// 上面是是MP4结构 //////////////////////////////

const filepath = './flower.mp4';
const BYTES_READ_PER_TIME = 256 * 1024; // 每次读取的字节数, 256kb
const OPEN_FLAGS = 'r';

const fd = fs.openSync(filepath, OPEN_FLAGS);
const buff = Buffer.alloc(BYTES_READ_PER_TIME);

let unconsumedBytes = null;
let bytesNum = 0;

const stats = fs.statSync(filepath);
const fileSize = stats.size; // 文件大小, 单位是字节
const mp4 = { boxes: [] };

function read(done) {
        fs.read(fd, buff, 0, BYTES_READ_PER_TIME, null, function (err, bytesRead, buffer) {

                bytesNum += bytesRead;

                // const movie = new Movie( buffer.slice(0, bytesRead) );

                let bytesToParse = bytesRead < BYTES_READ_PER_TIME ? buffer.slice(0, bytesRead) : buffer;

                if (unconsumedBytes) {
                        bytesToParse = Buffer.concat([unconsumedBytes, bytesToParse]);
                }

                let { boxes, bytesConsumed } = parseMovie(bytesToParse); // { boxes: [], bytesConsumed: 0 }

                if (boxes.length !== 0) {
                        mp4.boxes.push(...boxes);
                }

                if (bytesConsumed < bytesRead) {
                        unconsumedBuffer = bytesToParse.slice(bytesConsumed);
                }

                if (bytesNum < fileSize) { // 还没读完
                        read(done);
                } else { // 已读完
                        // console.log('done. \n');
                        // console.log(mp4.boxes);
                        done(mp4);
                }
        });
}

function parseMovie(buffer) {
        let movie = new Movie(buffer);
        return movie;
}

function describeMovie(movie, parentBoxType = '') {
        // const boxes = movie.boxes;
        // boxes.forEach(box => {
        //  console.log(`<< ${parentBoxType}.${box.type} >>`);
        //  console.log(box);
        //  if (box.boxes) {
        //   describeMovie(box.boxes, [parentBoxType, box.type].join('.'));
        // /     }
        // });

        /*
                获取视频 时长、宽、高
                获取视频、音频 编码
                获取视频关键帧
                获取视频帧率
     *   /

        // 获取视频时长
        const mvhd = findBoxes(movie, 'moov.mvhd');
        let duration = 0; // 时长, 单位为秒
        if (mvhd.length === 1) {
                duration = mvhd[0].duration / mvhd[0].timescale;
        }

        // 获取视频宽、高
        const tkhdBoxes = findBoxes(movie, 'moov.trak.tkhd');
        const tkhdOfVideo = tkhdBoxes.length >= 1 ? tkhdBoxes.find(box => box.width > 0) : null;
        let width = 0;
        let height = 0;
        if (tkhdOfVideo) {
                width = tkhdOfVideo.width;
                height = tkhdOfVideo.height;
        }

        console.log(`视频时长=${duration}s, 宽=${width}, 高=${height}`);
}

/**
 * 返回特定层级的box, 比如 moov.trak.tkhd, 需要注意, 返回的可能不止一个box
 * 比如 moov.trak, 包含了 video trak、audio trak
 *
 * @param {Object} box 举例:{ boxes: [{type: 'ftyp'}, {type: 'moov'}]}
 * @param {String} chanin 举例:'moov.mvhd'
 *
 * @returns {Array} 对应的boxes, 每个元素的类型为 Box|FullBox
 */
function findBoxes(outterBox, chanin) {
        const types = chanin.split('.');
        let outterBoxes = [outterBox];

        for(let i = 0; i < types.length; i++) {
                const type = types[i];
                // 比如, moov.trak, 匹配中了2个trak, 因此, 需要将2个trak的boxes合并
                const boxes = outterBoxes.reduce((boxes, curOutterBox) => {
                        return [...boxes, ...(curOutterBox.boxes || [])];
                }, []);

                outterBoxes = boxes.filter(box => box.type === type);

                if (outterBoxes.length === 0) {
                        break;
                }
        }

        return outterBoxes;
}

function run() {
        read(describeMovie);
}

run();
```