



Search



[Sign in](#)

[Sign up](#)

master

blog / demo / 2020.08.24-mp4-analyze / mp4.js

Go to file

...

chyingp get width and height

Latest commit c97a17e on 30 Aug 2020 History

1 contributor

1671 lines (1381 sloc)45.3 KB

RawBlame

```
1  /**
2   * @fileoverview MP4结构解析. 仅用于测试学习使用
3   *
4   * @author chyingp
5   */
6  const fs = require('fs');
7
8  function getBox(buffer, offset = 0) {
9      let size = buffer.readInt32BE(offset); // 4个字节
10     let type = buffer.slice(offset + 4, offset + 8).toString(); // 4个字节
11
12     if (size === 1) {
13         size = buffer.readUIntBE(offset + 8, 8); // 8个字节, largeSize
14     } else if (size === 0) {
15         // last box
16     }
17
18     let boxBuffer = buffer.slice(offset, offset + size);
19
20     return {
21         size,
22         type,
23         buffer: boxBuffer
24     };
25 }
26
27 function getInnerBoxes(buffer) {
28     let boxes = [];
29     let offset = 0;
30     let totalByteLen = buffer.byteLength;
31
32     do {
33         // let size = buffer.readInt32BE(offset); // 4个字节
34         // let type = buffer.slice(offset + 4, offset + 8).toString(); // 4个字节
35
36         // if (size === 1) {
37         //     size = buffer.readUIntBE(offset + 8, 8); // 8个字节, largeSize
38         // } else if (size === 0) {
39         //     // last box
40         // }
41
42         // let boxBuffer = buffer.slice(offset, offset + size);
43
44         // boxes.push({
```

```

45         // type: type,
46         // size: size,
47         // buffer: boxBuffer
48         // });
49
50         let box = getBox(buffer, offset);
51         boxes.push(box);
52
53         offset += box.size;
54
55         // console.log('size: ${size}, type: ${type}, offset: ${offset}, totalBytesLen: ${totalBytesLen}');
56
57     } while(offset < totalBytesLen);
58
59     // console.log(boxes);
60
61     return boxes;
62 }
63
64 /*
65 aligned(8) class Box (unsigned int(32) boxtype, optional unsigned int(8)[16] extended_type) {
66     unsigned int(32) size;
67     unsigned int(32) type = boxtype;
68     if (size == 1) {
69         unsigned int(64) largeSize;
70     } else if (size == 0) {
71         // box extends to end of file
72     }
73     if (boxtype == 'uuid') {
74         unsigned int(8)[16] usertype = extended_type;
75     }
76 }
77 */
78 class Box {
79     constructor(boxType, extendedType, buffer) {
80         this.type = boxType; // 必选. 字符串. 4个字节. box类型
81         this.size = 0; // 必选. 整数. 4个字节. box的大小. 单位是字节
82         this.headerSize = 8; //
83         this.boxes = [];
84
85         // this.largeSize = 0; // 可选. 8个字节
86         // this.extendedType = extendedType || boxType; // 可选. 16个字节
87         this._initialize(buffer);
88     }
89
90     _initialize(buffer) {
91         this.size = buffer.readUInt32BE(0); // 4个字节
92         this.type = buffer.slice(4, 8).toString(); // 4个字节
93
94         let offset = 8;
95
96         if (this.size === 1) {
97             this.size = buffer.readUIntBE(8, 8); // 8个字节. largeSize
98             this.headerSize += 8;
99             offset = 16;
100         } else if (this.size === 0) {
101             // last box
102         }
103
104         if (this.type === 'uuid') {
105             this.type = buffer.slice(offset, 16); // 16个字节
106             this.headerSize += 16;
107         }
108     }
109
110     setInnerBoxes(buffer, offset = 0) {
111         const innerBoxes = getInnerBoxes(buffer.slice(this.headerSize + offset, this.size));
112
113         innerBoxes.forEach(item => {
114             let { type, buffer } = item;
115
116             type = type.trim(); // 备注. 有些box类型不一定四个字节. 比如 url, urn
117
118             if (this[type]) {
119                 const box = this[type](buffer);
120                 this.boxes.push(box);
121             } else {
122                 this.boxes.push('TODO 待实现');
123                 // console.log('unknown type: ${type}');
124             }
125         });
126     }
127
128     // _setInnerBoxes(buffer, offset = 0) {
129     //     const innerBoxes = getInnerBoxes(buffer.slice(this.headerSize + offset, this.size));
130
131     //     const innerBoxes = getInnerBoxes(buffer);
132     //     const boxes = [];
133
134     //     innerBoxes.forEach(item => {
135     //         let { type, buffer } = item;
136
137     //         type = type.trim(); // 备注. 有些box类型不一定四个字节. 比如 url, urn
138
139     //         if (this[type]) {
140     //             const box = this[type](buffer);
141     //             this.boxes.push(box);
142     //         } else {
143     //             // console.log('unknown type: ${type}');
144     //         }
145     //     });
146     // }
147 }
148
149 /*
150 aligned(8) class FullBox(unsigned int(32) boxtype, unsigned int(8) v, bit(24) f) extends Box(boxtype) {
151     unsigned int(8) version = v;
152     bit(24) flags = f;
153 }
154 */
155 class FullBox extends Box {
156     constructor(boxType, buffer) {
157         super(boxType, '', buffer);
158
159         const headerSize = this.headerSize;
160
161         this.version = buffer.readUInt8(headerSize); // 必选. 1个字节
162         this.flags = buffer.readUIntBE(headerSize + 1, 3); // 必选. 3个字节
163
164         this.headerSize = headerSize + 4;
165     }
166 }
167
168 class Movie {
169     constructor(buffer) {
170
171         this.boxes = [];
172         this.bytesConsumed = 0;
173
174         const innerBoxes = getInnerBoxes(buffer);
175
176         innerBoxes.forEach(item => {
177             const { type, buffer, size } = item;
178             if (this[type]) {
179                 const box = this[type](buffer);
180                 this.boxes.push(box);

```

```

181         } else {
182             // console.log('unknown type: ${type}');
183             this.boxes.push('TODO 待实现');
184         }
185         this.bytesConsumed += size;
186     });
187 }
188
189 ftyp(buffer) {
190     return new FileTypeBox(buffer);
191 }
192
193 pdin() {
194     return 'TODO pdin';
195 }
196
197 moov(buffer) {
198     return new MovieBox(buffer);
199 }
200
201 mdet(buffer) {
202     return new MediaDataBox(buffer);
203 }
204 }
205
206 /*
207 aligned(8) class FileTypeBox extends Box('ftyp') {
208     unsigned int(32) major_brand;
209     unsigned int(32) minor_version;
210     unsigned int(32) compatible_brands[];
211 }
212 */
213 class FileTypeBox extends Box {
214     constructor(buffer) {
215         super('ftyp', '', buffer);
216
217         const headerSize = this.headerSize;
218
219         this.majorBrand = buffer.slice(headerSize, headerSize + 4).toString(); // 必选. 字符串. 4个字节
220         this.minorVersion = buffer.readInt32BE(headerSize + 4); // 必选. 整数. 4个字节
221         this.compatibleBrands = []; // 必选. 数组. 每个元素4个字节. 填充至box末尾
222
223         for(let i = headerSize + 8; i < this.size; i = i + 4) {
224             const compatibleBrand = buffer.slice(i, i + 4).toString();
225             this.compatibleBrands.push(compatibleBrand);
226         }
227         // console.log(this);
228     }
229 }
230
231 /*
232 aligned(8) class MediaDataBox extends Box('mdat') {
233     bit(8) data[];
234 }
235 */
236 class MediaDataBox extends Box {
237     constructor(buffer) {
238         super('mdat', '', buffer);
239         this.data = buffer.slice(this.headerSize);
240     }
241 }
242
243
244 /*
245 aligned(8) class MovieBox extends Box('moov'){ }
246 */
247 class MovieBox extends Box {
248     constructor(buffer) {
249         super('moov', '', buffer);
250         this.setInnerBoxes(buffer);
251     }
252
253     mvhd(buffer) {
254         return new MovieHeaderBox(buffer);
255     }
256
257     trak(buffer) {
258         return new TrackBox(buffer);
259     }
260 }
261
262 /*
263
264 aligned(8) class MovieHeaderBox extends FullBox('mvhd', version, 0) {
265     if (version == 1) {
266         unsigned int(64) creation_time;
267         unsigned int(64) modification_time;
268         unsigned int(32) timescale;
269         unsigned int(64) duration;
270     } else { // version == 0
271         unsigned int(32) creation_time;
272         unsigned int(32) modification_time;
273         unsigned int(32) timescale;
274         unsigned int(32) duration;
275     }
276     template int(32) rate = 0x00010000; // typically 1.0
277     template int(16) volume = 0x0100; // typically, full volume const bit(16) reserved = 0;
278     const unsigned int(32)[2] reserved = 0;
279     template int(32)[9] matrix = { 0x00010000, 0, 0, 0, 0x00010000, 0, 0, 0, 0x40000000 }; // Unity matrix
280     bit(32)[6] pre_defined = 0;
281     unsigned int(32) next_track_ID;
282 }
283 */
284 class MovieHeaderBox extends FullBox {
285     constructor(buffer) {
286         super('mvhd', buffer);
287
288         const headerSize = this.headerSize;
289         let offset = 0;
290
291         if (this.version == 1) {
292             this.creation_time = buffer.readUIntBE(headerSize, 8);
293             this.modification_time = buffer.readUIntBE(headerSize + 8, 8);
294             this.timescale = buffer.readUInt32BE(headerSize + 16, 4);
295             this.duration = buffer.readUIntBE(headerSize + 20, 8);
296             offset = headerSize + 28;
297         } else {
298             this.creation_time = buffer.readUInt32BE(headerSize);
299             this.modification_time = buffer.readUInt32BE(headerSize + 4);
300             this.timescale = buffer.readUInt32BE(headerSize + 8);
301             this.duration = buffer.readUInt32BE(headerSize + 12);
302             offset = headerSize + 16;
303         }
304
305         this.rate = buffer.readUInt16BE(offset) + buffer.readUInt16BE(offset + 2) / 10; // 4个字节. 按照 16.16 来解析. 默认 0x00010000
306         this.volume = buffer.readUInt8(offset + 4) + buffer.readUInt8(offset + 5); // 2个字节. 按照 8.8 来解析. 默认 0x0100
307
308         // 接下来6个字节是保留用途
309         // const bit(16) reserved = 0;
310         // const unsigned int(32)[2] reserved = 0;
311
312         this.matrix = [
313             // buffer.readUInt32BE(offset + 12),
314             buffer.readUInt32BE(offset + 16),
315             buffer.readUInt32BE(offset + 20),

```

```

316         buffer.readInt32BE(offset + 24),
317         buffer.readInt32BE(offset + 28),
318         buffer.readInt32BE(offset + 32),
319         buffer.readInt32BE(offset + 36),
320         buffer.readInt32BE(offset + 40),
321         buffer.readInt32BE(offset + 44),
322         buffer.readInt32BE(offset + 48),
323     ];
324
325     const preDefinedBytes = 32 * 6 / 8;
326     this.pre_defined = buffer.slice(offset + 52, offset + 52 + preDefinedBytes); //
327     this.next_track_ID = buffer.readInt32BE(offset + 52 + preDefinedBytes);
328 }
329 }
330
331
332 /*
333  aligned(8) class TrackBox extends Box('trak') { }
334 */
335 class TrackBox extends Box {
336     constructor(buffer) {
337         super('trak', '', buffer);
338
339         this.setInnerBoxes(buffer);
340         // console.log(this.boxes);
341     }
342
343     tkhd(buffer) {
344         return new TrackHeaderBox(buffer);
345     }
346
347     edts(buffer) {
348         return 'TODO edts';
349     }
350
351     mdia(buffer) {
352         return new MediaBox(buffer);
353     }
354 }
355
356 /*
357  aligned(8) class TrackHeaderBox extends FullBox('tkhd', version, flags){
358  if (version == 1) {
359      unsigned int(64) creation_time;
360      unsigned int(64) modification_time;
361      unsigned int(32) track_ID;
362      const unsigned int(32) reserved = 0;
363      unsigned int(64) duration;
364  } else { // version == 0
365      unsigned int(32) creation_time;
366      unsigned int(32) modification_time;
367      unsigned int(32) track_ID;
368      const unsigned int(32) reserved = 0;
369      unsigned int(32) duration;
370  }
371  const unsigned int(32)[2] reserved = 0;
372  template int(16) layer = 0;
373  template int(16) alternate_group = 0;
374  template int(16) volume = (if track_is_audio 0x0100 else 0); const unsigned int(16) reserved = 0;
375  template int(32)[9] matrix= { 0x00010000,0,0,0,0x00010000,0,0,0,0x40000000 }; // unity matrix
376      unsigned int(32) width;
377      unsigned int(32) height;
378  }
379 */
380 class TrackHeaderBox extends FullBox {
381     constructor(buffer) {
382         super('tkhd', buffer);
383
384         const headerSize = this.headerSize;
385         let offset = 0;
386
387         if (this.version === 1) {
388             this.creation_time = buffer.readUIntBE(headerSize, 8); // 8个字节
389             this.modification_time = buffer.readUIntBE(headerSize + 8, 8); // 8个字节
390             this.track_ID = buffer.readInt32BE(headerSize + 16);
391             // const unsigned int(32) reserved = 0; // 预留4个字节
392             this.duration = buffer.readUIntBE(headerSize + 24, 8); // 4个字节
393             offset = headerSize + 32;
394         } else {
395             this.creation_time = buffer.readInt32BE(headerSize); // 4个字节
396             this.modification_time = buffer.readInt32BE(headerSize + 4); // 4个字节
397             this.track_ID = buffer.readInt32BE(headerSize + 8);
398             // const unsigned int(32) reserved = 0; // 预留4个字节
399             this.duration = buffer.readInt32BE(headerSize + 16); // 4个字节
400             offset = headerSize + 20;
401         }
402
403         // 请勿删, 接下来8个字节是保留用途
404         // const unsigned int(32)[2] reserved = 0; // 8个字节
405
406         this.layer = buffer.readUInt16BE(offset + 8); // 2个字节
407         this.alternate_group = buffer.readUInt16BE(offset + 10); // 2个字节
408         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)
409
410         // 请勿删, 接下来2个字节是保留用途
411         // const unsigned int(16) reserved = 0;
412
413         offset += 16; // 加上
414
415         // 36个字节:9个元素, 每个元素4个字节
416         this.matrix = [
417             buffer.readInt32BE(offset),
418             buffer.readInt32BE(offset + 4),
419             buffer.readInt32BE(offset + 8),
420             buffer.readInt32BE(offset + 12),
421             buffer.readInt32BE(offset + 16),
422             buffer.readInt32BE(offset + 20),
423             buffer.readInt32BE(offset + 24),
424             buffer.readInt32BE(offset + 28),
425             buffer.readInt32BE(offset + 32)
426         ];
427         this.width = buffer.readUInt16BE(offset + 36); // 4个字节, 格式为 16.16
428         this.height = buffer.readUInt16BE(offset + 40); // 4个字节, 格式为 16.16
429     }
430 }
431
432 /*
433  aligned(8) class MediaBox extends Box('mdia') { }
434 */
435 class MediaBox extends Box {
436     constructor(buffer) {
437         super('mdia', '', buffer);
438         this._handler_type = '';
439
440         // TODO 需要确保 hdlr 比 minf 先解析
441         this.setInnerBoxes(buffer);
442     }
443
444     mdhd(buffer) {
445         return new MediaHeaderBox(buffer);
446     }
447
448     hdlr(buffer) {
449         let hdlr = new HandlerBox(buffer);
450         this._handler_type = hdlr.handler_type;
451     }

```



```

452     }
453     return hdlr;
454 }
455 minf(buffer) {
456     return new MediaInformationBox(buffer, this_handler_type);
457 }
458 }
459
460
461 /*
462 aligned(8) class MediaHeaderBox extends FullBox('mdhd', version, 0) {
463     if (version == 1) {
464         unsigned int(64) creation_time;
465         unsigned int(64) modification_time;
466         unsigned int(32) timescale;
467         unsigned int(64) duration;
468     } else { // version == 0
469         unsigned int(32) creation_time;
470         unsigned int(32) modification_time;
471         unsigned int(32) timescale;
472         unsigned int(32) duration;
473     }
474     bit(1) pad = 0;
475     unsigned int(5)[3] language; // ISO-639-2/T language code unsigned
476     int(16) pre_defined = 0;
477 }
478 */
479 class MediaHeaderBox extends FullBox {
480     constructor(buffer) {
481         super('mdhd', buffer);
482
483         const headerSize = this.headerSize;
484         let offset = 0;
485
486         if (this.version == 1) {
487             this.creation_time = buffer.readUIntBE(headerSize, 8); // 8个字节, 单位是秒
488             this.modification_time = buffer.readUIntBE(headerSize + 8, 8); // 8个字节, 单位是秒
489             this.timescale = buffer.readUInt32BE(headerSize + 16); // 4个字节, 每秒包含的时间单位(time units), 比如1000
490             this.duration = buffer.readUIntBE(headerSize + 20, 8); // 8个字节, duration/timescale 得到实际的时长(秒)
491             offset = headerSize + 28;
492         } else {
493             this.creation_time = buffer.readUInt32BE(headerSize); // 8个字节
494             this.modification_time = buffer.readUInt32BE(headerSize + 4); // 8个字节
495             this.timescale = buffer.readUInt32BE(headerSize + 8);
496             this.duration = buffer.readUInt32BE(headerSize + 12); // 4个字节
497             offset = headerSize + 16;
498         }
499
500         this.pad = 0; // 1 bit
501
502         const codeDifferences = [ // 15 bits, 3个元素, 每个元素 5 bits, ISO-639-2/T language code unsigned
503             buffer.readInt8(offset) >> 2,
504             buffer.readInt16BE(offset) >> 5 & 0b00000000000011111,
505             buffer.readInt8(offset + 1) & 0b00011111,
506         ];
507
508         // Each character is packed as the difference between its ASCII value and 0x60.
509         // 举例, codeDifferences 为 [21, 14, 4], 则 language 为 ['u', 'n', 'd']
510         this.language = codeDifferences.map(code => {
511             return String.fromCharCode(code + 0x60);
512         });
513
514         // pad + language 共2个字节
515         this.pre_defined = buffer.readUInt16BE(offset + 2); // 2个字节, 预留
516
517         // console.log(this);
518     }
519 }
520
521 /*
522 aligned(8) class HandlerBox extends FullBox('hdlr', version = 0, 0) {
523     unsigned int(32) pre_defined = 0;
524     unsigned int(32) handler_type;
525     const unsigned int(32)[3] reserved = 0;
526     string name;
527 }
528
529 // 视频轨道
530 HandlerBox {
531     type: 'hdlr',
532     size: 54,
533     headerSize: 12,
534     boxes: [],
535     version: 0,
536     flags: 0,
537     pre_defined: 0,
538     handler_type: 'vide',
539     name: 'L-SMASH Video Handler\u0000' }
540
541 // 音频轨道
542 HandlerBox {
543     type: 'hdlr',
544     size: 54,
545     headerSize: 12,
546     boxes: [],
547     version: 0,
548     flags: 0,
549     pre_defined: 0,
550     handler_type: 'soun',
551     name: 'L-SMASH Audio Handler\u0000' }
552 */
553 class HandlerBox extends FullBox {
554     constructor(buffer) {
555         super('hdlr', buffer);
556
557         let offset = this.headerSize;
558
559         this.pre_defined = buffer.readUInt32BE(offset); // 4个字节, 预留
560
561         // vide(video track), soun(audio track), hint(hint track)
562         // 4个字节, 表明是 video track, audio track 还是 hint track
563         this.handler_type = buffer.slice(offset + 4, offset + 8).toString();
564
565         // 12个字节, 预留
566         // const unsigned int(32)[3] reserved = 0;
567
568         this.name = buffer.slice(offset + 20).toString();
569
570         // console.log(this);
571     }
572 }
573
574 /*
575 aligned(8) class MediaInformationBox extends Box('minf') { }
576
577 video media header, overall information (video track only)
578 */
579 class MediaInformationBox extends Box {
580     constructor(buffer, handler_type) {
581         super('minf', '', buffer);
582         this_handler_type = handler_type;
583         this.setInnerBoxes(buffer);
584     }
585
586     vmhd(buffer) {
587         return new VideoMediaHeaderBox(buffer);

```

```

588     }
589
590     smhd(buffer) {
591         return new SoundMediaHeaderBox(buffer);
592     }
593
594     hmhd(buffer) {
595         return 'TODO hmhd';
596     }
597
598     dinf(buffer) {
599         return new DataInformationBox(buffer);
600     }
601
602     stbl(buffer) {
603         return new SampleTableBox(buffer, this._handler_type);
604     }
605 }
606
607 /*
608 aligned(8) class VideoMediaHeaderBox extends FullBox('vmhd', version = 0, 1) {
609     template unsigned int(16) graphicsmode = 0; // copy, see below template
610     unsigned int(16)[3] opcolor = {0, 0, 0};
611 }
612
613 / / 例子
614 VideoMediaHeaderBox {
615     type: 'vmhd',
616     size: 20,
617     headerSize: 12,
618     boxes: [],
619     version: 0,
620     flags: 1,
621     graphicsmode: 0,
622     opcolor: [ 0, 0, 0 ] },
623 */
624 class VideoMediaHeaderBox extends FullBox {
625     constructor(buffer) {
626         super('vmhd', buffer);
627
628         this.version = 0;
629         this.flags = 1;
630
631         const offset = this.headerSize;
632
633         this.graphicsmode = buffer.readUInt16E(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)
634
635         this.opcolor = [ // 6个字节. TODO 这个字段干嘛的?
636             buffer.readUInt16E(offset + 2),
637             buffer.readUInt16E(offset + 4),
638             buffer.readUInt16E(offset + 6)
639         ];
640     }
641 }
642
643
644 /*
645 aligned(8) class SoundMediaHeaderBox extends FullBox('smhd', version = 0, 0) {
646     template int(16) balance = 0;
647     const unsigned int(16) reserved = 0;
648 }
649
650 / / 例子:
651 SoundMediaHeaderBox {
652     type: 'smhd',
653     size: 16,
654     headerSize: 12,
655     boxes: [],
656     version: 0,
657     flags: 0,
658     balance: 0 }
659 */
660 class SoundMediaHeaderBox extends FullBox {
661     constructor(buffer) {
662         super('smhd', buffer);
663
664         this.version = 0;
665         this.flags = 0;
666
667         const offset = this.headerSize;
668
669         // is a fixed-point 8.8 number that places mono audio tracks in a stereo space;
670         // 0 is center (thenormal value); full left is -1.0 and full right is 1.0.
671         this.balance = buffer.readInt8(offset) + buffer.readInt8(offset + 1) / 10; // 2个字节
672
673         // 预留 2个字节
674         // const unsigned int(16) reserved = 0;
675     }
676 }
677
678 /*
679 sample table box, container for the time/space map
680
681 aligned(8) class SampleTableBox extends Box('stbl') { }
682 */
683 class SampleTableBox extends Box {
684     constructor(buffer, handler_type) {
685         super('stbl', '', buffer);
686         this._handler_type = handler_type;
687         this.setInnerBoxes(buffer);
688     }
689
690     stsd(buffer) {
691         return new SampleDescriptionBox(buffer, this._handler_type);
692     }
693
694     stco(buffer) {
695         return new ChunkOffsetBox(buffer);
696     }
697
698     stsc(buffer) {
699         return new SampleToChunkBox(buffer);
700     }
701
702     stsz(buffer) {
703         return new SampleSizeBox('stsz', buffer);
704     }
705
706     stz2(buffer) {
707         return new SampleSizeBox('stz2', buffer);
708     }
709
710     stts(buffer) {
711         return new TimeToSampleBox(buffer);
712     }
713
714     stss(buffer) {
715         return new SyncSampleBox(buffer);
716     }
717
718     ctts(buffer) {
719         return new CompositionOffsetBox(buffer);
720     }
721 }
722

```

```

723  /*
724  sample descriptions (codec types, initialization etc.)
725
726  aligned(8) abstract class SampleEntry (unsigned int(32) format) extends Box(format){
727      const unsigned int(8)[6] reserved = 0;
728      unsigned int(16) data_reference_index;
729  }
730
731  class HintSampleEntry() extends SampleEntry (protocol) {
732      unsigned int(8) data [ ];
733  }
734
735  // Visual Sequences
736  class VisualSampleEntry(codingname) extends SampleEntry (codingname){
737      unsigned int(16) pre_defined = 0;
738      const unsigned int(16) reserved = 0;
739      unsigned int(32)[3] pre_defined = 0;
740      unsigned int(16) width;
741      unsigned int(16) height;
742      template unsigned int(32) horizresolution = 0x00480000; // 72 dpi
743      template unsigned int(32) vertresolution = 0x00480000; // 72 dpi const
744      unsigned int(32) reserved = 0;
745      template unsigned int(16) frame_count = 1;
746      string[32] compressorname;
747      template unsigned int(16) depth = 0x0018;
748      int(16) pre_defined = -1;
749  }
750
751  // Audio Sequences
752  class AudioSampleEntry(codingname) extends SampleEntry (codingname){
753      const unsigned int(32)[2] reserved = 0;
754      template unsigned int(16) channelcount = 2;
755      template unsigned int(16) samplesize = 16;
756      unsigned int(16) pre_defined = 0;
757      const unsigned int(16) reserved = 0 ;
758      template unsigned int(32) samplerate = {timescale of media}<<16;
759  }
760
761  aligned(8) class SampleDescriptionBox (unsigned int(32) handler_type) extends FullBox('std', 0, 0){
762      int i ;
763      unsigned int(32) entry_count;
764      for (i = 1 ; i <= entry_count ; i++){
765          switch ( handler_type ){
766              case 'soun': // for audio tracks
767              }
768          }
769      }
770
771  aligned(8) class SampleDescriptionBox (unsigned int(32) handler_type) extends FullBox('std', 0, 0){
772      int i ;
773      unsigned int(32) entry_count;
774      for (i = 1 ; i <= entry_count ; i++){
775          switch ( handler_type ){
776              case 'soun': // for audio tracks
777                  AudioSampleEntry ( ) ;
778                  break ;
779              case 'vide': // for video tracks
780                  VisualSampleEntry ( ) ;
781                  break ;
782              case 'hint': // Hint track
783                  HintSampleEntry ( ) ;
784                  break ;
785          }
786      }
787  }
788  /*
789  class SampleEntry extends Box {
790      constructor(format = '', buffer) {
791          super(format, '', buffer);
792
793          // 预留 8*6 = 48位 => 6个字节
794          // const unsigned int(8)[6] reserved = 0;
795
796          this.data_reference_index = buffer.readUInt16BE(this.headerSize + 6); // 2个字节
797      }
798  }
799
800  /*
801  AVC decoder configuration record
802  参考 mpeg-4 part 15.5.2.4.1节
803
804  aligned(8) class AVCDeroderConfigurationRecord {
805      unsigned int(8) configurationVersion = 1;
806      unsigned int(8) AVCProfileIndication;
807      unsigned int(8) profile_compatibility;
808      unsigned int(8) AVCLlevelIndication;
809      bit(6) reserved = '11111'b;
810      unsigned int(2) lengthSizeMinusOne;
811      bit(3) reserved = '111'b;
812      unsigned int(5) numOfSequenceParameterSets;
813
814      for (i=0; i< numOfSequenceParameterSets; i++) {
815          unsigned int(16) sequenceParameterSetLength ;
816          bit(8*sequenceParameterSetLength) sequenceParameterSetNALUnit;
817      }
818
819      unsigned int(8) numOfPictureParameterSets;
820
821      for (i=0; i< numOfPictureParameterSets; i++) {
822          unsigned int(16) pictureParameterSetLength;
823          bit(8*pictureParameterSetLength) pictureParameterSetNALUnit;
824      }
825
826      if( profile_idc == 100 || profile_idc == 110 ||
827          profile_idc == 122 || profile_idc == 144 )
828      {
829          bit(6) reserved = '111111'b;
830          unsigned int(2) chroma_format;
831          bit(5) reserved = '11111'b;
832          unsigned int(3) bit_depth_luma_minus8;
833          bit(5) reserved = '11111'b;
834          unsigned int(3) bit_depth_chroma_minus8;
835          unsigned int(8) numOfSequenceParameterSetExt;
836
837          for (i=0; i< numOfSequenceParameterSetExt; i++) {
838              unsigned int(16) sequenceParameterSetExtLength;
839              bit(8*sequenceParameterSetExtLength) sequenceParameterSetExtNALUnit;
840          }
841      }
842  }
843  /*
844  class AVCDeroderConfigurationRecord{
845      constructor(buffer) {
846          // super(boxType, '', buffer);
847
848          // const offset = this.headerSize;
849          let offset = 0;
850
851          this.configurationVersion = buffer.readUInt8(offset++); // 1个字节
852
853          // AVCProfileIndication contains the profile code as defined in ISO/IEC 14496-10.
854          this.AVCProfileIndication = buffer.readUInt8(offset++); // 1个字节
855
856          this.profile_compatibility = buffer.readUInt8(offset++); // 1个字节
857
858          // AVCLlevelIndication contains the level code as defined in ISO/IEC 14496-10.

```

```

859 this.AVCLevelIndication = buffer.readUInt8(offset++); // 1个字节
860
861 // bit(6) reserved = '11111'b; // 高6位. 保留
862 this.lengthSizeMinusOne = buffer[offset++] & 0b00000011; // 高2位
863
864 // bit(3) reserved = '111'b; // 高3位. 保留
865 this.numOfSequenceParameterSets = buffer[offset++] & 0b00011111; // 低5位
866 this.sequenceParameterSets = [];
867 for(let i = 0; i < this.numOfSequenceParameterSets; i++) {
868     const sequenceParameterSetLength = buffer.readUInt16BE(offset); // 2个字节
869     offset += 2;
870     const sequenceParameterSetNALUnit = buffer.slice(offset, offset += sequenceParameterSetLength); // sequenceParameterSetLength 个字节
871     this.sequenceParameterSets.push({ sequenceParameterSetLength, sequenceParameterSetNALUnit });
872 }
873
874 this.numOfPictureParameterSets = buffer[offset++]; // 1个字节
875 this.pictureParameterSets = [];
876 for (let i = 0; i < this.numOfPictureParameterSets; i++) {
877     const pictureParameterSetLength = buffer.readUInt16BE(offset); // 2个字节
878     offset += 2;
879     const pictureParameterSetNALUnit = buffer.slice(offset, offset += pictureParameterSetLength); // pictureParameterSetLength 个字节
880     this.pictureParameterSets.push({ pictureParameterSetLength, pictureParameterSetNALUnit });
881 }
882
883 const profile_idc = this.AVCProfileIndication;
884 if( profile_idc == 100 || profile_idc == 110 ||
885     profile_idc == 122 || profile_idc == 144 )
886 {
887     // bit(6) reserved = '111111'b; // 高6位. 保留
888     this.chroma_format = buffer[offset++] & 0b00000011; // 低2位
889
890     // bit(5) reserved = '11111'b; // 高5位. 保留
891     this.bit_depth_luma_minus8 = buffer[offset++] & 0b00000111; // 低3位
892
893     // bit(5) reserved = '11111'b; // 高5位. 保留
894     this.bit_depth_chroma_minus8 = buffer[offset++] & 0b00000111; // 3位
895
896     this.numOfSequenceParameterSetExt = buffer.readUInt8(offset++); // 1个字节
897     this.sequenceParameterSetExt = [];
898     for (let i = 0; i < this.numOfSequenceParameterSetExt; i++) {
899         const sequenceParameterSetExtLength = buffer.readUInt16BE(offset); // 2个字节
900         const sequenceParameterSetExtNALUnit = buffer.slice(offset + 2, offset += sequenceParameterSetExtLength); // sequenceParameterSetExtLength 个字节
901         this.sequenceParameterSetExt.push({ sequenceParameterSetExtLength, sequenceParameterSetExtNALUnit });
902     }
903 }
904 }
905 }
906 /*
907 // Visual Sequences
908 class AVCConfigurationBox extends Box('avcC') {
909     AVCDecoderConfigurationRecord() AVCConfig;
910 }
911
912 例子:
913 AVCConfigurationBox {
914   type: 'avcC',
915   size: 58,
916   headerSize: 8,
917   boxes: [] ,
918   AVCConfig: AVCDecoderConfigurationRecord {
919     configurationVersion: 1,
920     AVCProfileIndication: 100,
921     profile_compatibility: 0,
922     AVCLevelIndication: 31,
923     lengthSizeMinusOne: 3,
924     numOfSequenceParameterSets: 1,
925     sequenceParameterSets: [ [Object] ],
926     numOfPictureParameterSets: 1,
927     pictureParameterSets: [ [Object] ],
928     chroma_format: 1,
929     bit_depth_luma_minus8: 0,
930     bit_depth_chroma_minus8: 0,
931     numOfSequenceParameterSetExt: 0,
932     sequenceParameterSetExt: [ ]
933   }
934 }
935 */
936 class AVCConfigurationBox extends Box {
937     constructor(buffer) {
938         super('avcC', '', buffer);
939
940         this.AVCConfig = new AVCDecoderConfigurationRecord(buffer.slice(this.headerSize));
941
942         // console.log(this);
943     }
944 }
945
946 /*
947 Color Parameter Atoms ('colr')
948 参考:https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFChap3/qtff3.html#apple_ref/doc/uid/TP40000939-CH205-125526
949
950 Color parameter type:4字节
951 Primaries index:2字节
952 Transfer function index:2字节
953 Matrix index:2字节
954
955 例子:
956 Colr { type: 'colr', size: 19, headerSize: 8, boxes: Array(0), colorParameterType: 'nclx'}
957 */
958
959 class Colr extends Box {
960     constructor(buffer) {
961         super('colr', '', buffer);
962         const offset = this.headerSize;
963
964         // 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
965         this.colorParameterType = buffer.slice(offset, offset + 4).toString().trim(); // 4字节
966         // 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
967         this.primariesIndex = buffer.readUInt32BE(offset + 4); // 2字节
968         // A 16-bit unsigned integer containing an index into a table specifying the nonlinear transfer function coefficients used to translate between RGB c
969         this.transferFunctionIndex = buffer.readUInt16BE(offset + 2); // 2字节
970         // A 16-bit unsigned integer containing an index into a table specifying the transformation matrix coefficients used to translate between RGB color s
971         this.matrix = buffer.readUInt16BE(offset + 2); // 2字节
972     }
973 }
974 /*
975 This extension specifies the height-to-width ratio of pixels found in the video sample.
976 This is a required extension for MPEG-4 and uncompressed Y'CbCr video formats when non-square pixels are used.
977 It is optional when square pixels are used.
978
979 参考这里:https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFChap3/qtff3.html#apple_ref/doc/uid/TP40000939-CH205-124550
980
981 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.
982 vSpacing:4字节. An unsigned 32-bit integer specifying the vertical spacing of pixels, such as video picture lines
983
984 例子:
985 PixelAspectRatio {type: 'pasp', size: 16, headerSize: 8, boxes: Array(0), hSpacing: 1}
986 */
987 class PixelAspectRatio extends Box {
988     constructor(buffer) {
989         super('pasp', '', buffer);
990         let offset = this.headerSize;
991
992         this.hSpacing = buffer.readUInt32BE(offset);
993         this.vSpacing = buffer.readUInt32BE(offset + 4);
994     }
995 }

```

```

995 }
996
997 /*
998
999 The 'protocol' and 'codingname' fields are registered identifiers that uniquely identify the streaming protocol or compression format decoder to be used.
1000 A given protocol or codingname may have optional or required extensions to the sample description (e.g. codec initialization parameters).
1001 All such extensions shall be within boxes; these boxes occur after the required fields.
1002 Unrecognized boxes shall be ignored.
1003
1004     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)
1005     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。
1006     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)
1007
1008     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内存放了SPS, PPS信息
1009
1010 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
1011 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。
1012
1013 Part 15: Carriage of network abstraction layer (NAL) unit structured video in the ISO base media file format
1014 For storage of Part 10 video. File format is based on Part 12, but also allows storage in other file formats.
1015
1016     例子:
1017 VisualSampleEntry {
1018     type : 'avc1',
1019     size : 179,
1020     headerSize : 8,
1021     boxes : [
1022         AVCCConfigurationBox {
1023             type : 'avcC',
1024             size : 58,
1025             headerSize : 8,
1026             boxes : [ ] ,
1027             AVCCConfig: [AVCDecoderConfigurationRecord]
1028         } ,
1029         'TODD colr',
1030         'TODD pasp'
1031     ] ,
1032     data_reference_index : 1,
1033     width : 960,
1034     height : 540,
1035     horizresolution: 4718592,
1036     vertresolution: 4718592,
1037     frame_count : 1,
1038     compressorname: 'AVC Coding',
1039     depth : 24
1040 }
1041 */
1042 class VisualSampleEntry extends SampleEntry {
1043     constructor(codingname = '', buffer) {
1044         super(codingname, buffer);
1045
1046         let offset = this.headerSize + 6 + 2; // SampleEntry 额外占据了 6+2 个字节
1047
1048         // 预留16个字节
1049         // unsigned int(16) pre_defined = 0; // 2个字节
1050         // const unsigned int(16) reserved = 0; // 2个字节
1051         // unsigned int(32)[3] pre_defined = 0; // 12个字节
1052
1053         this.width = buffer.readUInt16BE(offset += 16); // 2个字节
1054         this.height = buffer.readUInt16BE(offset += 2); // 2个字节
1055
1056         this.horizresolution = buffer.readUInt32BE(offset += 2) || 0x00480000; // 4个字节, 默认值, 72dpi
1057         this.vertresolution = buffer.readUInt32BE(offset += 4) || 0x00480000; // 4个字节, 默认值, 72dpi
1058
1059         // unsigned int(32) reserved = 0; // 4个字节
1060
1061         this.frame_count = buffer.readUInt16BE(offset += 8) || 1; // 2个字节
1062
1063         offset += 2;
1064
1065         const bytesOfCompressorname = buffer.readUInt8(offset); // 1个字节, compressorname 的实际字节数
1066         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
1067
1068         this.depth = buffer.readUInt16BE(offset += 32) || 0x0018; // 2个字节
1069
1070         // 预留2个字节
1071         // int(16) pre_defined = -1; // 2个字节
1072
1073         // 解析内部的box
1074         this.setInnerBoxes(buffer, offset + 2 + 2 - this.headerSize, 71);
1075     }
1076
1077     avcC(buffer) {
1078         return new AVCCConfigurationBox(buffer);
1079     }
1080
1081     // 参考:https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFChap3/qtff3.html#//apple_ref/doc/uid/TP40000939-CH205-125526
1082     colr(buffer) {
1083         return new Colr(buffer);
1084     }
1085
1086     pasp(buffer) {
1087         return new PixelAspectRatio(buffer);
1088     }
1089 }
1090
1091 class SampleDescriptionBox extends FullBox {
1092     constructor(buffer, handler_type) {
1093         super('tsd', buffer);
1094
1095         let offset = this.headerSize;
1096
1097         this.entry_count = buffer.readUInt32BE(offset); // 4个字节, 条目数
1098         offset += 4;
1099
1100         this.sampleDescriptionEntries = [];
1101
1102         for (let i = 0; i < this.entry_count; i++) {
1103             let box = getBox(buffer, offset); // { size: xx, type: yy, buffer: zz }
1104             switch (handler_type) {
1105                 case 'soun':
1106                     // box = getBox(buffer, offset);
1107                     // TODO
1108                     break;
1109                 case 'vide':
1110                     box = new VisualSampleEntry('', box.buffer);
1111                     break;
1112                 case 'hint':
1113                     // TODO
1114                     break;
1115             }
1116             offset += box.size;
1117             this.sampleDescriptionEntries.push(box);
1118         }
1119     }
1120 }
1121 }
1122
1123 /*
1124 aligned(8) class ChunkOffsetBox extends FullBox('stco', version = 0, 0) {
1125     unsigned int(32) entry_count;
1126     for (i=1; i <= entry_count; i++) {
1127         unsigned int(32) chunk_offset;
1128     }
1129 }

```

```

1130
1131 // 例子
1132 ChunkOffsetBox {
1133     type: 'stco',
1134     size: 56,
1135     headerSize: 12,
1136     boxes: [],
1137     version: 0,
1138     flags: 0,
1139     entry_count: 10,
1140     chunk_offsets: [ 4286, 192256, 282028, 389838, 488519, 592669, 689195, 841434, 939661, 1047160 ]
1141 }
1142 */
1143 class ChunkOffsetBox extends FullBox {
1144     constructor(buffer) {
1145         super('stco', buffer);
1146
1147         this.version = 0;
1148         this.flags = 0;
1149
1150         const offset = this.headerSize;
1151
1152         this.entry_count = buffer.readUInt32BE(offset); // 4个字节. entry条目数
1153
1154         this.chunk_offsets = [];
1155
1156         for(let i = 1; i <= this.entry_count; i++) {
1157             const chunk_offset = buffer.readUInt32BE(offset + i * 4); // 4个字节. chunk相对于文件的偏移量
1158             this.chunk_offsets.push(chunk_offset);
1159         }
1160
1161         // console.log(this);
1162     }
1163 }
1164
1165 /*
1166 // 例子:视轨
1167 SampleToChunkBox {
1168     type: 'stsc',
1169     size: 28,
1170     headerSize: 12,
1171     boxes: [],
1172     version: 0,
1173     flags: 0,
1174     entry_count: 1,
1175     entries: [
1176     { first_chunk: 1, samples_per_chunk: 15, sample_description_index: 1 }
1177     ]
1178 }
1179
1180 // 例子:音轨
1181 SampleToChunkBox {
1182     type: 'stsc',
1183     size: 40,
1184     headerSize: 12,
1185     boxes: [],
1186     version: 0,
1187     flags: 0,
1188     entry_count: 2,
1189     entries: [
1190     { first_chunk: 1, samples_per_chunk: 24, sample_description_index: 1 },
1191     { first_chunk: 10, samples_per_chunk: 21, sample_description_index: 1 }
1192     ]
1193 }
1194
1195 aligned(8) class SampleToChunkBox extends FullBox('stsc', version = 0, 0) {
1196     unsigned int(32) entry_count;
1197     for (i=1; i <= entry_count; i++) {
1198         unsigned int(32) first_chunk;
1199         unsigned int(32) samples_per_chunk;
1200         unsigned int(32) sample_description_index;
1201     }
1202 }
1203 */
1204 class SampleToChunkBox extends FullBox {
1205     constructor(buffer) {
1206         super('stsc', buffer);
1207
1208         this.version = 0;
1209         this.flags = 0;
1210
1211         const offset = this.headerSize;
1212
1213         this.entry_count = buffer.readUInt32BE(offset); // 4个字节. entry条目数
1214
1215         this.entries = [];
1216
1217         for(let i = 0; i < this.entry_count; i++) {
1218             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
1219             const samples_per_chunk = buffer.readUInt32BE(offset + 4 + 12 * i + 4); // 4个字节. 每个chunk里的sample数
1220             const sample_description_index = buffer.readUInt32BE(offset + 4 + 12 * i + 8); // 4个字节. 条目的序号
1221             this.entries.push({ first_chunk, samples_per_chunk, sample_description_index});
1222         }
1223
1224         // console.log(this);
1225     }
1226 }
1227
1228 /*
1229 // 例子:video track
1230 SampleSizeBox {
1231     type: 'stsz',
1232     size: 620,
1233     headerSize: 12,
1234     boxes: [],
1235     version: 0,
1236     flags: 0,
1237     sample_size: 0,
1238     sample_count: 150,
1239     entry_sizes: [ 58070, 21324, 6598, 4720, 4316, 19998, 3844, 1749, 1232, 1615, ... ]
1240 }
1241
1242 // 例子:sound track
1243 SampleSizeBox {
1244     type: 'stsz',
1245     size: 968,
1246     headerSize: 12,
1247     boxes: [],
1248     version: 0,
1249     flags: 0,
1250     sample_size: 0,
1251     sample_count: 237,
1252     entry_sizes: [ 683, 682, 683, 683, 682, 683, 683, 682, ... ]
1253 }
1254
1255 aligned(8) class SampleSizeBox extends FullBox('stsz', version = 0, 0) {
1256     unsigned int(32) sample_size;
1257     unsigned int(32) sample_count;
1258     if (sample_size==0) {
1259         for (i=1; i <= sample_count; i++) {
1260             unsigned int(32) entry_size;
1261         }
1262     }
1263 }
1264
1265

```

```

1266 aligned(8) class CompactSampleSizeBox extends FullBox('stz2', version = 0, 0) {
1267     unsigned int(24) reserved = 0;
1268     unsigned int(8) field_size;
1269     unsigned int(32) sample_count;
1270     for (i=1; i < sample_count; i++) {
1271         unsigned int(field_size) entry_size;
1272     }
1273 }
1274 */
1275 class SampleSizeBox extends FullBox {
1276     constructor(boxType = 'stsz', buffer) {
1277         super(boxType, buffer);
1278
1279         this.version = 0;
1280         this.flags = 0;
1281
1282         let offset = this.headerSize;
1283
1284         this.sample_size = buffer.readUInt32E(offset); // 4个字节. 每个sample的大小(如果不为0. 则所有sample的大小相等)
1285         this.sample_count = buffer.readUInt32E(offset + 4); // 4个字节. sample的数目
1286
1287         this.entry_sizes = [];
1288
1289         offset += 8;
1290
1291         if (this.sample_size === 0) {
1292             for (let i = 0; i < this.sample_count; i++) {
1293                 const entry_size = buffer.readUInt32E(offset + i * 4); // 4个字节. sample的大小
1294                 this.entry_sizes.push(entry_size);
1295             }
1296         }
1297
1298         // console.log(this);
1299     }
1300 }
1301
1302 /*
1303
1304 / / 例子:video track
1305 TimeToSampleBox {
1306   type: 'stts',
1307   size: 24,
1308   headerSize: 12,
1309   boxes: [],
1310   version: 0,
1311   flags: 0,
1312   entry_count: 1,
1313   entries: [
1314     { sample_count: 150, sample_delta: 1001 }
1315   ]
1316 }
1317
1318 / / 例子:audio track
1319 TimeToSampleBox {
1320   type: 'stts',
1321   size: 24,
1322   headerSize: 12,
1323   boxes: [],
1324   version: 0,
1325   flags: 0,
1326   entry_count: 1,
1327   entries: [
1328     { sample_count: 237, sample_delta: 1024 }
1329   ]
1330 }
1331
1332 aligned(8) class TimeToSampleBox extends FullBox('stts', version = 0, 0) {
1333     unsigned int(32) entry_count;
1334     int i;
1335     for (i=0; i < entry_count; i++) {
1336         unsigned int(32) sample_count;
1337         unsigned int(32) sample_delta;
1338     }
1339 }
1340 */
1341 class TimeToSampleBox extends FullBox {
1342     constructor(buffer) {
1343         super('stts', buffer);
1344
1345         this.version = 0;
1346         this.flags = 0;
1347
1348         let offset = this.headerSize;
1349
1350         this.entry_count = buffer.readUInt32E(offset); // 4个字节. entry条目数
1351         this.entries = [
1352             // { sample_count: 1, sample_delta: 30 }
1353         ];
1354
1355         offset += 4;
1356
1357         for (let i = 0; i < this.entry_count; i++) {
1358             const sample_count = buffer.readUInt32E(offset + i * 8);
1359             const sample_delta = buffer.readUInt32E(offset + i * 8 + 4);
1360             this.entries.push({ sample_count, sample_delta });
1361         }
1362
1363         // console.log(this);
1364     }
1365 }
1366
1367
1368 /*
1369 //
1370 例子:video track
1371 SyncSampleBox {
1372   type: 'stss',
1373   size: 24,
1374   headerSize: 12,
1375   boxes: [],
1376   version: 0,
1377   flags: 0,
1378   entry_count: 2,
1379   sample_numbers: [ 1, 91 ]
1380 }
1381
1382 aligned(8) class SyncSampleBox extends FullBox('stss', version = 0, 0) {
1383     unsigned int(32) entry_count;
1384     int i;
1385     for (i=0; i < entry_count; i++) {
1386         unsigned int(32) sample_number;
1387     }
1388 }
1389
1390 */
1391 class SyncSampleBox extends FullBox {
1392     constructor(buffer) {
1393         super('stss', buffer);
1394
1395         this.version = 0;
1396         this.flags = 0;
1397
1398         let offset = this.headerSize;
1399
1400         this.entry_count = buffer.readUInt32E(offset); // 4个字节. entry条目数
1401         this.sample_numbers = [];

```

```

1402         offset += 4;
1403     }
1404
1405     for (let i = 0; i < this.entry_count; i++) {
1406         const sample_number = buffer.readUInt32BE(offset + i * 4); // 4个字节. sample序号. 从1开始
1407         this.sample_numbers.push(sample_number);
1408     }
1409
1410     // console.log(this);
1411 }
1412 }
1413
1414 /*
1415 / / 例子:video track
1416 CompositionOffsetBox {
1417   type: 'ctts',
1418   size: 1128,
1419   headerSize: 12,
1420   boxes: [],
1421   version: 0,
1422   flags: 0,
1423   entry_count: 139,
1424   entries: [
1425     { sample_count: 1, sample_offset: 2002 },
1426     { sample_count: 1, sample_offset: 5005 },
1427     { sample_count: 1, sample_offset: 2002 },
1428     { sample_count: 1, sample_offset: 0 },
1429     { sample_count: 1, sample_offset: 1001 },
1430     . . .
1431   ]
1432 }
1433 aligned(8) class CompositionOffsetBox extends FullBox('ctts', version = 0, 0) {
1434   unsigned int(32) entry_count;
1435   int i;
1436   for (i=0; i < entry_count; i++) {
1437     unsigned int(32) sample_count;
1438     unsigned int(32) sample_offset;
1439   }
1440 }
1441 */
1442 class CompositionOffsetBox extends FullBox {
1443   constructor(buffer) {
1444     super('ctts', buffer);
1445
1446     this.version = 0;
1447     this.flags = 0;
1448
1449     let offset = this.headerSize;
1450
1451     this.entry_count = buffer.readUInt32BE(offset); // 4个字节. entry条目数
1452     this.entries = [];
1453
1454     offset += 4;
1455
1456     for (let i = 0; i < this.entry_count; i++) {
1457       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)
1458       const sample_offset = buffer.readUInt32BE(offset + i * 8 + 4); // 4个字节. 偏移量
1459       this.entries.push({ sample_count, sample_offset });
1460     }
1461     // console.log(this);
1462   }
1463 }
1464
1465 /*
1466 The data information box contains objects that declare the location of the media information in a track.
1467
1468 aligned(8) class DataInformationBox extends Box('dinf') { }
1469 */
1470 class DataInformationBox extends Box {
1471   constructor(buffer) {
1472     super('dinf', '', buffer);
1473     this.setInnerBoxes(buffer);
1474   }
1475
1476   dref(buffer) {
1477     return new DataReferenceBox(buffer);
1478   }
1479 }
1480
1481 /*
1482 aligned(8) class DataEntryUrlBox (bit(24) flags) extends FullBox('url ', version = 0, flags) {
1483   string location;
1484 }
1485
1486 aligned(8) class DataEntryUrnBox (bit(24) flags) extends FullBox('urn ', version = 0, flags) {
1487   string name;
1488   string location;
1489 }
1490 aligned(8) class DataReferenceBox extends FullBox('dref', version = 0, 0) {
1491   unsigned int(32) entry_count;
1492   for (i=1; i < entry_count; i++) {
1493     DataEntryBox(entry_version, entry_flags) data_entry;
1494   }
1495 }
1496 */
1497 class DataEntryUrlBox extends FullBox {
1498   constructor(buffer) {
1499     super('url', buffer);
1500     if (this.flags !== 1) { // flag 为 1. 表示媒体数据包含在了当前movie文件里
1501       this.location = '';
1502     } else {
1503       this.location = buffer.slice(this.headerSize).toString();
1504     }
1505     // console.log(this.flags);
1506   }
1507 }
1508
1509 class DataEntryUrnBox extends FullBox {
1510   constructor(buffer) {
1511     super('urn', buffer);
1512     if (this.flags !== 1) { // flag 为 1. 表示媒体数据包含在了当前movie文件里
1513       this.name = '';
1514       this.location = '';
1515     } else {
1516       const nullIndex = buffer.slice(this.headerSize).indexOf(0x00);
1517       this.name = buffer.slice(this.headerSize, nullIndex);
1518       this.location = buffer.slice(nullIndex + 1);
1519     }
1520     // console.log(this.flags);
1521   }
1522 }
1523
1524 class DataReferenceBox extends FullBox {
1525   constructor(buffer) {
1526     super('dref', buffer);
1527
1528     let offset = this.headerSize;
1529
1530     this.entry_count = buffer.readUInt32BE(offset); // 4个字节. entry条目数
1531
1532     this.setInnerBoxes(buffer, 4);
1533
1534     // console.log(this);
1535   }
1536 }

```



```

1537         uri(buffer) {
1538             return new DataEntryUriBox(buffer);
1539         }
1540     }
1541     urn(buffer) {
1542         return new DataEntryUrnBox(buffer);
1543     }
1544 }
1545
1546 ////////////////////////////////////////////////// 上面是MP4结构 ///////////////////////////////////
1547
1548 const filepath = './flower.mp4';
1549 const BYTES_READ_PER_TIME = 256 * 1024; // 每次读取的字节数. 256kb
1550 const OPEN_FLAGS = 'r';
1551
1552 const fd = fs.openSync(filepath, OPEN_FLAGS);
1553 const buff = Buffer.alloc(BYTES_READ_PER_TIME);
1554
1555 let unconsumedBytes = null;
1556 let bytesNum = 0;
1557
1558 const stats = fs.statSync(filepath);
1559 const fileSize = stats.size; // 文件大小. 单位是字节
1560 const mp4 = { boxes: [] };
1561
1562 function read(done) {
1563     fs.read(fd, buff, 0, BYTES_READ_PER_TIME, null, function (err, bytesRead, buffer) {
1564
1565         bytesNum += bytesRead;
1566
1567         // const movie = new Movie( buffer.slice(0, bytesRead) );
1568
1569         let bytesToParse = bytesRead < BYTES_READ_PER_TIME ? buffer.slice(0, bytesRead) : buffer;
1570
1571         if (unconsumedBytes) {
1572             bytesToParse = Buffer.concat([unconsumedBytes, bytesToParse]);
1573         }
1574
1575         let { boxes, bytesConsumed } = parseMovie(bytesToParse); // { boxes: [], bytesConsumed: 0 }
1576
1577         if (boxes.length !== 0) {
1578             mp4.boxes.push(...boxes);
1579         }
1580
1581         if (bytesConsumed < bytesRead) {
1582             unconsumedBuffer = bytesToParse.slice(bytesConsumed);
1583         }
1584
1585         if (bytesNum < fileSize) { // 还没读完
1586             read(done);
1587         } else { // 已读完
1588             // console.log('done. \n');
1589             // console.log(mp4.boxes);
1590             done(mp4);
1591         }
1592     });
1593 }
1594
1595 function parseMovie(buffer) {
1596     let movie = new Movie(buffer);
1597     return movie;
1598 }
1599
1600 function describeMovie(movie, parentBoxType = '') {
1601     // const boxes = movie.boxes;
1602     // boxes.forEach(box => {
1603     //     console.log('<< ${parentBoxType}.${box.type} >>');
1604     //     console.log(box);
1605     //     if (box.boxes) {
1606     //         describeMovie(box.boxes, [parentBoxType, box.type].join('.'));
1607     //     }
1608     // });
1609
1610     /*
1611         获取视频 时长、宽、高
1612         获取视频、音频 编码
1613         获取视频关键帧
1614         获取视频帧率
1615     */
1616
1617     // 获取视频时长
1618     const mvhd = findBoxes(movie, 'moov.mvhd');
1619     let duration = 0; // 时长. 单位为秒
1620     if (mvhd.length === 1) {
1621         duration = mvhd[0].duration / mvhd[0].timescale;
1622     }
1623
1624     // 获取视频宽、高
1625     const tkhdBoxes = findBoxes(movie, 'moov.trak.tkhd');
1626     const tkhdOfVideo = tkhdBoxes.length >= 1 ? tkhdBoxes.find(box => box.width > 0) : null;
1627     let width = 0;
1628     let height = 0;
1629     if (tkhdOfVideo) {
1630         width = tkhdOfVideo.width;
1631         height = tkhdOfVideo.height;
1632     }
1633
1634     console.log(`视频时长=${duration}s. 宽=${width}. 高=${height}`);
1635 }
1636
1637 /**
1638  * 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
1639  * 比如 moov.trak. 包含了 video trak.audio trak
1640  */
1641 * @param {Object} box 示例:{ boxes: [{type: 'ftyp'}, {type: 'moov'}]}
1642 * @param {String} chanin 示例:'moov.mvhd'
1643 *
1644 * @returns {Array} 对应的boxes. 每个元素的类型为 Box|FullBox
1645 */
1646 function findBoxes(outterBox, chanin) {
1647     const types = chanin.split('.');
1648     let outterBoxes = [outterBox];
1649
1650     for(let i = 0; i < types.length; i++) {
1651         const type = types[i];
1652         // 比如. moov.trak. 匹配中了2个trak. 因此. 需要把2个trak的boxes合并
1653         const boxes = outterBoxes.reduce((boxes, curOutterBox) => {
1654             return [...boxes, ...(curOutterBox.boxes || [])];
1655         }, []);
1656
1657         outterBoxes = boxes.filter(box => box.type === type);
1658
1659         if (outterBoxes.length === 0) {
1660             break;
1661         }
1662     }
1663
1664     return outterBoxes;
1665 }
1666
1667 function run() {
1668     read(describeMovie);
1669 }
1670
1671 run();

```

