

H5直播系列二 MSE(Media Source Extensions)



合肥黑

关注

2 2018.10.09 15:07:25 字数 8,718 阅读 32,585

参考

w3c media-source

Media Source 系列 - 使用 Media Source Extensions 播放视频

全面进阶 H5 直播

无 Flash 时代, 让直播拥抱 H5(MSE篇)

使用 MediaSource 搭建流式播放器

一、MSE 意义

1.粗识 HTML5 video 标签和MSE媒体源扩展

当前网页上能够搜到的HTML5和MSE相关的内容一抓一大把, 本文的目的是尽量用较短的篇幅, 简述浏览器为何要使用HTML5的MSE扩展。这也是在我最开始接触有关内容时的最大的疑惑。

以往用户在浏览网页内容尤其是视频内容时, 需要使用像Adobe Flash或是微软的Silverlight这样的插件, 播放视频内容即使是电脑小白也知道, 需要媒体播放器的支持, 前面提到的插件就是起到媒体播放器的作用。但是使用插件这样的方式是很便捷且很不安全的, 一些不法分子会在这些插件上动手脚。因此W3C的最新的HTML5标准中, 定义了一系列新的元素来避免使用插件, 其中就包含了<video>标签这一大名鼎鼎的元素。

正是使用了<video>标签, 支持HTML5的浏览器得以实现无插件就原生支持播放媒体内容, 但是对媒体内容的格式有所限制。说到媒体内容, 就自然地需要谈到媒体的封装格式和编码格式, 这里总结一下, 原视频文件通过编码来压缩文件大小, 再通过封装将压缩音频、字幕组合到一个容器内, 具体内容请大家自行查阅。

我们可以把<video>标签看做拥有解封装和解码功能的浏览器自带播放器。随着视频点播、直播等视频业务的发展, 视频通过流媒体传输协议(目前常用的有两种, MPEG-DASH和Apple的HLS)从服务器端分发给客户端, 媒体内容进一步包含在一层传输协议中, 这样<video>就无法识别了。以HLS为例, 将源文件内容分散地封装到了一个个TS文件中。

仅靠<video>标签无法识别这样的TS文件, 那么就引入了MSE拓展来帮助浏览器识别并处理TS文件, 将其变回原来可识别的媒体容器格式, 这样<video>就可以识别并播放原来的文件了。那么支持HTML5的浏览器就相当于内置了一个能够解析流协议的播放器。

比如在hls.js 源码解读【1】中, 介绍的hls.js

hls实际会先通过 ajax(loader 是可以完成自定义的) 请求 m3u8文件, 然后会读取到文件的分片列表, 以及视频的编码格式, 时长等。随后会按照顺序(非 seek)去对分片进行请求, 这些也是通过 ajax 请求二进制的文件, 然后借助 Media Source Extensions 将 buffer 内容进行合流, 然后组成一个可播的媒体资源文件。

2.为什么国内大部分视频厂商不对PC开放HTML5?

视频源存在兼容性问题。原生的 HTML5 <video> 元素在 Windows PC 上仅支持 mp4 (H.264 编码), webm, ogg 等格式视频的播放。而由于历史遗留问题 (HTML5 视频标准最终被广泛支持以前, Flash 在 Web 视频播放方面有着统治地位), 视频网站的视频源和转码设置, 很多都高清源都是适用于 Flash 播放的 FLV 格式, 只有少量低清晰度视频是 mp4 格式, webm 和 ogg 更是听都没听说过。比如优酷只有高清和标清才有 MP4 源, 超清、1080P 等, 基本都是 FLV 和 HLS(M3U8)的视频源 (在 Windows PC 上支持 M3U8 比支持 FLV 更复杂, 我们不做过多赘述)。而腾讯视频, 因为转型 MP4 比较早, 视频源几乎全部都是 MP4 和 HLS, 所以现在可以在 Mac OS X 上率先支持 PC Web 端的 HTML5 播放器 (Safari 下 HLS、Chrome 下 MP4)。

但是 HTML5 是不是就真的没办法播放 FLV 等格式视频了呢? 不是。解决方案是 MSE, Media Source Extensions, 就是说, HTML5 <video> 不仅可以直接播放上面支持的 mp4、m3u8、webm, ogg 格式, 还可以支持由 JS 处理过后的视频流, 这样我们就可以用 JS 把一些不支持的视频流格式, 转化为支持的格式 (如 H.264 的 mp4)。B 站开源的 flv.js 就是这个技术的一个典型实现。B 站的 PC HTML5 播放器, 就是用 MSE 技术, 将 FLV 源用 JS 实时转码成 HTML5 支持的视频流编码格式 (其实就一个文件头的差异 (这里文件头改成容器。感谢评论区谦逊的指教, 是容器的差异, 容器不只是文件头)), 提供给 HTML5 播放器播放。

一些人问我为什么不直接采用 MP4 格式, 并表示对 FLV 格式的厌恶。这个问题一方面是历史遗留问题, 由于视频网站前期完全依赖 Flash 播放而选择 FLV 格式; 另一方面, 如果仔细研究过 FLV/MP4 封装格式, 你会发现 FLV 格式非常简洁, 而 MP4 内部 box 种类繁杂, 结构复杂固实而又有太多冗余数据。FLV 天生具备流式特征适合网络流传输, 而 MP4 这种使用最广泛的存储格式, 设计却并不一定优雅。

3.Media Source Extensions

我们已经可以在 Web 应用程序上无插件地播放视频和音频了。但是, 现有架构过于简单, 只能满足一次播放整个曲目的需要, 无法实现拆分/合并数个缓冲文件。流媒体直到现在还在使用 Flash 进行服务, 以及通过 RTMP 协议进行视频串流的 Flash 媒体服务器。

MSE 使我们我们可以把通常的单个媒体文件的 src 值替换成引用 MediaSource 对象 (一个包含即将播放的媒体文件的准备状态等信息的容器), 以及引用多个 SourceBuffer 对象 (代表多个组成整个串流的不同媒体块) 的元素。MSE 让我们能够根据内容获取的大小和频率, 或是内存占用详情 (例如什么时候缓存被回收), 进行更加精准地控制。它是基于它可扩展的 API 建立自适应比特率流客户端 (例如DASH 或 HLS 的客户端) 的基础。

Download ---> Response.arrayBuffer(适用fetch/xhr等异步获取流媒体数据) ---> SourceBuffer(添加到MediaSource的buffer中) ---> <video/> or <audio/>

二、运行DEMO

参考MDN在线DEMO bufferAll, 将HTML代码及所用的文件frag_bunny.mp4下载到本地, 即可运行

```
1 <html><head>
2   <meta charset="utf-8">
3 </head>
4 <body>
5   <video controls=""></video>
6   <script>
7     var video = document.querySelector('video');
8
9
10    var assetURL = 'frag_bunny.mp4';
```

推荐阅读

流式播放器的实现原理

阅读 160

超全! iOS 面试题汇总(清晰易懂)

阅读 647

OpenGL ES +MediaCodec音视频框架

集录制录

阅读 256

常见的技术问题

阅读 176

50道前端笔面试题及参考答案

阅读 723

```
11 // Need to be specific for Blink regarding codecs
12 // ./mp4info frag_bunny.mp4 | grep Codec
13 var mimeType = 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"';
14
15
16 if ('MediaSource' in window &&
17 MediaSource.isTypeSupported(mimeType)) {
18   var mediaSource = new MediaSource();
19   //console.log(mediaSource.readyState); // closed
20   video.src = URL.createObjectURL(mediaSource);
21   mediaSource.addEventListener('sourceopen', sourceOpen);
22 } else {
23   console.error('Unsupported MIME type or codec: ', mimeType);
24 }
25
26
27 function sourceOpen (e) {
28   //console.log(this.readyState); // open
29   var mediaSource = e.target;
30   var sourceBuffer = mediaSource.addSourceBuffer(mimeType);
31   fetchAB(assetURL, function (buf) {
32     sourceBuffer.addEventListener('updateend', function () {
33       mediaSource.endOfStream();
34       video.play();
35     });
36     //console.log(mediaSource.readyState); // ended
37   });
38   console.log("buf",buf);
39   sourceBuffer.appendBuffer(buf);
40 });
41
42
43
44
45 function fetchAB (url, cb) {
46   console.log(url);
47   var xhr = new XMLHttpRequest();
48   xhr.open('get', url);
49   xhr.responseType = 'arraybuffer';
50   xhr.onload = function () {
51     cb(xhr.response);
52   };
53   xhr.send();
54 }
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
```

```
frag_keyframe+empty_mov_fragmented.mp4
```

或者使用 mp4fragment: `mp4fragment input.mp4 output.mp4`

三、URL.createObjectURL

在[H5直播系列一 Blob File FileReader URL](#)曾经介绍过URL.createObjectURL方法。

```
1 //blob参数是一个File对象或者Blob对象。
2 var objecturl = window.URL.createObjectURL(blob);
```

上面的代码会对二进制数据生成一个 URL, 这个 URL 可以放置于任何通常可以放置 URL 的地方, 比如 img 标签的 src 属性。需要注意的是, 即使是同样的二进制数据, 每调用一次 URL.createObjectURL 方法, 就会得到一个不一样的 URL。这个 URL 的存在时间, 等同于网页的存在时间, 一旦网页刷新或卸载, 这个 URL 就失效。(File 和 Blob 又何尝不是这样呢)除此之外, 也可以手动调用 URL.revokeObjectURL 方法, 使 URL 失效。

```
1 window.URL.revokeObjectURL(objectURL);
```

举个简单的例子。

```
1 var blob = new Blob(["Hello hanzichi"]);
2 var a = document.createElement("a");
3 a.href = window.URL.createObjectURL(blob);
4 a.download = "a.txt";
5 a.textContent = "Download";
6
7 document.body.appendChild(a);
```

页面上生成了一个超链接, 点击它就能下载一个名为 a.txt 的文件, 里面的内容是 Hello hanzichi。

四、使用createObjectURL将MediaSource和video标签连接起来

```
1 var mediaSource = new MediaSource;
2 //console.log(mediaSource.readyState); // closed
3 video.src = URL.createObjectURL(mediaSource);
4 mediaSource.addEventListener('sourceopen', sourceOpen);
```

这里传入createObjectURL的不是File或Blob了, 而是MediaSource。MS 的实例通过 URL.createObjectURL() 创建的 url 并不会同步连接到 video.src。换句话说, URL.createObjectURL() 只是底层流 (MS) 和 video.src 的连接中间者, 一旦两者连接到一起之后, 该对象就没用了。那么什么时候 MS 才会和 video.src 连接到一起呢? 创建实例都是同步的, 但是底层流和 video.src 的连接是异步的。MS 提供了一个 sourceopen 事件给我们进行这项异步处理。一旦连接到一起之后, 该 URL object 就没用了, 处于内存节省的目的, 可以使用 URL.revokeObjectURL(vidElement.src) 销毁指定的 URL object。

```
1 mediaSource.addEventListener('sourceopen', sourceOpen);
2 function sourceOpen(){
3   URL.revokeObjectURL(vidElement.src)
4 }
```

MSE 支持具体的事件

- sourceopen 绑定到媒体元素后开始触发
- sourceclosed 未绑定到媒体元素后开始触发
- sourceended 所有数据接收完成后触发

对应的属性mediaSource.readyState

- open MSE 实例, 已经绑定到了媒体元素上, 等待接受数据或者正在接受数据
- closed MSE 实例未绑定到了媒体元素上。MS刚创建时就是该状态。
- ended MSE 实例, 已经绑定到了媒体元素上, 并且所有数据都已经接受到了。当endOfStream() 执行完成, 会变为该状态。

五、设置编码类型mime 字符串

```
1 function sourceOpen(e) {
2   URL.revokeObjectURL(videoMp4.src);
3   var mime = 'video/webm; codecs="opus, vp9"';
4   // e.target refers to the mediaSource instance.
5   // Store it in a variable so it can be used in a closure.
6   var mediaSource = e.target;
7   var sourceBuffer = mediaSource.addSourceBuffer(mime);
8   // Fetch and process the video.
9 }
```

```
1 var mime = 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"'
```

首先, 前面的 video/mp4 代表这是一段 mp4 格式封装的视频, 同理也存在类似 video/webm, audio/mpeg, audio/mp4 这样的 mime 格式。一般情况下, 可以通过 [canPlayType](#) 这个方法来判断浏览器是否支持当前格式。

后面的这一段 codecs="..." 比较特别, 以逗号相隔, 分为两段:

第一段 'avc1.42E01E', 即它用于告诉浏览器关于视频编解码的一些重要信息, 诸如编码方式、分辨率、帧率、码率以及对解码器解码能力的要求。

在这个例子中, 'avc1' 代表视频采用 H.264 编码, 随后是一个分隔点, 之后是 3 个两位的十六进制的数, 这 3 个十六进制数分别代表:

1. AVCPProfileIndication(42)
2. profile_compatibility(E0)
3. AVCLLevelIndication(1E)

第一个用于标识 H.264 的 profile, 后两个用于标识视频对于解码器的要求。

对于一个 mp4 视频, 可以使用 [mp4file](#) 这样的命令行工具:

```
1 mp4file --dump xxx.mp4
2
```

找到 avcC Box 后, 就可以看到这三个值:

```
1 mp4file --dump movie.mp4
2 ...
3 type avcC (moov.trak.mdia.minf.stbl.stsd.avc1.avcC) // avc1
4 configurationVersion = 1 (0x01)
5 AVCProfileIndication = 66 (0x42) // 42
6 profile_compatibility = 224 (0xe0) // E0
7 AVCLevelIndication = 30 (0x1e) // 1E
8 ...
9
```

有一处要注意, 后面两个值(profile_compatibility, AVCLevelIndication)只是浏览器用于判断自身的解码能力能否满足需求, 所以不需要和视频完全对应, 更高也是可以的。

下面来看 codecs 的第二段 'mp4a.40.2', 这一段信息是关于音频部分的, 代表视频的音频部分采用了 AAC LC 标准:'mp4a' 代表此视频的音频部分采用 MPEG-4 压缩编码。随后是一个分隔点, 和一个十六进制数(40), 这是 ObjectTypeIndication, 40 对应的是 Audio ISO/IEC 14496-3 标准。(不同的值具有不同的含义, 详细可以参考[官方文档](#))

然后又是一个分隔点, 和一个十进制数(2), 这是 MPEG-4 Audio Object Type, 维基百科中的解释是 "MPEG-4 AAC LC Audio Object Type is based on the MPEG-2 Part 7 Low Complexity profile (LC) combined with Perceptual Noise Substitution (PNS) (defined in MPEG-4 Part 3 Subpart 4)", 具体是什么意思就不翻译了, 其实就是一种 H.264 视频中常用的音频编码规范。

这一整段 codecs 都有完善的官方文档, 可以参考:[The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types](#)

六、请求资源

sourceBuffer对象提供了一系列接口, 这里用到的是 appendBuffer 方法, 可以动态地向 MediaSource 中添加视频/音频片段(对于一个 MediaSource, 可以同时存在多个 SourceBuffer)

如果视频很长, 存在多个chunk 的话, 就需要不停地向 SourceBuffer 中加入新的 chunk。这里就需要注意一个问题了, 即 appendBuffer 是异步执行的, 在完成前, 不能 append 新的 chunk:

```
1 sourceBuffer.appendBuffer(buffer1)
2 sourceBuffer.appendBuffer(buffer2)
3 // Uncaught DOMException:
4 Failed to set the 'timestampOffset' property on 'SourceBuffer':
5 This SourceBuffer is still processing
6 an 'appendBuffer' or 'remove' operation.
7
```

而是应该监听 SourceBuffer 上的 updateend 事件, 确定空闲后, 再加入新的 chunk:

```
1 sourceBuffer.addEventListener('updateend', () => {
2   // 这个时候才能加入新 chunk
3   // 先设定新chunk加入的位置, 比如第26秒处
4   sourceBuffer.timestampOffset = 20
5   // 然后加入
6   sourceBuffer.append(newBuffer)
7 })
```

七、SourceBuffer简介

SourceBuffer 是由 mediaSource 创建, 并直接和 HTMLMediaElement 接触。简单来说, 它就是一个流的容器, 里面提供的 append(), remove() 来进行流的操作, 它可以包含一个或者多个 media segments。

```
1 interface SourceBuffer : EventTarget {
2   attribute AppendMode mode;
3   readonly attribute boolean updating;
4   readonly attribute TimeRanges buffered;
5   attribute double timestampOffset;
6   readonly attribute AudioTrackList audioTracks;
7   readonly attribute VideoTrackList videoTracks;
8   readonly attribute TextTrackList textTracks;
9   attribute double appendWindowStart;
10  attribute unrestricted double appendWindowEnd;
11  attribute EventHandler onupdatestart;
12  attribute EventHandler onupdate;
13  attribute EventHandler onupdateend;
14  attribute EventHandler onerror;
15  attribute EventHandler onabort;
16  void appendBuffer(BufferSource data);
17  void abort();
18  void remove(double start, unrestricted double end);
19 };
20
```

1.mode

上面说过, SB(SourceBuffer) 里面存储的是 media segments(就是你每次通过 append 添加进去的流片段)。SB.mode 有两种格式:

- segments: 乱序播放。通过 timestamps 来标识其具体播放的顺序。比如:20s的 buffer, 30s 的 buffer 等。
- sequence: 按序播放。通过 appendBuffer 的顺序来决定每个 mode 添加的顺序。timestamps 根据 sequence 自动产生。

那么上面两个哪个是默认值呢? 看情况, 讲真, 没骗你。当 media segments 天生自带 timestamps, 那么 mode 就为 segments, 否则为 sequence。所以, 一般情况下, 我们是不用管它的值。不过, 你可以在后面, 将 segments 设置为 sequence 这个是没毛病的。反之, 将 sequence 设置为 segments 就有问题了。

```
1 var bufferMode = sourceBuffer.mode;
2 if (bufferMode !== 'segments') {
3   sourceBuffer.mode = 'sequence';
4 }
```

segments 表示 A/V 的播放时根据你视频播放流中的 pts 来决定, 该模式也是最常用的。因为音视频播放中, 最重要的就是 pts 的排序。因为, pts 可以决定播放的时长和顺序, 如果一旦 A/V 的 pts 错开, 有可能就会造成 A/V sync drift。

sequence 则是根据空间上来进行播放的。每次通过 appendBuffer 来添加指定的 Buffer 的时候, 实际上就是添加一段 A/V segment。此时, 播放器会根据其添加的位置, 来决定播放顺序。还需要注意, 在播放的同时, 你需要告诉 SB, 这段 segment 有多长, 也就是该段 Buffer 的实际偏移量。

而该段偏移量就是由 timestampOffset 决定的。整个过程用代码描述一下就是：

```
1 sb.appendBuffer(media.segment);
2 sb.timestampOffset += media.duration;
```

另外, 如果你想手动更改 mode 也是可以的, 不过需要注意几个先决条件:

- 对应的 SB.updating 必须为 false.
- 如果该 parent MS 处于 ended 状态, 则会手动将 MS readyState 变为 open 的状态。

2.buffered

返回一个 [timeRange](#) 对象。用来表示当前被存储在 SB 中的 buffer。

3. updating

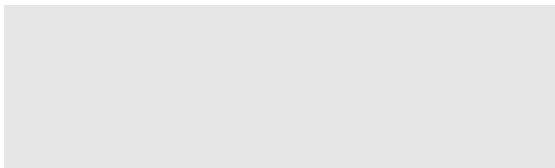
返回 Boolean, 表示当前 SB 是否正在被更新。例如: SourceBuffer.appendBuffer(), SourceBuffer.appendStream(), SourceBuffer.remove() 调用时。

- true: 当前 SB 正在处理添加或者移除的 segment
- false: 当前 SB 处于空闲状态。当且仅当 updating = false 的时候, 才可以对 SB 进行额外的操作。

SB 内部的 buffer 管理主要是通过 appendBuffer(BufferSource data) 和 remove() 两个方法来实现的。当然, 并不是所有的 Buffer 都能随便添加给指定的 SB, 这里是需要条件和相关顺序的。

- 该 buffer, 必须满足 MIME 限定的类型
- 该 buffer, 必须包含 initialization segments (IS) 和 media segments (MS)

下图是相关的支持 [MIME](#):



这里需要提醒大家一点, MSE 只支持 fmp4 的格式。具体内容可以参考: [学好 MP4, 让直播更给力](#)。上面提到的 IS 和 MS 实际上就是 FMP4 中不同盒子的集合而已。

4.事件

在 SB 中, 相关事件触发包括:

- updatestart: 当 updating 由 false 变为 true。
- update: 当 append()/remove() 方法被成功调用完成时, updating 由 true 变为 false。
- updateend: append()/remove() 已经结束
- error: 在 append() 过程中发生错误, updating 由 true 变为 false。
- abort: 当 append()/remove() 过程中, 使用 abort() 方法废弃时, 会触发。此时, updating 由 true 变为 false。

注意上面有两个事件比较类似: update 和 updateend。都是表示处理的结束, 不同的是, update 比 updateend 先触发。

```
1 sourceBuffer.addEventListener('updateend', function (e) {
2   // 当指定的 buffer 加载完后, 就可以开始播放
3   mediaSource.endOfStream();
4   video.play();
5 });
```

5.添加/移除 buffer

在添加 Buffer 的时候, 你需要了解你所采用的 mode 是哪种类型, sequence 或者 segments。这两种是完全两种不同的添加方式。

(1)segments

这种方式是直接根据 MP4 文件中的 pts 来决定播放的位置和顺序, 它的添加方式极其简单, 只需要判断 updating === false, 然后, 直接通过 appendBuffer 添加即可。

```
1 if (!sb.updating) {
2   let MS = this._mergeBuffer(media.tmpBuffer);
3
4   sb.appendBuffer(MS); // ****
5
6   media.duration += lib.duration;
7   media.tmpBuffer = [];
8 }
```

(2)sequence

如果你是采用这种方式进行添加 Buffer 进行播放的话, 那么你也就没必要了解 FMP4 格式, 而是了解 MP4 格式。因为, 该模式下, SB 是根据具体添加的位置来进行播放的。所以, 如果你是 FMP4 的话, 可能就有点不适合了。针对 sequence 来说, 每段 buffer 都必须有自己本身的指定时长, 每段 buffer 不需要参考的 baseDts, 即, 他们直接可以毫无关联。那 sequence 具体怎么操作呢?

简单来说, 在每一次添加过后, 都需要根据指定 SB 上的 timestampOffset。该属性, 是用来控制具体 Buffer 的播放时长和位置的。

```
1 if (!sb.updating) {
2   let MS = this._mergeBuffer(media.tmpBuffer);
3
4   sb.appendBuffer(MS); // ****
5
6   sb.timestampOffset += lib.duration; // ****
7   media.tmpBuffer = [];
8 }
```

上面两端打 * 号的就是重点内容。该方式比较容易用来直接控制 buffer 片段的添加, 而不用过度关注相对 baseDTS 的值。

6.控制播放片段

如果要在 video 标签中控制指定片段的播放, 一般是不可能的。因为, 在加载整个视频 buffer 的时候, 视频长度就已经固定的, 剩下的只是你如果在 video 标签中控制播放速度和音量大小。而在 MSE 中, 如何在已获得整个视频流 Buffer 的前提下, 完成底层视频 Buffer 的切割和指定时间段播放呢?

这里, 需要利用 SB 下的 appendWindowStart 和 appendWindowEnd 这两个属性。

他们两个属性主要是为了设置, 当有视频 Buffer 添加时, 只有符合在 [start,end] 之间的 media frame 才能 append, 否则, 无法 append。例如:

```
1 | sourceBuffer.appendWindowStart = 2.0;
2 | sourceBuffer.appendWindowEnd = 5.0;
```

设置添加 Buffer 的时间戳为 [2s,5s] 之间。appendWindowStart 和 appendWindowEnd 的基准单位为 s。该属性值, 通常在添加 Buffer 之前设置。

6.SB 内存释放

SB 内存释放其实就和在 JS 中, 将一个变量指向 null 一样的过程。

```
1 | var a = new ArrayBuffer(1024 * 1000);
2 | a = null; // start garbage collection
```

在 SB 中, 简单的来说, 就是移除指定的 time ranges' buffer。需要用到的 API 为:

```
1 | remove(double start, unrestricted double end);
```

具体的步骤为:

- 找到具体需要移除的 segment。
- 得到其开始(start)的时间戳(以 s 为单位)
- 得到其结束(end)的时间戳(以 s 为单位)
- 此时, updating 为 true, 表明正在移除
- 完成之后, 出发 updateend 事件

如果, 你想直接清空 Buffer 重新添加的话, 可以直接利用 abort() API 来做。它的工作是清空当前 SB 中所有的 segment, 使用方法也很简单, 不过就是需要注意不要和 remove 操作一起执行。更保险的做法就是直接, 通过 updating===false 来完成:

```
1 | if(sb.updating===false){
2 |   sb.abort();
3 | }
```

这时候, abort 的主要流程为:

- 确保 MS.readyState=="open"
- 将 appendWindowStart 设置为 pts 原始值, 比如, 0
- 将 appendWindowEnd 设置为正无限大, 即, Infinity。

abort() 用来放弃当前 append 流的操作。不过, 该方法的业务场景也比较有限。它只能用在当 SB 正在更新流的时候。即, 此时通过 fetch 已经接受到新流, 并且使用 appendBuffer 添加, 此为开始的时间。然后到 updateend 事件触发之前, 这段时间之内调用 `abort()`。有一个业务场景是, 当用户移动进度条, 而此时 fetch 已经获取前一次的 media segments, 那么可以使用 `abort` 放弃该操作, 转而请求新的 media segments。具体可以参考:[abort 使用](#)

7.appendBuffer(ArrayBuffer)

用来添加 ArrayBuffer。该 ArrayBuffer 一般是通过 fetch 的 `response.arrayBuffer()` 来获取的。在使用 addSourceBuffer 创建之前, 还需要保证当前浏览器是否支持该编码格式。当然, 不支持也行, 顶多是当前 MS 报错, 断掉当前 JS 线程。

八、MediaSource简介

```
1 | [Constructor]
2 | interface MediaSource : EventTarget {
3 |   readonly attribute SourceBufferList    sourceBuffers;
4 |   readonly attribute SourceBufferList    activeSourceBuffers;
5 |   readonly attribute ReadyState         readyState;
6 |   attribute unrestricted double duration;
7 |   attribute EventHandler                 onsourceopen;
8 |   attribute EventHandler                 onsourceended;
9 |   attribute EventHandler                 onsourceclose;
10 |
11 |   SourceBuffer addSourceBuffer(DOMString type);
12 |   void         removeSourceBuffer(SourceBuffer sourceBuffer);
13 |   void         endOfStream(optional EndOfStreamError error);
14 |   void         setLiveSeekableRange(double start, double end);
15 |   void         clearLiveSeekableRange();
16 |   static boolean isTypeSupported(DOMString type);
17 | };
```

1.isTypeSupported

isTypeSupported 主要是用来检测 MS 是否支持某个特定的编码和容器盒子。例如:

```
1 | MediaSource.isTypeSupported("video/mp4; codecs='avc1.42E01E, mp4a.40.2'")
```

这里有一份具体的 [mimeType](#) 参考列表。

2.addSourceBuffer

用来返回一个具体的视频流 SB, 接受一个 mimeType 表示该流的编码格式。例如:

```
1 | var mimeType = 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"';
2 | var sourceBuffer = mediaSource.addSourceBuffer(mimeType);
```

3.removeSourceBuffer

用来移除某个 sourceBuffer。比如当前流已经结束, 那么你就没必要再保留当前 SB 来占用空间, 可以直接移除。具体格式为:

```
1 | mediaSource.removeSourceBuffer(sourceBuffer);
```

4.endOfStream()

用来表示接受的视频流的停止。注意, 这里并不是断开, 相当于只是下好了一部分视频, 然后你

可以进行播放。此时，MS 的状态变为:ended, 例如：

```
1  var mediaSource = this;
2  var sourceBuffer = mediaSource.addSourceBuffer(mimeCodec);
3  fetchAB(assetURL, function (buf) {
4      sourceBuffer.addEventListener('updateend', function (_) {
5          mediaSource.endOfStream(); // 结束当前的接受
6          video.play(); // 可以播放当前获得的流
7      });
8      sourceBuffer.appendBuffer(buf);
9  });
```

5.sourceBuffers

sourceBuffers 是 MS 实例上的一个属性。它返回的是一个 SourceBufferList 的对象，里面可以获取当前 MS 上挂载的所有 SB。不过，只有当 MS 为 open 状态的时候，它才可以访问。具体使用为：

```
1 | let SBs = mediaSource.sourceBuffers;
```

那我们怎么获取到具体的 SB 对象呢？因为，其返回值是 SourceBufferList 对象，具体格式为：

```
1  interface SourceBufferList : EventTarget {
2      readonly attribute unsigned long length;
3      attribute EventHandler onaddsourcebuffer;
4      attribute EventHandler onremovesourcebuffer;
5      getter SourceBuffer (unsigned long index);
6  };
```

简单来说，你可以直接通过 index 来访问具体的某个 SB：

```
1 | let SBs = mediaSource.sourceBuffers;
2
3 | let SB1 = SBs[0];
```

SB 对象还提供了 addsourcebuffer 和 removesourcebuffer 事件，如果你想监听 SB 的变化，可以直接通过 SBL 来做。这也是为什么 MS 没有提供监听事件的一个原因。所以，删除某一个 SB 就可以通过 SBL 查找，然后，利用 remove 方法移除即可：

```
1 | let SBs = mediaSource.sourceBuffers;
2
3 | let SB1 = SBs[0];
4
5 | mediaSource.removeSourceBuffer(SB1);
```

6.activeSourceBuffers

activeSourceBuffers 实际上是 sourceBuffers 的子集，返回的同样也是 SBL 对象。为什么说也是子集呢？

因为 ASBs 包含的是当前正在使用的 SB。因为前面说了，每个 SB 实际上都可以具体代表一个 track，比如，video track，audio track，text track 等等，这些都算。那怎么标识正在使用的 SB 呢？很简单，不用标识啊，因为控制哪一个 SB 正在使用是你来决定的。如果非要标识，就需要使用到 HTML 中的 video 和 audio 节点。通过

```
1  audioTrack = media.audioTracks[index]
2  videoTrack = media.videoTracks[index]
3
4  // media 为具体的 video/audio 的节点
5  // 返回值就是 video/audio 的底层 tracks
6
7
8  audioTrack = media.audioTracks.getTrackById( id )
9  videoTrack = media.videoTracks.getTrackById( id )
10
11 videoTrack.selected // 返回 boolean 值，标识是否正在被使用
```

上面的代码只是告诉你，正在使用的含义是什么。对于我们实际编码的 SB 来说，并没有太多关系，了解就好。上面说了 ASBs 返回值也是一个 SBL，所以，使用方式可以直接参考 SBL 即可。

7.状态切换

要说道状态切换，我们得先知道 MS 一共有几个状态值。MS 本身状态并不复杂，一共只有三个状态值：

```
1  enum ReadyState {
2      "closed",
3      "open",
4      "ended"
5  };
```

- closed: 当前的 MS 并没有和 HTMLMedia 元素连接
- open: MS 已经和 HTMLMedia 连接，并且等待新的数据被添加到 SB 中去。
- ended: 当调用 endOfStream 方法时会触发，并且此时依然和 HTMLMedia 元素连接。

记住，closed 和 ended 到的区别关键点在于有没有和 HTMLMedia 元素连接。

其对应的还有三个监听事件：

- sourceopen: 当状态变为 open 时触发。常常在 MS 和 HTMLMedia 绑定时触发。
- sourceended: 当状态变为 ended 时触发。
- sourceclose: 当状态变为 closed 时触发。

那哪种条件下会触发呢？

(1)sourceopen 触发

sourceopen 事件相同于是一个总领事件，只有当 sourceopen 时间触发后，后续对于 MS 来说，才是一个可操作的对象。通常来说，只有当 MS 和 video 元素成功绑定时，才会正常触发：

```
1 | let mediaSource = new MediaSource();
2 | vidElement.src = URL.createObjectURL(mediaSource);
```

其实这简单的来说，就是给 MS 添加 HTML media 元素。其整个过程为：

- 先延时 media 元素的 load 事件，将 delaying-the-load-event-flag 设置为 false
- 将 readyState 设置为 open。

触发 MS 的 sourceopen 事件

(2)sourceended 触发

sourceended 的触发条件其实很简单, 只有当你调用 endOfStream 的时候, 会进行相关的触发。`mediaSource.endOfStream()`; 这个就没啥需要过多讲的了。

(3)sourceclose 的触发

sourceclose 是在 media 元素和 MS 断开的时候, 才会触发。那这个怎么断开呢? 难道直接将 media 的元素的 src 直接设置为 null 就 OK 了吗? 要是这样, 我就日了狗了。MS 会这么简单么? 实际上并不, 如果要手动触发 sourceclose 事件的话, 则需要下列步骤:

- 将 readyState 设置为 closed
- 将 MS.duration 设置为 NaN
- 移除 activeSourceBuffers 上的所有 Buffer
- 触发 activeSourceBuffers 的 removesourcebuffer 事件
- 移除 sourceBuffers 上的 SourceBuffer。
- 触发 sourceBuffers 的 removesourcebuffer 事件
- 触发 MediaSource 的 sourceclose 事件

到这里, 三个状态事件基本就介绍完了。不过, 感觉只有 sourceopen 才是最有用的一个。

8.track 的切换

track 这个概念其实是音视频播放的轨道, 它和 MS 没有太大的关系。不过, 和 SB 还是有一点关系的。因为, 某个一个 SB 里面可能会包含一个 track 或者说是几个 track。所以, 推荐某一个 SB 最好包含一个值包含一个 track, 这样, 后面的 track 也方便更换。在 track 中的替换里, 有三种类型, audio, video, text 轨道。

(1)video 切换

切换的含义有两种, 一种是移除原有的, 一种是添加新的。这里, 我们需要分两部分来讲解。

(a)移除原有不需要 track

- 从 activeSourceBuffers 移除与当前 track 相关的 SB
- 触发 activeSourceBuffers 的 removesourcebuffer 事件

(b)添加指定的 track

- 从 activeSourceBuffers 添加指定的 SourceBuffer
- 触发 activeSourceBuffers 的 addsourcebuffer 事件

(2)audio 切换

audio 的切换和 video 的过程一模一样。这里我就不过多赘述了。

9.MS duration 修正机制

MS 的 duration 实际上就是 media 中播放的时延。通常来说, A/V track 实际上是两个独立的播放流, 这中间必定会存在先关的差异时间。但是, media 播放机制永远会以最长的 duration 为准。这种情况对于 live stream 的播放, 特别适合。因为 liveStream 是不断动态添加 buffer, 但是 buffer 内部会有一定的时长的, 而 MS 就需要针对这个 buffer 进行动态更新。整个更新机制为:

- 当前 MS.duration 更新为 new duration。
- 如果 new duration 比 sourceBuffers 中的最大的 pts 小, 这时候就会报错。
- 让最后一个的 sample 的 end time 为所后 timeRanges 的 end time。
- 将 new duration 设置为当前 SourceBuffer 中最大的 endTime。
- 将 video/audio 的播放时长(duration) 设置为最新的 new duration。

10.如何界定 track

这里先声明一下, track 和 SB 并不是一一对应的关系。他们的关系只能是 SB : track = 1: 1 or 2 or 3。即, 一个 SB可能包含, 一个 A/V track(1), 或者, 一个 Video track , 一个Audio track(2), 或者再额外加一个 text track(3)。

上面也说过, 推荐将 track 和 SB 设置为一一对应的关系, 应该这样比较好控制, 比如, 移除或者同步等操作。具体编码细节我们有空再说, 这里先来说一下, SB 里面怎么决定 track 的播放。

track 最重要的特性就是 pts, duration, access point flag。track 中最基本的单位叫做 Coded Frame, 表示具体能够播放的音视频数据。它本身其实就是一些列的 media data, 并且这些 media data 里面必须包含 pts, dts, sampleDuration 的相关信息。在 SB 中, 有几个基本内部属性是用来标识前面两个字段的。

- last decode timestamp: 用来表示最新一个 frame 的编码时间(pts)。默认为 null 表示里面没有任何数据
- last frame duration: 表示 coded frame group 里面最新的 frame 时长。
- highest end timestamp: 相当于就是最后一个 frame 的 pts + duration
- need random access point flag: 这个就相当于同步帧的意思。主要设置是根据音视频流 里面具体字段决定的, 和前端这边编码没关系。
- track buffer ranges: 该字段表示的是 coded frame group 里面, 每一帧对应存储的 pts 范围。

这里需要特别说一下 last frame duration 的概念, 其实也就是 Coded Frame Duration 的内容。Coded Frame Duration 针对不同的 track 有两种不同的含义。一种是针对 video/text 的 track, 一种是针对 audio 的 track:

- video/text: 其播放时长(duration)直接是根据 pts 直接的差值来决定, 和你具体播放的 samplerate 没啥关系。虽然, 官方也有一个计算 refsampelDuration 的公式: duration = timescale / fps, 不过, 由于视频的帧率是动态变化的, 没什么太大的作用。
- audio: audio 的播放时长必须是严格根据采样频率来的, 即, 其播放时间必须和你自己定制的 timescale 以及 sampleRate 一致才行。针对 AAC, 因为其采样频率常为 44100Hz, 其固定播放时长则为: duration = 1024 / sampleRate * timescale

所以, 如果你在针对 unstable stream 做同步的话, 一定需要注意这个坑。有时候, dts 不同步, 有可能才是真正的同步。

我们再回到上面的子 title 上-- 如果界定 track。一个 SB 里面是否拥有一个或者多个 track, 主要是根据里面的视频格式来决定的。打个比方, 比如, 你是在编码 MP4 的流文件。它里面的 track 内容, 则是根据 moov box 中的 trak box 来判断的。即, 如果你的 MP4 文件只包含一个, 那么, 里面的 track 也只有有一个。

九、MSE兼容性 caniuse

1.iOS Safari 不支持 Media Source Extensions 因此无法使用 flv.js

以下摘自HTML5 媒体源扩展(MSE):把影视制作级别的视频格式带入 Web

要覆盖99%的用户, 我们需要做一个视频流兼容设置, 这样也可以让那些不支持MSE的浏

...