PPTV API 流媒体协议

（PPTV API Streaming Protocol）

修订记录

日期	作者	版本	主要内容	备注
2012-2-9	郭春茂	1.0	初稿	

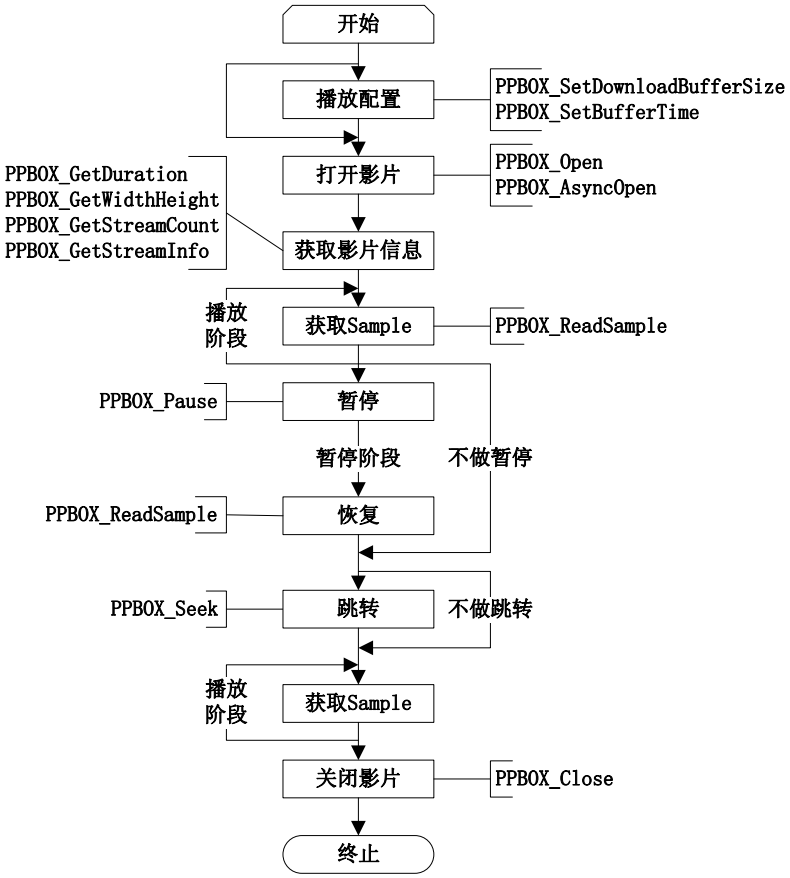
目录

1 前言

PPTV API 流媒体协议是基于 C 语言的接口函数集。协议定义了支持基本播放控制所需的接口方法，以及这些方法的调用序列。为了明确接口调用方和接口实现方的界限，协议也规定了接口的框架模型。

2 播放流程

下图概括性的描述了 API 协议方法调用流程。



2.1 正常播放流程

正常播放流程是指从开始播放，之后在没有人为操作、数据正常下载的情况下，完整流畅的播放整个影片的过程。正常播放流程包括打开影片、获取影片信息、获取影片数据、播放到末尾结束这几个部分。

1. **打开影片**

调用 ***PPBOX_Open*** 或者 ***PPBOX_AsyncOpen*** 打开影片。***PPBOX_Open*** 会阻塞调用线程，而 ***PPBOX_AsyncOpen*** 立即返回，它是通过回调函数的方式通知打开完成。

2. **获取影片信息**

调用 ***PPBOX_GetDuration***、***PPBOX_GetStreamCount***、***PPBOX_GetStreamInfo*** 获取影片的信息。这些信息用来初始化解码器和 UI 进度条等。

3. **获取影片数据**

循环调用 ***PPBOX_GetSample(Ex2)***，获取一个个数据帧，音视频帧是交错的，每个帧里面包含流索引、时间戳和帧数据。这些帧在通过解码器解码和时间同步后展现出来。

4. **播放到末尾结束**

当 ***PPBOX_GetSample(Ex2)***返回错误码 ***ppbox_stream_end*** 时，说明已经到达影片结尾，这时候通过 ***PPBOX_Close*** 关闭影片，释放资源。

2.2 卡顿缓冲流程

当数据下载跟不上播放进度，PPBOX_GetSample(Ex2)返回 ppbox_would_block 的时候，播放器进入卡顿缓冲流程，根据缓冲策略的不同，该流程有两种实现方式。

2.2.1 调用方缓冲策略

此策略下，缓冲区由播放器提供，已经缓冲的数据量（缓冲了多长时间）由播放器计算。具体步骤如下：

1. **获取下一帧**

通过 ***PPBOX_GetSample(Ex2)*** 读取下一帧，如果成功转第二步，否则（返回 ***ppbox_would_block*** 时）转第三步

2. **填充缓冲区**

将 Sample 填充到缓冲区，如果缓冲的数据已经足够，结束该流程，否则转第一步循环。

3. **暂停一段时间**

将缓冲线程暂停一段时间，然后转第一步循环

2.2.2 实现方缓冲策略

此策略下，播放器不需要额外的缓冲区，已经缓冲的数据量由 PPBOX 库在内部计算，外部调用的步骤如下：

1. **检查缓冲状态**

   通过 ***PPBOX_GetPlayMsg*** 获取，如果成功转第二步，否则（返回 ***ppbox_would_block*** 时）转第三步

2. **检查缓冲是否完成**

   检查缓冲的数据已经足够，如果已经足够，结束该流程，否则转第一步循环。

3. **暂停一段时间**

   将缓冲线程暂停一段时间，然后转第一步循环

2.3 拖动播放流程

当在播放阶段，调整播放位置时，进入拖动播放流程，该流程包括下面的两个步骤：

1. **确保停止调用方法**

   先确保 ***PPBOX_GetSample(Ex2)***或者 ***PPBOX_GetPlayMsg*** 循环结束，并且不再调用任何方法。

2. **调整播放位置**

   调用 ***PPBOX_Seek*** 调整播放位置，如果方法返回 ***ppbox_would_block***，需要进入卡顿缓冲流程。

2.4 取消打开流程

取消打开流程和下面的结束播放流程对观看的用户来说，都是退出播放。在播放器内部，他们是有区别的：当打开的过程还没有结束时，属于取消打开流程；当打开完成，已经进入了播放阶段时，属于结束播放流程。对于第一种情形，也就是取消打开流程，需要下面的两个步骤：

1. **取消打开过程**

   调用 ***PPBOX_Close*** 取消进行中的打开过程；对于同步打开，因为已经阻塞在 ***PPBOX_Open*** 中了，需要在另一个线程调用 ***PPBOX_Close***。

2. **等待打开过程结束**

   需要等待 ***PPBOX_Open*** 返回或者 ***PPBOX_AsyncOpen*** 的回调被调用，该流程才算结束。

2.5 结束播放流程

结束播放流程是播放阶段收到用户退出信号后的处理过程，包括下面的两个步骤：

1. **确保停止调用方法**

先确保 ***PPBOX_GetSample(Ex2)***或者 ***PPBOX_GetPlayMsg*** 循环结束，并且不再调用任何方法。

2. **关闭影片**

调用 ***PPBOX_Close*** 关闭影片，释放资源。

3 模型描述

3.1 同步非阻塞

协议采用同步非阻塞模型，所以方法是同步非阻塞的，也有例外，如同步打开（***PPBOX_Open***）方法是阻塞的。在同步非阻塞模型下，如果操作不能立即完成，会返回相应错误（***ppbox_would_block***），当返回该错误时，程序可以睡眠一段时间，或者转而去处理其他任务，一段时间后再来重试。

有可能返回该错误的方法有（其他方法都是能够立即完成）：

- 取 Sample——***PPBOX_ReadSample***
- 播放进度跳转——***PPBOX_Seek***（该方法不能立即完成时，不需要重试，返回“不能立即完成”错误意味着下一个取 Sample 操作肯定不能立即完成，程序可以转而处理其他任务）

3.2 单线程

为了减少线程锁开销，所有方法都是非线程安全的，只能在同一个线程调用。但是打开视频方法（包括同步打开视频 ***PPBOX_Open*** 和异步打开视频 ***PPBOX_AsyncOpen***）与关闭视频方法（***PPBOX_Close***）可以多线程调用。

特别需要注意的是：异步打开过程持续到异步回调被触发前，所以在异步回调被触发之前不能调用任何方法（保证方法不被并发执行），除了关闭视频（***PPBOX_Close***）方法。

阻塞在打开视频中的时候，可以通过另一个线程调用关闭视频（***PPBOX_Close***）方法，使阻塞中断，提前结束，这种情况下仍要注意线程安全性。实际上，关闭视频（***PPBOX_Close***）方法只是让打开过程尽快结束，但是并没有马上结束，外部调用者要在确保打开过程结束后才能调用其他方法。对于同步打开视频，需要确保同步打开视频（***PPBOX_Open***）方法返回；对于异步打开视频的情形，可以把异步回调被触发当作打开过程结束的标识。

3.3 内存管理

为了平滑网络抖动，Demux 模块内部使用循环缓存（Sycle Buffer）保存一定量的视频

数据，该缓存在视频打开（***PPBOX_Open***）的时候分配，并一直保持到视频被关闭（***PPBOX_Close***）。程序可以通过缓存配置（***PPBOX_SetDownloadBufferSize***）方法设置可以使用的缓存大小（默认是 20M），但是必须在视频打开之前配置才能生效。

其他内部结构的零碎小缓存使用 C++默认内存分配器分配，在视频被关闭（***PPBOX_Close***）的时候释放。

3.4 方法缓存

当方法需要返回的数据大小不确定时，所需的内存由模块内部申请，这些内存是复用的并动态增长，因此只在下次调用同一方法前有效，程序不应当保存这些缓存指针，而是应该马上处理（如拷贝到自己的缓存中）然后忽略它。这些缓存在视频被关闭（***PPBOX_Stop***）的时候释放。下列方法使用复用缓存：

- 取媒体流信息（***PPBOX_GetStreamInfo***）方法

  返回的解码器配置数据（***PPBOX_StreamInfo.format_buffer***）

- 取 Avc 解码配置（***PPBOX_GetAvcConfig***）方法

  返回的数据（***buffer***）

- 取 Sample（***PPBOX_ReadSample***）方法

  返回的 Sample 的数据（***PPBOX_Sample.buffer***）

- 取错误描述信息（***PPBOX_GetLastErrorMsg***）方法

  返回的错误描述信息（返回值）

3.5 影片缓冲

为了保证平滑播放，通常需要缓存一定长度的影片数据，才开始播放。很多解码器有自己的内部缓冲，这样的情况下只要不停的调用取 Sample（***PPBOX_ReadSample***）方法，向解码器填充时间就行了。

作为另一种替代方案，中间层也提供了一种简单的缓冲机制：通过在打开前调用 ***PPBOX_SetBufferTime*** 设置可接受缓冲时间，然后在打开或者跳转之后可以通过 ***PPBOX_GetPlayMsg*** 不间断地检查缓冲状态，直到状态变成播放状态，此时可以按照正常的解码速度读取 Sample；当读取 Sample 返回 ***ppbox_would_block*** 时，也需要重新缓冲。

3.6 下载驱动

中间层从网络获取数据，但是并没有启动专门的下载线程来下载数据。模块内部借用调用者线程来完成下载工作，在所有方法中能够驱动下载的有：

- 打开视频（***PPBOX_Open*** 或者 ***PPBOX_AysncOpen***）

- 跳转视频（***PPBOX_Seek***）
- 读取 Sample（***PPBOX_ReadSample***，***PPBOX_ReadSampleEx2***）
- 获取播放状态（***PPBOX_GetPlayMsg***）

前两个接口不能频繁调用，因此，为了保证数据下载不会停滞，必须频繁的调用“读取 Sample”接口或者“获取播放状态”接口。

如果依赖于中间层内部的缓冲机制（通过 ***PPBOX_GetPlayMsg*** 检查缓冲状态、缓冲百分比），就不能在没有完成缓冲时简单地暂停一段时间再重复检查，那样的话，由于下载驱动力少（调用 ***PPBOX_GetPlayMsg*** 不够频繁），将使下载速度变得很慢。

但是也不能太频繁的调用 ***PPBOX_GetPlayMsg***，那样会在网络上没有数据到达时陷入循环消耗。事实上，通过检查 ***PPBOX_GetPlayMsg*** 的返回值，在返回 ***ppbox_would_block*** 时暂停一段时间，可以在保证正常下载速度的前提下有效的控制循环次数，降低 CPU 消耗。

3.7 单播放实例

只能进行单视频播放，不能同时播放多个视频。

4 方法索引

4.1 播放配置

4.1.1 设置下载缓冲区大小

void PPBOX_SetDownloadBufferSize(PP_uint32 length);

参数：

length 下载缓冲区大小（单位：字节）

返回值：无

说明：

主要用于控制内存，如果下载速度大于 ***PPBOX_ReadSample*** 的速度，那么下载的数据将存放于内存之中，当内存中的下载缓冲大于这个预设值，那么将停止下载。直到被调用了 ***PPBOX_ReadSample***，少了一些内存占用后，再继续下载。

需要在调用 ***PPBOX_Open***，***PPBOX_AsyncOpen*** 之前设置才有效。

4.1.2 设置可接受缓冲时间

void PPBOX_SetPlayBufferTime(uint32 time)

参数：

time 可接受的缓冲时间（单位：毫秒）

返回值：无

说明：

主要用于计算播放状态，如果不调用这个函数，默认 3s。如果 下载缓冲区数据的总时间小于可接受缓冲时间，则处于 ***buffering*** 状态；如果下载缓冲区数据的总时间大于播放缓冲时间则处于 ***playing*** 状态；如果人为调用了 ***PPBOX_Pause*** 使之暂停的，则处于 p***aused*** 状态。

4.2 播放控制

4.2.1 同步打开视频

PP_int32 PPBOX_Open(PP_char const * playlink);

参数：

playlink 播放串

返回值：错误码

返回***ppbox_success***表示成功，其他见错误码表。可以通过***PPBOX_GetLastErrorMsg***获取进一步的错误描述。

说明：

函数阻塞执行。

4.2.2 异步打开视频

typedef void (*PPBOX_CallBack)(PP_int32);

void PPBOX_AsyncOpen(PP_char const * playlink, PPBOX_CallBack callback);

参数：

playlink 播放串

callback 回调函数

返回值：无

说明：

在调用 ***PPBOX_AsyncOpen*** 之后，没有被回调 handler 之前，可以调用 ***PPBOX_Close*** 将其取消掉，即使被取消，回调 ***callback*** 也会被调用，并指示相应错误。异步打开失败，仍需要调用 ***PPBOX_Close*** 关闭视频，以释放资源。

回调函数定义：

void PPBOX_CallBack (PP_int32 err);

参数：

err 返回错误码，意义同 ***PPBOX_Open*** 返回值

返回值：无

4.2.3 关闭视频

void PPBOX_Close();

参数：无

返回值：无

说明：

在异步打开启动后，调用该方法将取消还未完成的异步打开操作。

4.2.4 暂停视频

PP_int32 PPBOX_Pause();

参数：无

返回值：错误码

返回 ***ppbox_success*** 表示成功，其他见错误码表。

可以通过 ***PPBOX_GetLastErrorMsg*** 获取进一步的错误描述。

说明：

处于暂停状态时，只需要调用 ***PPBOX_ReadSample*** 即可恢复。

4.2.5 跳转视频

PP_int32 PPBOX_Seek(PP_uint32 time)

参数：

start_time 拖动的时间点，单位毫秒

返回值：错误码

返回 ***ppbox_success*** 表示成功，返回 ***ppbox_would_block*** 表示跳转正在处理中，此错误可以忽略，更好的处理方式是过一段时间再去读取 Sample。其他见错误码表。可以通过 ***PPBOX_GetLastErrorMsg*** 获取进一步的错误描述。

4.3 获取视频数据

4.3.1 获取流的个数

PP_uint32 PPBOX_GetStreamCount();

参数：无

返回值：流的个数

说明：

如果返回 0，说明发生错误，可以通过 ***PPBOX_GetLastError*** 获取错误码、通过 ***PPBOX_GetLastErrorMsg*** 获取错误描述。

4.3.2 获得流的格式信息

4.3.2.1获得流的格式的基本信息

PP_int32 PPBOX_GetStreamInfo(PP_int32 index, PPBOX_StreamInfo* stream_info);

参数：

index 流的编号

stream_info 作为返回参数接收流格式信息

返回值：错误码

返回***ppbox_success***表示成功，其他见错误码表。可以通过***PPBOX_GetLastErrorMsg***获取进一步的错误描述。

流描述结构定义：

```
struct PPBOX_StreamInfo {
    PP_int32 type;
    PP_int32 sub_type;
    PP_int32 format_type;
    PP_uint32 format_size;
    PP_uchar const * format_buffer;
};
```

其中：

type 流的基本类型，具体参见第 6 节：媒体格式

sub_type 流的子类型，具体参见第 6 节：媒体格式

format_type 解码器需要的配置数据的类型，具体参见第 6 节：媒体格式

format_size 解码器需要的配置数据的长度

format_buffer 解码器需要的配置数据的内容，具体参见第 6 节：媒体格式

说明：

返回的***format_buffer***指向的内存是由函数内部申请，在调用***PPBOX_Close***的时候释放。配置数据的格式为官方文档中标准定义。

4.3.2.2获得流的格式的扩充信息

PP_int32 PPBOX_GetStreamInfoEx(PP_int32 index, PPBOX_StreamInfoEx* stream_info);

参数：

index 流的编号

stream_info 作为返回参数接收流格式信息

返回值：错误码

返回***ppbox_success***表示成功，其他见错误码表。可以通过***PPBOX_GetLastErrorMsg***获取进一步的错误描述。

流描述结构定义：

```
struct PPBOX_StreamInfoEx {
```

```
    PP_int32 type;
    PP_int32 sub_type;
    union
    {
        PPBOX_VideoInfo video_format;
        PPBOX_AudioInfo audio_format;
    };
    PP_int32 format_type;
    PP_uint32 format_size;
    PP_uchar const * format_buffer;
};
```

相对于 ***PPBOX_StreamInfo*** 增加了更多的音视频描述信息。其中 ***video_format***，***audio_format*** 的定义如下：

```
struct PPBOX_VideoInfo
{
    PP_int32 width;
    PP_int32 height;
    PP_int32 frame_rate;
};
```

其中：

width	保存获得的图像宽度
height	保存获得的图像高度
frame_rate	视频帧率

```
struct PPBOX_AudioInfo
{
    PP_int32 channel_count;
    PP_int32 sample_size;
    PP_int32 sample_rate;
};
```

其中：

channel_count	音频 channel 数
sample_size	音频采样率
sample_rate	音频的采样位宽

4.3.3 获得视频总时长

PP_uint32 PPBOX_GetDuration()

参数：无

返回值: 文件的总时长（单位：毫秒）

如果返回值为 0，说明出错，可以通过 ***PPBOX_GetLastError*** 获取错误码、通过 ***PPBOX_GetLastErrorMsg*** 获取错误描述。

4.3.4 获得视频图像尺寸

PP_int32 PPBOX_GetWidthHeight(PP_uint32* pwidth, PP_uint32* pheight);

参数：

pwidth 返回参数，保存获得的图像宽度

pheight 返回参数，保存获得的图像高度

返回值：错误码

返回***ppbox_success***表示成功，其他见错误码表。可以通过***PPBOX_GetLastErrorMsg***获取进一步的错误描述。

4.3.5 获得 AVC 配置数据

PP_int32 PPBOX_GetAvcConfig(PP_uchar const ** buffer, PP_uint32* length);

参数：

buffer 返回参数，保存配置数据的缓冲区指针

length 返回参数，保存配置数据的长度

返回值：错误码

返回***ppbox_success***表示成功，其他见错误码表。可以通过***PPBOX_GetLastErrorMsg***获取进一步的错误描述。

说明：

返回的***buffer***指向的内存是由函数内部申请，在调用***PPBOX_Close***的时候释放。AVC 配置数据的格式与 H264 视频格式信息（***StreamInfo***）里中的***format_buffer***一样。

注：不推荐调用该方法，使用***PPBOX_GetStreamInfo(Ex)***能够获取更多信息。

4.3.6 读取 Sample

4.3.6.1标准读取 Sample

PP_int32 PPBOX_ReadSample(PPBOX_Sample* sample);

参数：

sample 读取后的结果写入这个结构体中

返回值：错误码

返回***ppbox_success***表示成功，返回***ppbox_would_block***表示这次没有读到数据，返回***ppbox_stream_end***表示视频流已经结束。其他见错误码表。可以通过***PPBOX_GetLastErrorMsg***获取进一步的错误描述。

Sample 的定义：

```
struct PPBOX_Sample
{
    PP_uint32 stream_index;
    PP_uint32 start_time;
```

PP_uint32_t buffer_length;
PP_uchar const * buffer;
};

其中：

stream_index 流的编号
start_time 这个 Sample 对应的时间戳（单位：毫秒）
buffer_length Sample 的数据长度
buffer Sample 的数据

说明：

返回的 ***sample.buffer*** 指向的内存是由函数内部申请，有效期保持到下次调用 ***PPBOX_ReadSample*** 或者调用 ***PPBOX_Close*** 前。

sample.buffer 中内容的具体格式，参见第 6 节：媒体格式。

4.3.6.2扩充读取 Sample

PP_int32 PPBOX_ReadSampleEx2(PPBOX_SampleEx2* sample);

参数：

sample 读取后的结果写入这个结构体中

返回值：错误码

返回 ***ppbox_success*** 表示成功，返回 ***ppbox_would_block*** 表示这次没有读到数据，返回 ***ppbox_stream_end*** 表示视频流已经结束。其他见错误码表。可以通过 ***PPBOX_GetLastErrorMsg*** 获取进一步的错误描述。

说明：

返回的 ***sample.buffer*** 指向的内存是由函数内部申请，有效期保持到下次调用 ***PPBOX_ReadSampleEx2*** 或者调用 ***PPBOX_Close*** 前。

Sample 的定义：

struct PPBOX_SampleEx2
{
PP_uint32 stream_index;
PP_uint64 start_time;
PP_uint32 buffer_length;
PP_uint32 duration;
PP_uint32 desc_index;
PP_uint64 decode_time;
PP_uint32 composite_time_delta;
PP_bool is_sync;
PP_bool is_discontinuity;
PP_uchar const * buffer;
};

其中：

stream_index 流的编号
start_time 这个 Sample 对应的时间戳（单位：微妙）

buffer_length Sample 的数据长度
duration Sample 的播放时间
desc_index （没有使用）
decode_time （没有使用）
composite_time_delta （没有使用）
buffer Sample 的数据
is_sync 是否是关键帧
is_discontinuity 该位标识此帧与上一帧之间是否是连续的 sample; 如果该 sample 是 seek 后的 sample 或者前面丢失了 sample,则该位置 1，否则为 0。

4.3.7 获得播放状态

int32 PPBOX_GetPlayMsg(PPBOX_PlayStatistic* stat);
参数：
stat 读取后的结果写入这个结构体中
返回值：错误码
返回 ***ppbox_success*** 表示成功，返回 ***ppbox_would_block*** 表示调用者可以暂停一段时间，其他见错误码表。可以通过 ***PPBOX_GetLastErrorMsg*** 获取进一步的错误描述。
播放状态描述结构：

```
struct PPBOX_PlayStatistic {
    PP_uint32 length;
    PP_int32 play_status;
    PP_uint32 buffering_present;
    PP_uint32 buffer_time;
}
```

其中：
length 本结构体的长度
play_status 播放状态：0-未启动、1-playing、2-buffering、3-Pausing
buffering_present 播放缓冲百分比 10 表示 10%
buffer_time 下载缓冲区中的数据可以支持播放的时间（单位：毫秒）
播放状态：
0 未启动，没有成功 open
1 正常播放，***buffer_time*** 已经超过可接受的值（由 ***PPBOX_SetPlayBufferTime*** 过设置）
2 缓冲，***buffer_time*** 没有超过可接受的值（由 ***PPBOX_SetPlayBufferTime*** 过设置）
3 暂停，手动暂停了

4.3.8 获取下载统计

PP_int32 PPBOX_GetDownMsg(PPBOX_DownloadMsg* stat)
参数：

stat 读取后的结果写入这个结构体中

返回值：错误码

返回***ppbox_success***表示成功，其他见错误码表。可以通过***PPBOX_GetLastErrorMsg***获取进一步的错误描述。

下载统计描述结构：

```
struct PPBOX_DownloadMsg{
    PP_uint32   length;                     //本结构体的长度
    PP_uint32   start_time;                 //开始时刻
    PP_uint32   total_download_bytes;       //总共下载的字节数
    PP_uint32   total_upload_bytes;         //总共上传时字节数
    PP_uint32   http_download_bytes;        // Http 下载的字节数
    PP_uint32   http_downloader_count;      // Http 下载个数
    PP_uint32   p2p_download_bytes;         // P2P 下载的字节数
    PP_uint32   p2p_upload_bytes;           // P2P 上传的字节数
    PP_uint32   p2p_downloader_count;       // P2P 下载个数
    PP_uint32   p2p_downloader_count_ext;   // 候补 P2P 下载的资源个数
    PP_uint32   total_upload_cache_request_count;   // 总共的上传 Cache 请求数
    PP_uint32   total_upload_cache_hit_count;       // 总共的上传 Cache 命中数
    PP_uint32   download_duration_in_sec; // 下载总共持续时长(秒)
    PP_uint32   local_peer_version;         // 自己内核版本号
};
```

4.3.9 获取下载速度

PP_int32 PPBOX_GetDownSedMsg(PPBOX_DownloadSpeedMsg* stat)

参数：

stat 读取后的结果写入这个结构体中

返回值：错误码

返回***ppbox_success***表示成功，其他见错误码表。可以通过***PPBOX_GetLastErrorMsg***获取进一步的错误描述。

下载数据描述结构：

```
struct PPBOX_DownloadSpeedMsg{
    PP_uint32   length;                     //本结构体的长度
    PP_uint32   now_download_speed;         // 当前下载速度 <5s 统计>
    PP_uint32   now_upload_speed;           // 当前上传速度 <5s 统计>
    PP_uint32   minute_download_speed;      // 最近一分钟平均下载速度 <60s 统计>
    PP_uint32   minute_upload_speed;        // 最近一分钟平均上传速度 <60s 统计>
    PP_uint32   avg_download_speed;         // 历史平均下载速度
    PP_uint32   avg_upload_speed;           // 历史平均上传速度
    PP_uint32   recent_download_speed;      // 当前下载速度 <20s 统计>
    PP_uint32   recent_upload_speed;        // 当前上传速度 <20s 统计>
    PP_uint32   second_download_speed;      // 当前 1s 的下载速度
    PP_uint32   second_upload_speed;        // 当前 1s 的上传速度
```

};

注：以上所有数值的单位都是字节

5 编解码格式

类型	子类型	格式	配置格式	帧数据格式
视频(1)	AVC(1)	AVC Packet(1)	AVCDecoderConfigurationRecord	Nalu Packet
		AVC Stream(2)	SPS PPS	Nalu Stream
音频(2)	AAC(2)	AAC(3)	AAC Setup Data	MDCT
	MP3(3)	MP3(4)	无	MP3
	WMA(4)	WMA(5)	无	WMA

注：括号中的值对应 ***PPBOX_StreamInfo（Ex）***中的类型值

5.1 AVC Packet

对于 AVC 视频（也就是 H264），并且格式类型为 Packet 格式时，配置数据为 ***AVCDecoderConfigurationRecord*** 结构，帧的格式的 Nalu Packet 格式。

AVCDecoderConfigurationRecord 结构定义如下：

```
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< numOfSequenceParameterSet s; i++) {     //SPS 信息
        unsigned int(16) sequenceParameterSetLength ;
        bit(8*sequenceParameterSetLength) sequenceParameterSetNALUnit;
    }
    unsigned int(8) numOfPictureParameterSets;
    for (i=0; i< numOfPictureParameterSets; i++) {  //PPS 信息
        unsigned int(16) pictureParameterSetLength;
        bit(8*pictureParameterSetLength) pictureParameterSetNALUnit;
    }
}
```

5.2 AVC Stream

对于 AVC 视频（也就是 H264），并且格式类型为 Stream 格式时，配置数据为 SPS PPS，帧的格式的 Nalu Stream 格式。

5.3 AAC

对于 AAC，其配置数据为 AAC Setup Data：

5 bits: object type
if (object type == 31)
6 bits: object type - 32
4 bits: frequency index
if (frequency index == 15)
24 bits: frequency
4 bits: channel configuration
1 bit: frame length flag
1 bit: dependsOnCoreCoder
1 bit: extensionFlag

需要拼装 ADTS 头的时候：

- 12 bits of syncword 0xFFF, all bits must be 1
- 1 bit of field ID. 0 for MPEG-4, 1 for MPEG-2
- 2 bits of MPEG layer. If you send AAC in MPEG-TS, set to 0
- 1 bit of protection absense. Warning, set to 1 if there is no CRC and 0 if there is CRC
- 2 bits of profile code. The MPEG-4 Audio Object Type minus 1
- 4 bits of sample rate code. MPEG-4 Sampling Frequency Index (15 is not allowed)
- 1 bit of private stream. Set to 0
- 3 bits of channels code. MPEG-4 Channel Configuration (in the case of 0, the channel configuration is sent via an inband PCE)
- 1 bit of originality. Set to 0
- 1 bit of home. Set to 0
- 1 bit of copyrighted stream. Ignore it on read and set to 0
- 1 bit of copyright start. Set to 0
- 13 bits of frame length. This value must include 7 or 9 bytes of header length: FrameLength = (ProtectionAbsent == 1 ? 7 : 9) + size(AAC Frame)
- 11 bits of adts_buffer_fullness 0x7FF indicates VBR
- 2 bits of frames count in one packet. Set to 0

其中：我们的音频默认的字段有：

- NO CRC

- VBR

可以从 AAC 配置结构里读取的字段有：

- profile code = object type – 1
- frequency index
- channel configuration

6 错误码表

枚举类型	描述
ppbox_success	正确，无错误
ppbox_not_start	P2P引擎没有打开
ppbox_already_start	P2P引擎已经打开
ppbox_not_open	视频还没有打开，需要调用Open
ppbox_already_open	视频已经打开
ppbox_operation_canceled	异步操作被取消
ppbox_would_block	操作不能立即完成
ppbox_stream_end	用于ReadSample的时候，表示视频流已经结束
ppbox_logic_error	逻辑错误，程序bug，请报告PPBox
ppbox_network_error	网络发生错误
ppbox_demux_error	Demux错误
ppbox_certify_error	认证错误
ppbox_other_error	其他错误