音视频库 API 手册

2015.11.30

1.1 网络客户端函数

nav_open

建立到服务器的连接

```
typedef struct {
    uint64_t uid;
    const uint8_t* info;
    uint32_t bytes;
    uint32_t isp;
    uint32_t ms_wait;
    notify_fptr fptr;
    uint32_t dont_play;
    uint32_t dont_upload;
    uint32_t login_level;
} nav_open_args;

void* nav_open(nav_open_args* args)
```

nav_open 创建从客户端到服务器的 tcp 连接， 函数将阻塞直至 tcp 三次握手成功或者因某些原因失败， 比如网络不通或者超过最大的等待时间等， 成功时返回一个非空值， 失败返回空。 函数成功返回，连接可能会因各种情况导致中断，内部存在重连逻辑，无需调用者太多关注，连接的中断、重连事件会通过 fptr 提供的回调函数通知用户， 注意重连尝试间隔时间为几十个毫秒，没有某种形式的指数退让，因此在长时间网络中断情况下，重连会频繁发生而导致高 cpu 占有率。用户应该注意这个情况，并在长时间无法连接成功后调用 nav_close 函数来终止重连。

uid

客户端用户 id

info

服务器的地址信息

bytes

info 的字节数

isp

调用者的网络提供商类型，假如存在多个服务器可供选择，程序内部会使用优先连接位于相同 isp 提供商机房内的服务器，以避免网络连接跨服务商。错误的 isp 值只会影响服务器的选择，不会导致严重后果。

ms_wait

同服务器创建网络连接时的最大的等待时间， 单位为毫秒

fptr

```
typedef struct {
    int (*fptr) (void* ctx, uint32_t notify, void* any);
    void* ctx;
} notify_fptr;
```

网络事件的回调， ctx 为调用者提供的任意值，回调时， 该值被拷贝到回调函数的第一个参数， notify 为事件类型, any 为事件参数， 具体见后。

dont_play

收到音频后， 是否直接播放， 可取值 0、1。 0 直接播放， 1 将会将收到的音频丢掉， 直到外部通过 audio_play_pause 允许为止

dont_upload

启动音频、视频后是否直接发送到服务器， 可取值 0、1。0 会直接发送， 1 不会发送直到外部通过 audio_capture_pause 允许为止

login_level

是否激活 prejoin 流程， 可取值 0、2。 0 会在正式 join 频道前首先发起 prejoin， 2 会跳过 prejoin 直接发起正式 join。一般来说，prejoin 可提供电话的主叫方等待被叫接听的流程， 直接 join 则支持聊天室等无需振铃的场景

nav_close

关闭到服务器的连接

void nav_close(void* any, uint32_t code)

nav_close 发起关闭网络连接的过程，这应该是最后一个被调用的 nav 函数。调用本函数不要求其他正在使用 any 的函数已经返回，但在函数返回后 any 会被标记为销毁，不允许继续使用。本函数返回不意味着网络连接已经关闭，网络连接会在 nav_close 被调用后某个时间关闭， 可能发生在 nav_close 返回后， 也可能在返回前。

any， nav_open 的返回值

code，退出码， 这个值将在关闭网络连接前发送给服务器，以指示退出原因

nav_ioctl

在已经建立的网络连接上进行各种操作

int32_t nav_ioctl(void* any, int32_t cmd, void* param)

nav_ioctl 在已经建立的连接上进行启动音频采集，播放网络媒体等。操作的类型由 cmd 指

定， cmd 特定的参数为 param。any 为 nav_open 返回的非空值，代表网络连接。操作成功返回 0，失败返回 -1

audio_capture_start

启动音频采集，若 dont_upload 设置为 0，采集到的音频会被传输到网络，否则音频会被简单丢弃。采集的音频为采样率 8000 的单声道 pcm，编码为 silk 码流。param 为指向以下结构的指针:

```
typedef struct {
    enum aec_t aec;
    intptr_t effect;
} audio_param;
```

aec，回音消除的引擎，可供选择的引擎定义于 audioctl.h，注意 android 不能使用 ios 类型
effect，音效，可供选择的音效定义于 efftype.h

audio_capture_stop

关闭音频采集，param 没有被使用，这个操作永远不会失败

audio_capture_pause

param 为一个 int32_t 类型的指针，指向 0 代表 resume, 1 代表 pause。这个操作并不会真的 pause 音频采集，实际效果是改变 dont_upload 的值，以允许或者禁止音频流传输到网络。这个操作永远不会失败

audio_play_pause

param 为一个 int32_t 类型的指针，指向 0 代表 resume, 1 代表 pause。这个操作实际是改变 dont_play 的值，以允许或者禁止从网络接收到的音频流被播放。这个操作永远不会失败

audio_enable_dtx

音频传输是否启用不连续发送，启用 dtx 有助于节省网络流量，但由于静音检测算法存在误差，可能会导致有效的声音信号丢失，引起卡顿的感觉，param 为一个 int32_t 类型的指针，指向 0 代表不启用，1 代表启用

video_capture_start

启动视频采集，若 dont_upload 设置为 0，采集到的音频会被传输到网络，否则会被简单丢弃。采集的视频目前支持的分辨率为 352x288, 640x480, 1080x720，长宽可互调，即支持 288x352 等分辨率。编码为 h264 码流。param 为指向以下结构的指针:

```
typedef struct {
    enum camera_position_t campos;
    intptr_t feedback;
```

```
        intptr_t width, height;
    } video_param;
```

campos，摄像头的位置，前置或者后置，这个设置可在采集开始后通过 video_use_camera 来更改
feedback，采集的视频是否需要回显，0 不回显，1 回显
width,height 图画分辨率

注意在 android 系统中，不同手机支持不同的分辨率，当选择的分辨率不被系统支持时，则会选择某个长宽都比要求的大的视频，而后在编码时选择其中一部分进行编码。最终效果是用户的分辨率要求总会被满足。大分辨率图像的一部分与小分辨率的图像整幅虽然分辨率相同，但小分辨率图像显示的范围可能会更大，而大分辨率会更清楚。

vidio_capture_stop

关闭视频采集，param 没有被使用，这个操作永远不会失败

video_use_camera

切换前后摄像头，param 为指向 enum camera_position_t 类型的指针

1.2 网络回调函数

```
int (*fptr) (void* ctx, uint32_t notify, void* any)
```

网络回调发生在网络发生某些事件时，时间不可预测，回调可能会发生在任意线程，包括调用者自己的线程。目前实现可保证回调是顺序发生的， 即不会有两个回调同时发生，另外可保证在 nav_close 返回后，不会再有回调被调用。回调大多数时间发生在非常关键的线程中，意味着回调函数的实现不能占用时间太长而导致关键线程阻塞，另外也不能在回调函数中调用 nav 系列函数，否则可能会引起死锁。

ctx 用户设置的不透明值，内部不会对它做任何处理，只会在回调时将其拷贝到第一个回调参数。notify 为发生的事件类型，目前分为如下几种：

notify_login

在未启用 prejoin 情况下，只会有 join 成功后会被触发。在启用 prejoin 情况下用户 prejoin 或者 join 成功后各会触发一次，prejoin 只会触发一次，若之后因为网络重连等原因而引起重新登录，只会正式 join,不会 prejoin。实现保证在 disconnect 事件发生前，只会发生一次正式 join 触发的 notify_login 回调。

any 为指向 uint64_t 类型的指针，所指向的值为 0 时代表 prejoin 引起的回调，1 代表正式 join

notify_disconnected

网络连接中断后，该消息被触发。实现保证在收到另一个 notify_login 前，只有一个 notify_disconnected 回调被调用。any 未使用。
注意 nav_close 被调用后，notify_disconnected 有可能被调用，也可能不会被调用，这是为了保证 nav_close 返回后不会有回调被调用，当在 nav_close 返回前网络关闭，notify_disconnected 会被触发，返回后网络才发生关闭，则不会触发

notify_user_stream

网络来自某个源媒体流到来或者中断后被触发，实现保证某个源不会在收到流到达通知前收到流中断的消息，在收到流中断的消息前，只会收到唯一一个流到达的消息，若流在中断后重新到达，保证在收到重新到达的消息前，只会收到一个流中断的消息。
any 为指向以下结构的指针：

```
typedef struct {
    uint64_t uid;
    uint32_t aud0_vid1;
    uint32_t on;
} usr_stream_t;
```

uid 为源的 ID
aud0_vid1 如名称所暗示的,取 0 时代表音频，1 时代表视频
on 取 0 时代表流中断，1 时代表流到达

notify_user_count

第一次连接服务器或者重连成功收到 join 成功消息后，实现会向服务器取一次用户列表，因为协议只有在用户状态发生变化时才会主动通知，在原本已经在频道里的用户的状态不变化的情况下，连接到服务器的客户端不会得知任何状态消息来创建频道用户的列表信息，因此主动取一次列表是必要的，一旦获取到列表后，随后的用户状态变化可以通过用户状态通知来判断频道人数的增加或减少。

any 为指向 uint64_t 类型的指针，所指向的值为频道用户总数，包含自己

notify_user_stat

频道内用户的状态变化，不包含自己的状态变化，自己状态变化应该通过 notify_login 等获得，any 为指向 uint64_t 类型的指针

```
typedef struct {
    uint64_t uid;
    uint64_t stat;
```

```
    uint64_t reason;
} usr_stat_t;
```

uid 为用户 id
stat 为用户状态， 用户状态的可能值及对应含义请咨询服务器
reason nav_close 中的第二个参数是这个 reason 的一个来源，其他可能值及含义请咨询服务器

notify_statistic

某路媒体流的每 10 秒一次的统计，any 为指向以下结构的指针

```
typedef struct {
    uint64_t uid;
    uint32_t sid;
    uint32_t aud0_vid1;
    uintptr_t lost;
    uintptr_t recv;
} media_stat_t;
```

uid 为用户 id
sid 为频道号
aud0_vid1 如名称所暗示的,取 0 时代表音频，1 时代表视频
lost 代表在过去的 10 秒窗口里，有多少媒体包由于网络因素丢失
recv 代表在过去的 10 秒窗口里，收到了多少有效的媒体包

notify_io_stat

网络 udp 传输的媒体统计，注意是只包含媒体的，不包含任何其他消息，统计包含协议头，any 为指向以下结构的指针

```
typedef struct {
    uint64_t nb0;
    uint64_t nb1;
} iostat_t;
```

nb0 代表 output 的字节数
nb1 代表 input 的字节数

1.3 视频渲染区域管理函数

视频渲染支持渲染区域的概念，渲染区域指的是视图中的一个矩形区域，一路视频到来，实现会调用用户提供的函数 open_area，给该路视频分配一个渲染区域，该路视频会在这个矩形内进行渲染，而不会超出范围外。

渲染区域支持深度，以更方便的实现广泛应用的画中画模式，但只支持 0、1 两种深度，因此，没有直接的办法实现画中画中画，但使用者可以通过多视图方式来实现多级画中画效果。iOS 版可以支持一个视图中超过 2 个的渲染区域，这是因为苹果公司并不支持标准 opengles，行为与 opengles 有所不同，而 android 下支持标准的 opengles 标准，不保证提交一次渲染到窗口之后依然保留深度缓冲信息。Android 多渲染区域可以通过多视图方式来实现。

渲染区域结构定义如下

```
typedef struct {
    uint64_t identy;
    uintptr_t z, w, h;
    uintptr_t x, y, angle;
    void* egl;

    lock_t lck;
    intptr_t idx;
} area_t;
```

使用者只需了解其中部分成员，其余成员使用者不能修改。
identy 与渲染区域绑定的视频流的 id
x,y,z,w,h 标识了渲染区域矩形在三维坐标系里的位置
egl 为视图所提供的一个不透明值，使用者无需关心含义，但需要在 open_area 函数中将该值赋给这个成员

open_area

这是使用者必须提供实现的一个函数，功能是给视频流分配渲染区域。注意内部有一个默认实现，但这个默认实现不做任何事情，只是为了避免链接错误，因此使用者若不提供这个函数的实现，除了运行时不工作之外，编译链接阶段将得不到任何错误提示。该默认实现是弱符号的，因此使用者必须提供强符号的实现来避免链接器最终链接到默认实现。

```
intptr_t open_area(area_t* area) need_implemented
```

任何实现必须给 area 结构体的下 x,y,z,w,h,egl 赋值，其余的成员必须保持不动。
w,h 作为输入参数，含义为视频的宽与高，实现者可以利用此信息来恰当的分配渲染区域，以避免视频被拉伸变形。当函数返回时,w,h 必须为渲染区域的长宽，以像素数为单位，x,y,z 必须为渲染区左下角在三维坐标系的坐标。egl 必须被赋值为视图所提供的不透明值。若不允许显示该路视频或者由于某原因分配失败，返回-1，否则返回 0。返回

-1 只意味这当前帧不被显示，等下一帧到来的时候，open_area 依旧会被调用。若返回 0，则下一帧直接使用之前分配的渲染区域，不会再次调用 open_area

reopen_area

这是使用者必须提供实现的一个函数，这个函数会在视图大小变化时被调用，功能是给视频流重新分配渲染区域以适应新的视图尺寸。注意内部有一个默认实现，但这个默认实现不做任何事情，只是为了避免链接错误，因此使用者若不提供这个函数的实现，除了运行时不工作之外，编译链接阶段将得不到任何错误提示。该默认实现是弱符号的，因此使用者必须提供强符号的实现来避免链接器最终链接到默认实现。

```
void reopen_area(area_t* area, uintptr_t width, uintptr_t height) need_implemented
```

任何实现必须给 area 结构体的 x,y,z,w,h 赋值，其余的成员必须保持不动，包含 egl 成员。
w,h 作为输入参数，含义为视频的宽与高，width 为视图的宽度，height 为视图的高度，实现者可以利用此信息来恰当的分配渲染区域，以避免视频被拉伸变形。当函数返回时,w,h 必须为渲染区域的长宽，以像素数为单位，x,y,z 必须为渲染区左下角在三维坐标系的坐标。

do_close_area

关闭由 open_area 分配的渲染区域。

```
void do_close_area(area_t* area)
```

本函数做了真正的关闭工作，应该在 close_area 中被调用。这个函数永远不会失败。

close_area

这是使用者必须提供实现的一个函数，功能是关闭由 open_area 分配的渲染区域。注意内部有一个默认实现，但这个默认实现不做任何事情，只是为了避免链接错误，因此使用者若不提供这个函数的实现，除了运行时不工作之外，编译链接阶段将得不到任何错误提示。该默认实现是弱符号的，因此使用者必须提供强符号的实现来避免链接器最终链接到默认实现。

```
intptr_t close_area(area_t* area) need_implemented
```

这个函数应该调用 do_close_area 来关闭渲染区域，而后返回 0 或 1。返回 0 意味着关闭渲染区域后，该区域以前渲染的图像人就会残留在视图上，但该残留图像不再维护，也就是说当需要窗口重绘时，比如窗口被其他窗口挡住后重新显示出来，该图像很有可能会消失。返回 1 意味着该区域以前渲染的图像会立即消失。

area_invalid

刷新视图

```
void area_invalid(area_t* area, area_t* templ)
```

引起一次视图刷新操作。用户主动修改渲染区域后（非因为视图大小变化而引起的被动修改）需要调用这个函数使修改生效。若存在已经关闭的渲染区域的残留图像，该图像也会消失。

1.4 iOS 平台函数

lock_audio_dev

抢占音频设备，iOS 平台特有，Android 下音频设备层代码在 java 中，并以源代码形式提供，类似问题请自己在代码中做修改或增加逻辑来实现。

```
int lock_audio_dev()
```

iOS 下音频播放设备可能在使用中被其他应用抢掉，比如在播放过程中来了电话，电话就会抢占设备，导致原来播放的设备停止播放。当这类情况发生并被侦测到时，会有回调函数被触发来通知。用户可以重新尝试立即或等合适时机电话挂断后抢回来。

实现内部也对这类情况做了监测，在大多数情况下，无须任何工作也可以实现与其他程序的合作，但内部监测是基于事件的，只有某些事件发生时才会做这些事情，而且也不保证每次都成功。

当函数调用成功时返回 0，失败返回-1。冗余的调用不会引起问题，即允许在占用设备的情况下调用本函数，这种情况下函数不会做任何事情。

switch_to_speaker

切换音频播放设备到扬声器还是听筒。iOS 平台特有，Android 下音频设备层代码在 java 中，并以源代码形式提供，类似问题请自己在代码中做修改或增加逻辑来实现。

```
int switch_to_speaker(int one_or_zero, int retry_if_failed)
```

one_or_zero

可取的值 0 或 1，0 切换到听筒，1 切换到扬声器

retry_if_failed

可取的值 0 或 1，0 返回值永远不会返回失败，操作失败时内部会记住并在某些事件发生时重试，1 在操作失败时会返回错误，并且不会重试。

注意重试不会立即或持续进行，因此 retry_if_failed 其实并没有太大意义。以后可能会

取消这个参数。使用者建议以 0 作为实参来调用。

switch_to_category

切换音频设备模式。iOS 平台特有，Android 下音频设备层代码在 java 中，并以源代码形式提供，类似问题请自己在代码中做修改或增加逻辑来实现。

```
int switch_to_category(int category, int retry_if_failed)
```

category

可取的值定义于苹果系统头文件，就是 kAudioSessionCategory_AmbientSound 这些。使用者不应该以 kAudioSessionCategory_MediaPlayback 来调用该函数，这个与默认状态 kAudioSessionCategory_PlayAndRecord 并不冲突，因此被内部用来做了一个标识。使用者成功调用该函数后，内部将不再根据设备变化情况修改 category，直到用户通过以 kAudioSessionCategory_PlayAndRecord 为参数调用本函数来放弃控制权。

retry_if_failed

可取的值 0 或 1，0 返回值永远不会返回失败，操作失败时内部会记住并在某些事件发生时重试，1 在操作失败时会返回错误，并且不会重试。

注意重试不会立即或持续进行，因此 retry_if_failed 其实并没有太大意义。以后可能会取消这个参数。使用者建议以 0 作为实参来调用。

audio_notify

这是使用者应该提供实现的一个函数，iOS 音频设备事件通知将通过这个函数进行回调通知，不提供该函数的一个实现将不会收到此类通知。注意内部有一个默认实现，但这个默认实现不做任何事情，只是为了避免链接错误，因此使用者若不提供这个函数的实现，除了运行时不工作之外，编译链接阶段将得不到任何错误提示。该默认实现是弱符号的，因此使用者必须提供强符号的实现来避免链接器最终链接到默认实现。

```
void audio_notify(void* any) need_implemented
```

any 可能是如下值：

in_killed

当回音消除引擎使用 ios 时关闭音频播放时可能会出现，含义是同时进行的音频采集也被错误的结束了，这是因为 ios 回音消除引擎会底层要求使用同一个 audiounit 进行采集与播放，因此必须同时停止。当同时停止后，单独打开播放时失败时会导致该错误

out_killed

当回音消除引擎使用 ios 时关闭音频采集时可能会出现，含义是同时进行的音频播放也被错误的结束了，这是因为 ios 回音消除引擎会底层要求使用同一个 audiounit

进行采集与播放，因此必须同时停止。当同时停止后，单独打开采集时失败时会导致该错误

指向如下结构的一个指针，该结构反应了当前的设备状态，当且仅当至少一个状态发生变化时，回调函数才可能会被触发

```
typedef struct {
    int category;
    int speaker;
    int headset;
    int microphone;
    int interrupted;
} audio_state;
```

category 当前 catergory 值
speaker 当前音频播放是扬声器还是听筒
headset 当前有没有有效的耳机设备
microphone 当前有没有有效的麦克风设备
interrupted 当前音频设备有没有被抢占

注意由于 iOS 系统自身问题，有时候设备被其他程序抢占，但被抢占的程序无法获得任何通知，因此 interrupted 回调消息也有可能该被回调却没有出现。

1.5 视频渲染视图

视频渲染视图封装了底层绘图，使用者只需要遵循 1.3 渲染区域管理函数要求填充数据结构，就可以实现在视频渲染视图上的绘制操作。视频绘制视图要求使用者将视图添加到 GUI 窗口系统中的合适位置

1.5.1 GLView

GLView 是 iOS 下的视图类，继承于标准的 UIView:

- (id) initWithFrame:(CGRect)frame r:(float)r g:(float)g b:(float)b a:(float)a
创建视图，frame 为视图大小，r,g,b 为视图背景颜色，a 为透明度

- (void) suspend:(intptr_t) yes
平台要求当程序进入后台后，所有的绘图必须停止否则会引起程序崩溃。因此使用者有责任在程序即将进入后台时调用这个方法来暂停绘图，或者重新进入前台后重新启动绘图。参数 yes 为 1 时暂停绘图，为 0 是重启绘图

- (void *) egl_get
渲染视图提供的不透明值，需要在 open_area 中赋值给 area_t 中的 egl 字段。

1.5.2 surfacepool

surfacepool 是 android 下的视图类，混合了视图及渲染区域管理功能，以源文件形式提供，使用者可以自由修改该文件来实现需要的功能，只要满足几个特定的语意要求，surfacepool.java 实现本身是一个例子，可以用来参考。

public static pool_item add_surface(Surface sfc, int r, int g, int b, int width, int height)
将一个 android 的 Surface 对象加入 surfacepool 管理，width, height 为视图大小，r,g,b 为视图背景颜色，add_surface 应该在内部调用 jni 函数 surface_register，这个方法不会被非使用者调用，因此可以随意更改函数的签名。

static void surface_resize(Object handle, int width, int height)
Surface 尺寸发生变化后，通过这个函数来通知绘图层，内部应该调用 jni 函数 surface_resize 来做真正的通知。jni 实现会在合适的时机调用 area_reopen 来要求使用者重新分配渲染区域，这个方法不会被非使用者调用，因此可以随意更改函数的签名。

public static void remove_surface(Surface sfc)
将 Surface 从 surfacepool 中删除，不再受 surfacepool 的管理。所有在这个 Surface 上分配的渲染区域将都被 close。remove_surface 内部应该调用 jni 函数 surface_unreifster，这个方法不会被非使用者调用，因此可以随意更改函数的签名。

private static long[] area_open(long id, long w, long h)
在适当的时机，这个函数会被底层调用来分配某路视频的渲染区域，参考 1.3 视频区域管理函数。这个函数会被底层回调，不允许修改函数签名。

id，标识视频的一个 64 位数字，高 32 位代表频道号，本地采集的频道号为 0，低 32 位代表用户 uid，w,h 为视频原始宽高。返回值为一个长度为 6 的数组，前 5 个为渲染区域的三维坐标 x,y,z,w,h，x,y 对应渲染区域为左下角，z 只能取 0，1，z 值为 1 的图像若和为 0 的图像重叠，则 z 为 1 的图像会遮挡住为 0 的。数组最后一个元素为 surface_register 的返回值，注意在返回给 jni 前需要使用 handle_reference 增加其引用计数。

private static long[] area_reopen(long z, long w, long h, long width, long height)
当 Surface 尺寸变化后，在适当的时机，这个函数会被底层调用来重新分配某路视频的渲染区域，参考 1.3 视频区域管理函数。这个函数会被底层回调，不允许修改函数签名。

在 android 上，一个 Surface 只允许分配两个渲染区域。这里的 z 参数用来定位哪个渲染区域需要重新分配，若两个渲染区域存在在 area_open 时分配了相同的 z 值，那么将无法区分，未来的改进将在参数中增加视频 id，目前最好不要给两路视频分配相同的 z 除非永远不会调用 surface_resize.

private static int area_close(long id, long handle)
关闭所分配的渲染区域。在适当的时机，这个函数会被底层调用，参考 1.3 视频区域管理函数。这个函数会被底层回调，不允许修改函数签名。

id 视频流的 id

handle area_open 时在数组最后一个元素中返回的值。注意应该使用

handle_dereference 来释放这个 handle 的引用计数来抵消 area_open 时增加的引用

1.6 Android 平台 Java 接口

1.6.1 mmapi.java

mmapi 接口是 1.1 网络客户端函数的 jni 封装，详细信息请参考 1.1 节。

nav_open 与 nav_ctatus

参数与 1.1 节 nav_open 函数参数一一对应， 除了:

login_level 在内部被强制为 1，将来可将其开放到 java 层

notifys 由于 jni 特性，从 C 向 java 直接回调会存在 2 个问题，性能低和 C 线程需要转换成 java 线程，因此网络事件才用了另外一种方法，即在 jni 层存在一个事件缓冲区，回调的网络事件都会进入该缓冲区进行缓存，而后由 java 代码通过调用 nav_status 来主动读取事件。notifys 为这个缓冲区的最多可缓存的事件数，若长时间不读或事件发生的太过频发导致缓冲区溢出，那么最早发生的事件将丢失。使用者可将这个参数设置的足够大来防止这个情况的发生，比如 1 万个事件应该足够大能够避免 nav_status 调用间隔导致的事件丢失了。在实现中，每个事件占用 40 字节，因此 1 万个事件占用的内存为 400k，是可以接受的，每个事件的含义见 1.2 节网络回调函数，在 long 数组中的各个元素的含义请参考 mmv.java。

注意这要求 nav_status 调用间隔需要足够小，nav_status 被设计为阻塞调用，即若没有事件，该函数会一直阻塞下去直到有事件可以返回，如此可以避免无谓的周期性调用而导致的性能损失及调用不及时。nav_status 会尽量多的返回事件，通过同一个 long 数组。其中数组第一个元素为返回的事件数，剩下的每 5 个元素为一个事件，因此数组的长度永远是 1+5*事件数。当 nav_close 被调用后，nav_status 返回 null。

nav_compile

实际上是调试函数，在正式使用场景中，这个数据会从其他模块获取，而不是通过这个函数产生一个。

nav_ioctl

参考 1.1 节中的 nav_ioctl。函数参数为 long 数组，根据 cmd 不同不同元素具备不同含义。具体含义参考例子代码 mmv.java。

1.6.2 runtime.java

runtime 接口是管理不同库 (libmm,libmedia2,libomx) 之间的协作。同时由于 Android 的碎片化，不同厂商的手机对 jni 库支持各不同，比如 sony 手机通过 dlopen 形式加载动态库会报找不到库文件，这是因为库文件的安装目录不在库文件的搜索路径中。runtime 代码解决了这一个问题，但要求 libruntime.so 必须是第一个被加载的 jni 库文件。因此，所有其他的库文件加载 jni 时，要求使用 load 而不是 loadLibrary，并使用 libruntime.path 作为路径，

这是为了确保 libruntime.so 第一个被加载。

public static native void feature_mask(long key, long val)
由于 Android 碎片化，各个机型存在这样或那样的差异，可以使用这个方法根据机型来全局性禁止某些特性的激活以达到机型适配的目的，目前这个特性列表包含
omx_decode 是否激活硬解码
omx_encode 是否激活硬编码
eglbug_mask 在某些机型上，egl 存在 bug，通过设置这个 key 来通知底层使用一些肮脏的办法来绕过 bug(如果存在这个方法的话).

硬编码，硬解码默认都是激活状态，在激活状态下，内部实现存在一个探测逻辑来测试是否可以启用硬编硬解，具体方法就是尝试以一个 352x288 的尺寸的视频来打开硬编码器或硬解码器，并检查编解码器返回的内部数据，如果测试成功，则会在之后的编解码中使用硬编解码，否则会使用软编解码。因此，即便硬件编解码被启用，也有可能最终采用的是软件编解码。

1.6.2 media.java

底层通过调用这个文件里面的代码来做设备操作，一般不用关心这里面的代码，但由于 Android 平台的碎片化，有可能存在某些机型上会有某些特殊问题，此时可以修改这个文件里面的逻辑来做机型适配。

1.6.3 codec.java

底层通过调用这个文件里面的代码来做硬编码，由于碎片化，可能存在修改这个文件来做机型适配的需求。