4-打通UI到播放器核心的通道v1.1

腾讯课堂《FFmpeg/WebRTC/RTMP音视频流媒体高级开发》 https://ke.qq.com/course/468797?tuin=137bb271

1 消息队列

2-4-0voice_player -> 4-1-0voice_player

1.1 消息对象

C++ 复制代码

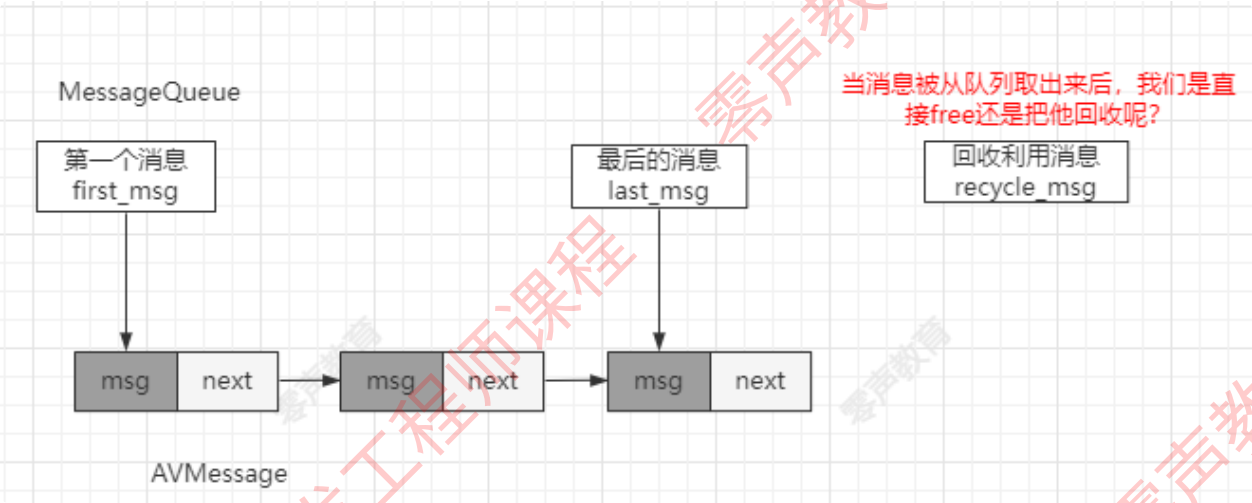
```cpp
typedef struct AVMessage {
    int what;                   // 消息类型
    int arg1;                   // 参数1
    int arg2;                   // 参数2
    void *obj;                  // 如果arg1 arg2还不够存储消息则使用该参数
    void (*free_l)(void *obj);  // obj的对象是分配的，这里要给出函数怎么释放
    struct AVMessage *next; // 下一个消息
} AVMessage;
```

1.2 消息队列对象

C++ 复制代码

```cpp
typedef struct MessageQueue {   // 消息队列
    AVMessage *first_msg, *last_msg;    // 消息头，消息尾部
    int nb_messages;    // 有多少个消息
    int abort_request;  // 请求终止消息队列
    SDL_mutex *mutex;   // 互斥量
    SDL_cond *cond;     // 条件变量
    AVMessage *recycle_msg; // 消息循环使用
    int recycle_count;  // 循环的次数，利用局部性原理
    int alloc_count;    // 分配的次数
} MessageQueue;
```

框架



1.3 消息队列api

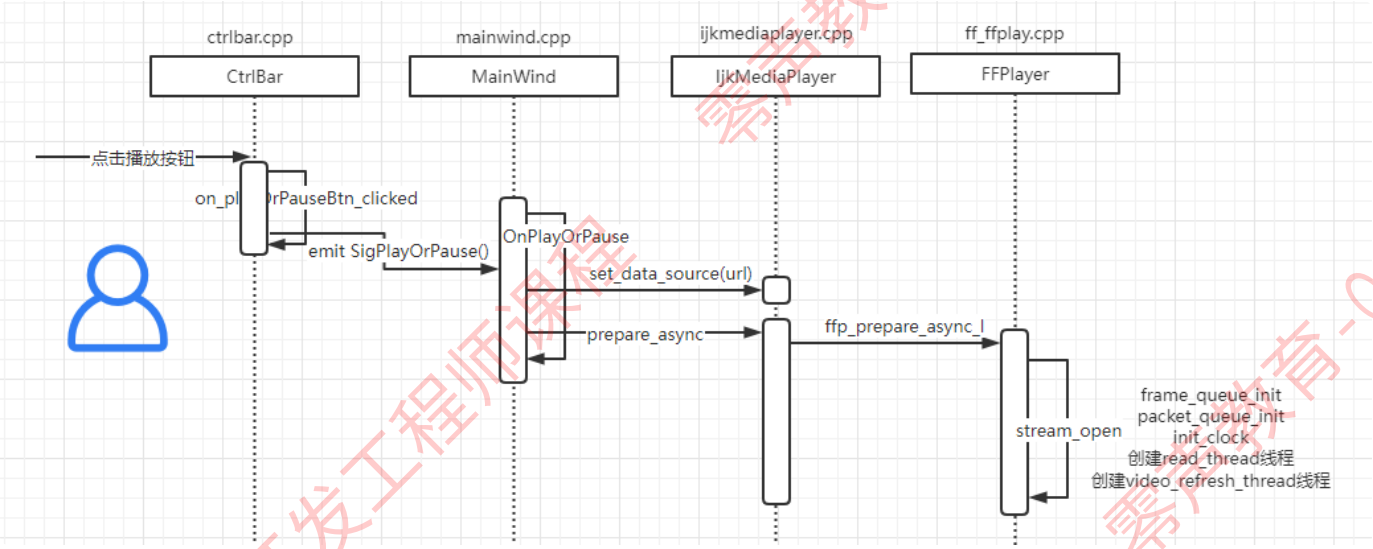
```
// 释放msg的obj资源
void msg_free_res(AVMessage *msg);
// 私有插入消息
int msg_queue_put_private(MessageQueue *q, AVMessage *msg);
//插入消息
int msg_queue_put(MessageQueue *q, AVMessage *msg);
// 初始化消息
void msg_init_msg(AVMessage *msg);
// 插入简单消息，只带消息类型，不带参数
void msg_queue_put_simple1(MessageQueue *q, int what);
// 插入简单消息，只带消息类型，只带1个参数
void msg_queue_put_simple2(MessageQueue *q, int what, int arg1);
// 插入简单消息，只带消息类型，带2个参数
void msg_queue_put_simple3(MessageQueue *q, int what, int arg1, int arg2)
// 释放msg的obj资源
void msg_obj_free_l(void *obj);
//插入消息，带消息类型，带2个参数，带obj
```

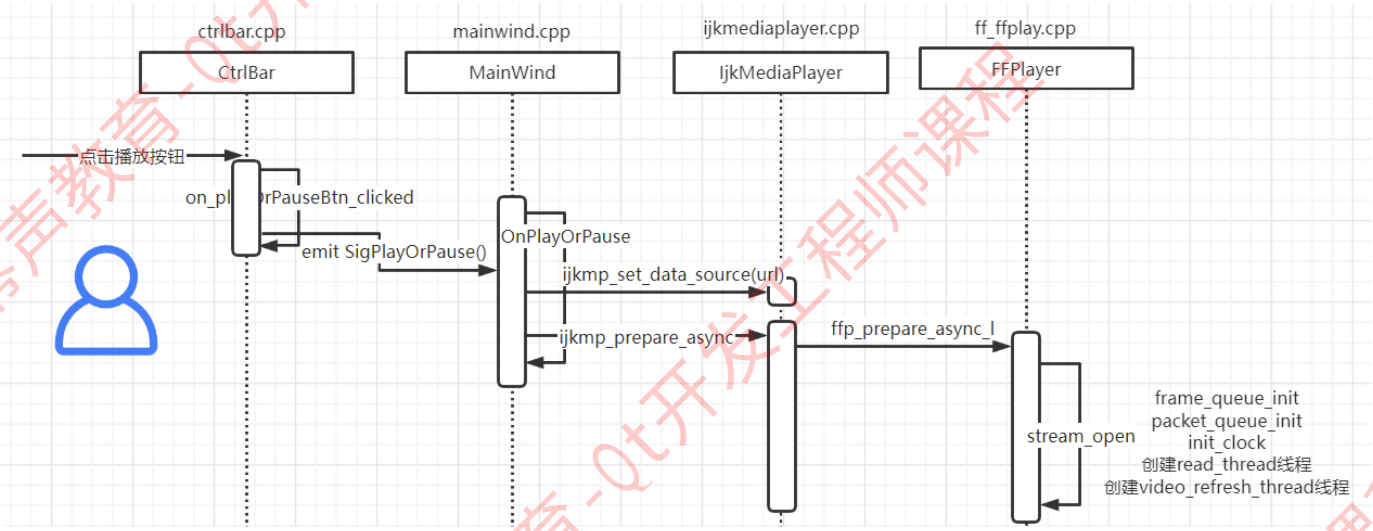
```
void msg_queue_put_simple4(MessageQueue *q, int what, int arg1, int arg2, void *obj, int obj_len);
// 消息队列初始化
void msg_queue_init(MessageQueue *q);
 // 消息队列flush，清空所有的消息
void msg_queue_flush(MessageQueue *q);
// 消息销毁
void msg_queue_destroy(MessageQueue *q);
// 消息队列终止
void msg_queue_abort(MessageQueue *q);
// 启用消息队列
void msg_queue_start(MessageQueue *q);
// 读取消息
/* return < 0 if aborted, 0 if no msg and > 0 if msg.  */
int msg_queue_get(MessageQueue *q, AVMessage *msg, int block);
// 消息删除 把队列里同一消息类型的消息全删除掉
void msg_queue_remove(MessageQueue *q, int what);
```

2 类名规划和接口设计

2.1 类名规划

IjkMediaPlayer FFplayer VideoState






```
int MainWind::message_loop(void *arg)
```

ui 和播放器核心直接的交互有以下几种方式：

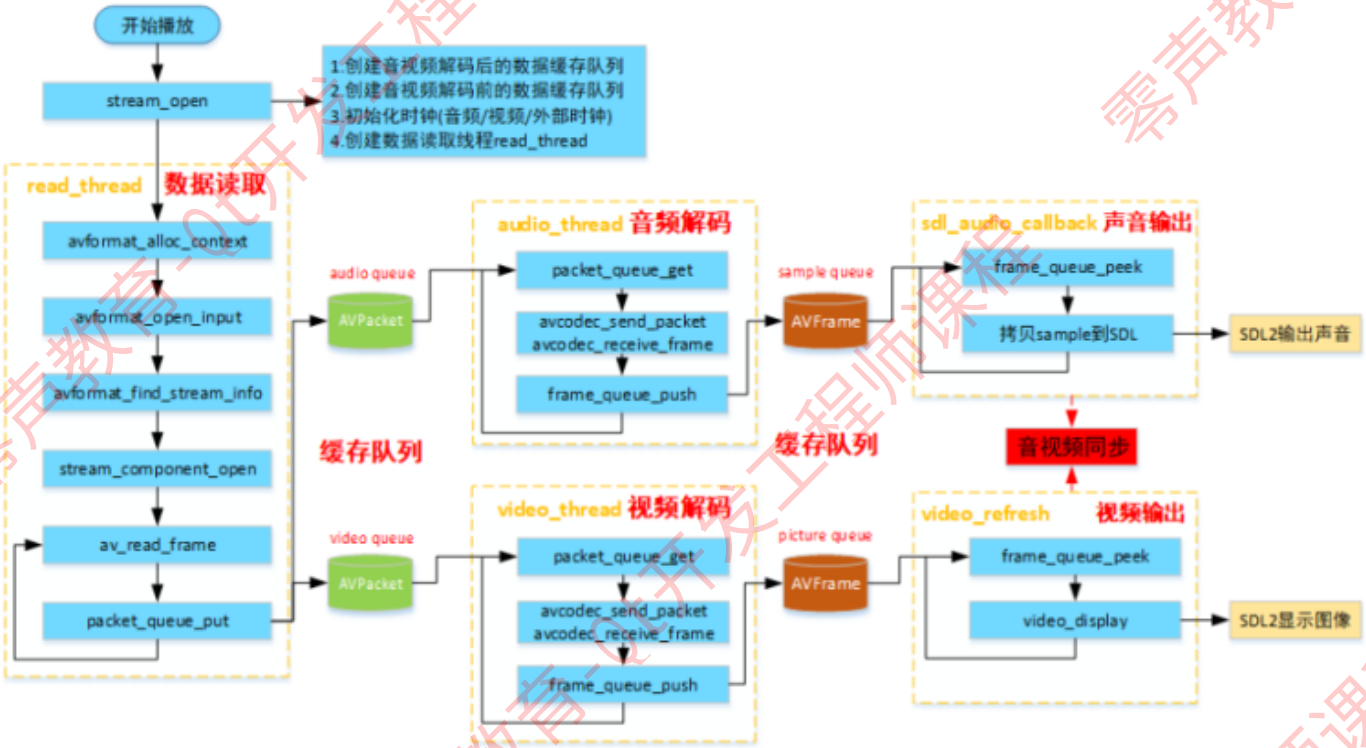
1. ui直接调用IjkMediaPlayer的接口
2. ui发送消息给消息循环线程，然后调用IjkMediaPlayer的接口
3. IjkMediaPlayer发消息给消息循环线程，线程调用ui的接口。

有部分消息是UI和IjkMediaPlayer都有处理，有部分消息只是IjkMediaPlayer要处理。比如：

- FFP_MSG_PREPARED： IjkMediaPlayer收到该消息后将mp_state_播放器状态设置为MP_STATE_PREPARED，而UI收到该消息后则知道资源已经准备好，可以调用start开始请求播放。

- FFP_REQ_START： IjkMediaPlayer收到该消息后调用ffp_start_l()触发播放，并将mp_state_设置为MP_STATE_STARTED。

2.2 接口函数



重点和难点接口解析

难点，以下五个接口的作用：

- ijkmp_create
- ijkmp_destroy
- ijkmp_prepare_async
- ijkmp_start
- ijkmp_stop

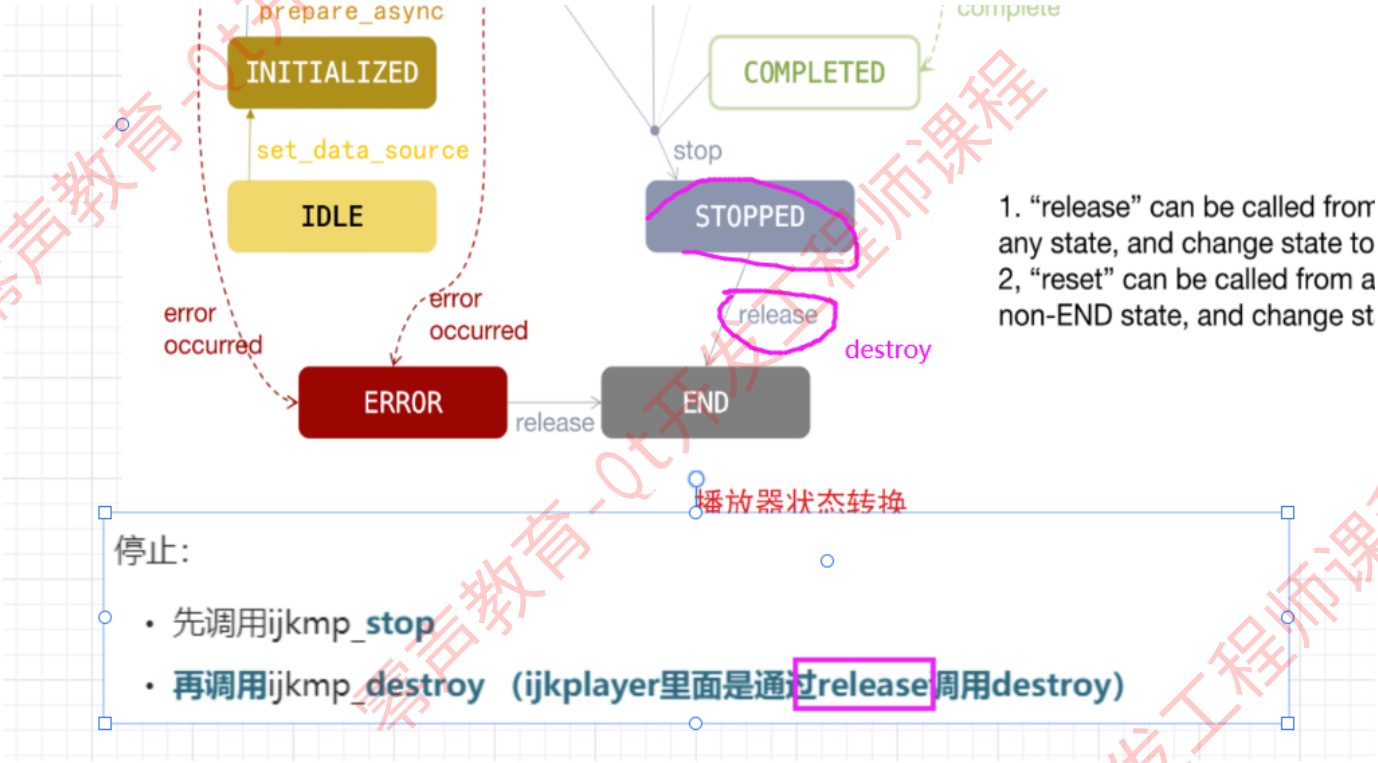
播放和停止

播放：

- ijkmp_create
- ijkmp_set_data_source
- ijkmp_prepare_async
- 然后等待消息MP_STATE_PREPARED再调用ijkmp_start启动播放。

停止：

- 先调用ijkmp_stop
- 再调用ijkmp_destroy （ijkplayer里面是通过release调用destroy）



下面详细说明这五个接口的具体作用：

- ijkmp_create
  - 创建IjkMediaPlayer
  - 创建FFPlayer
    - 初始化消息队列msg_queue_init
    - 初始化FFPlayer的成员变量
    - 刷新队列msg_queue_flush
  - 保存ui传入的回调msg_loop函数

    - 初始化mutex
    - 最终如果失败则调用destroy_p
- ijkmp_**destroy**
    - **停止msg_loop线程**
    - ffp_destroy_p**销毁**FFPlayer
        - stream_close
            - 请求abort_request
            - packet_queue_abort
            - stream_component_close 关闭audio、video、subtitle
            - avformat_close_input 关闭解复用器
            - packet_queue_destroy销毁audio、video、subtitle包队列
            - frame_queue_destory销毁audio、video、subtitle帧队列
        - 销毁消息队列msg_queue_destroy
    - 释放mutex
    - 释放自己 delete this
- ijkmp_**prepare_async**
    - **状态设置为**MP_STATE_ASYNC_PREPARING（正在准备），那什么时候状态转为MP_STATE_PREPARED(已经准备)。
        - 在FFPlayer的read_thread线程 解复用分析完码流情况、初始化完对应的解码器、音视频输出后，先调用 toggle_pause设置系统处于暂停播放状态，然后发送FFP_MSG_PREPARED。
        - IjkMediaPlayer收到FFP_MSG_PREPARED消息后，**把状态设置为MP_STATE_PREPARED**
        - UI收到FFP_MSG_PREPARED消息后，调用IjkMediaPlayer的start接口，开始正常播放。
    - 启动消息队列msg_queue_start
    - **创建msg_loop线程**
    - **调用FFplayer的**prepare_async_l
        - 调用stream_open
            - 分配VideoState
            - 保存filename到VideoState

- frame_queue_init初始化audio、video、subtitle帧队列
- packet_queue_init初始化audio、video、subtitle包队列
- 创建continue_read_thread解复用读取线程条件变量
- init_clock初始化audio、video、ext时钟
- 设置音量
- 创建视频刷新线程video_refresh_thread
- 创建数据读取线程read_thread
    - **一 准备工作**
        i. avformat_alloc_context 创建上下文
        ii. ic->interrupt_callback.callback = decode_interrupt_cb;
        iii. avformat_open_input打开媒体文件
        iv. avformat_find_stream_info 读取媒体文件的包获取更多的stream信息
        v. 检测是否指定播放起始时间，如果指定时间则seek到指定位置 avformat_seek_file
        vi. 查找查找AVStream，讲对应的index值记录到 st_index[AVMEDIA_TYPE_NB];
            1. 根据用户指定来查找流avformat_match_stream_specifier
            2. 使用av_find_best_stream查找流
        vii. 从待处理流中获取相关参数，设置显示窗口的宽度、高度及宽高比
        viii. stream_component_open打开音频、视频、字幕解码器，并创建相应的解码线程以及进行对应输出参数的初始化。

    二 For循环读取数据

        i. 检测是否退出
        ii. 检测是否暂停/继续
        iii. 检测是否需要seek
        iv. 检测video是否为attached_pic
        v. 检测队列是否已经有足够数据
        vi. 检测码流是否已经播放结束
            1. 是否循环播放
            2. 是否自动退出
        vii. 使用av_read_frame读取数据包
        viii. 检测数据是否读取完毕

ix. 检测是否在播放范围内

X. 到这步才将数据插入对应的队列

三 退出线程处理

i. 如果解复用器有打开则关闭avformat_close_input

ii. 消耗互斥量wait_mutex

- 保存filename

- ijkmp_**start**
  - ijkmp_start_l
    - 先检测当前的状态是否可以转为start，比如当前处于MP_STATE_IDLE、MP_STATE_INITIALIZED状态肯定是不能转为start状态的
    - 删除队列里的FFP_REQ_START消息，避免START消息请求重复
    - 删除队列里的FFP_REQ_PAUSE消息，因为现在是要START，所以如果队列里还有PAUSE消息，则队列里的PAUSE消息没有必要再被处理，因为接下来就要执行START。
    - 发送FFP_REQ_START消息
      - IjkMediaPlayer的循环里，ijkmp_get_msg处理FFP_REQ_START，然后调用**ffp_start_l**触发播放。
        - 本质而言，最终是调用toggle_pause实现“暂停->播放”的切换
- ijkmp_**stop**
  - ijkmp_stop_l
    - 先检测当前的状态是否可以执行stop，比如MP_STATE_IDLE状态就没有必要调用stop
    - 删除队列里的FFP_REQ_START/PAUSE消息，都已经要stop了，队列里面的start、pause消息已经没有意义。
    - 调用FFPlayer的ffp_stop_l
      - 先请求abort_request = 1，因为我们的packet queue、frame queue都需要abort退出
      - 然后暂停输出toggle_pause
      - msg_queue_abort消息队列也不允许再插入消息

比如什么时候该调用create创建IjkMediaPlayer，create接口本质上做了哪些操作，对于播放器我们一直说要划分。

目前的接口设计

```
IjkMediaPlayer();
    int ijkmp_create(std::function<int(void *)> msg_loop);
    int ijkmp_destroy();
    // 设置要播放的url
    int ijkmp_set_data_source(const char *url);
    // 准备播放
    int ijkmp_prepare_async();
    // 触发播放
    int ijkmp_start();
    // 停止
    int ijkmp_stop();
    // 暂停
    int ijkmp_pause();
    // seek到指定位置
    int ijkmp_seek_to(long msec);
    // 获取播放状态
    int ijkmp_get_state();
    // 是不是播放中
    bool ijkmp_is_playing();
    // 当前播放位置
    long ijkmp_get_current_position();
    // 总长度
    long ijkmp_get_duration();
    // 已经播放的长度
    long ijkmp_get_playable_duration();
    // 设置循环播放
    void ijkmp_set_loop(int loop);
```

```
// 获取是否循环播放
int  ijkmp_get_loop();
// 读取消息
int ijkmp_get_msg(AVMessage *msg, int block);
// 设置音量
void ijkmp_set_playback_volume(float volume);
```

2.3 手把代码实现步骤

IjkMediaPlayer类

IjkMediaPlayer成员变量

C++ 复制代码

```
    // 互斥量
      std::mutex mutex_;
      // 真正的播放器
      FFPlayer *ffplayer_ = NULL;
      //函数指针，指向创建的message_loop，即消息循环函数
//    int (*msg_loop)(void*);
      std::function<int(void *)> msg_loop_ = NULL; // ui处理消息的循环
      //消息机制线程
      std::thread *msg_thread_; // 执行msg_loop
//    SDL_Thread _msg_thread;
      //字符串，就是一个播放url
      char *data_source_;
      //播放器状态，例如prepared,resumed,error,completed等
      int mp_state_;  // 播放状态
```

IjkMediaPlayer成员函数

- ijkmp_create

- ijkmp_destroy
- ijkmp_prepare_async
- ijkmp_start
- ijkmp_stop

FFPlayer类

FFPlayer成员变量

C++ 复制代码

```cpp
std::function<int(const Frame *)> video_refresh_callback_ = NULL;
/* ffplay context */
VideoState *is;
const char* wanted_stream_spec[AVMEDIA_TYPE_NB];
```

FFPlayer成员函数

C++ 复制代码

```cpp
int ffp_create();
void ffp_reset_internal();
void ffp_destroy_p()
/* playback controll */
int       ffp_prepare_async_l(const char *file_name);
int       ffp_start_l();
int       ffp_stop_l();
```

消息循环线程

- 创建线程
- UI消息循环处理逻辑
- IjkMediaPlayer消息循环处理逻辑

UI MainWind消息循环

C++ 复制代码

```cpp
int MainWind::message_loop(void *arg)
{
    IjkMediaPlayer *mp = (IjkMediaPlayer *)arg;
    // 线程循环
    qDebug() << "message_loop into";
    while (1) {
        AVMessage msg;
        //取消息队列的消息，如果没有消息就阻塞，直到有消息被发到消息队列。
        int retval = mp->ijkmp_get_msg(&msg, 1);    // 主要处理Java->C的消息

        if (retval < 0)
            break;
        switch (msg.what) {
            case FFP_MSG_FLUSH:
                qDebug() << __FUNCTION__ << " FFP_MSG_FLUSH";
            break;
        case FFP_MSG_PREPARED:
            std::cout << __FUNCTION__ << " FFP_MSG_PREPARED" << std::endl;
            mp->ijkmp_start();
            break;
        default:
           qDebug()  << __FUNCTION__ << " default " << msg.what ;
          break;
        }
        msg_free_res(&msg);
//        qDebug() << "message_loop sleep, mp:" << mp;
        // 先模拟线程运行
        std::this_thread::sleep_for(std::chrono::milliseconds(1000));

    }
    qDebug() << "message_loop leave";
}
```

IjkMediaPlayer消息循环

C++ 复制代码

```cpp
int IjkMediaPlayer::ijkmp_get_msg(AVMessage *msg, int block)
{
    while (1) {
        int continue_wait_next_msg = 0;
        //取消息，如果没有消息则阻塞。
        int retval = msg_queue_get(&ffplayer_->msg_queue_, msg, block);
        if (retval <= 0)        // -1 abort, 0 没有消息
            return retval;
        switch (msg->what) {
            case FFP_MSG_PREPARED:
            std::cout << __FUNCTION__ << " FFP_MSG_PREPARED" << std::endl;
            break;
            case FFP_REQ_START:
            std::cout << __FUNCTION__ << " FFP_REQ_START" << std::endl;
            continue_wait_next_msg = 1;
            break;
          default:
            std::cout << __FUNCTION__ << " default " << msg->what << std::endl;
            break;
        }
        if (continue_wait_next_msg) {
            msg_free_res(msg);
            continue;
        }
        return retval;
    }
     return -1;
}
```

3 补充知识

参考1：Android中MediaPlayer的setDataSource方法的使用

https://blog.csdn.net/yanlinembed/article/details/51887642

ijkmp_set_data_source的设计来源于Android的 MediaPlayer，可以通过重载接口提供不同的资源类型。

MediaPlayer的setDataSource()方法主要有四种：

Sets the data source as a content Uri.

@param context the Context to use when resolving the Uri

@param uri the Content URI of the data you want to play

public void setDataSource(Context context, **Uri uri**)

Sets the data source (**file-path or http/rtsp URL**) to use.

@param path the path of the file, or the http/rtsp URL of the stream you want to play

public void setDataSource(**String** path)

Sets the data source (FileDescriptor) to use. It is the caller's responsibility

to close the file descriptor. It is safe to do so as soon as this call returns.

@param fd the FileDescriptor for the file you want to play

public void setDataSource(**FileDescriptor** fd)

Sets the data source (FileDescriptor) to use. The FileDescriptor must be

seekable (N.B. a LocalSocket is not seekable). It is the caller's responsibility

to close the file descriptor. It is safe to do so as soon as this call returns.

@param fd the FileDescriptor for the file you want to play

@param offset the offset into the file where the data to be played starts, in bytes

@param length the length in bytes of the data to be played

public void setDataSource(FileDescriptor fd, long offset, long length)

1. 播放应用的资源文件

```
法1. 直接调用create函数实例化一个MediaPlayer对象，播放位于res/raw/test.mp3文件
MediaPlayer  mMediaPlayer = MediaPlayer.create(this, R.raw.test);

法2. test.mp3放在res/raw/目录下，使用setDataSource(Context context, Uri uri)
mp = new MediaPlayer();
Uri setDataSourceuri =
Uri.parse("android.resource://com.android.sim/"+R.raw.test);
mp.setDataSource(this, uri);

说明：此种方法是通过res转换成uri然后调用setDataSource()方法，需要注意格式
Uri.parse("android.resource://[应用程序包名Application package
name]/"+R.raw.播放文件名);
例子中的包名为com.android.sim，播放文件名为：test;特别注意包名后的"/"。

法3. test.mp3文件放在assets目录下，使用setDataSource(FileDescriptor fd, long
offset, long length)
AssetManager assetMg = this.getApplicationContext().getAssets();
AssetFileDescriptor fileDescriptor = assetMg.openFd("test.mp3");
mp.setDataSource(fileDescriptor.getFileDescriptor(),
fileDescriptor.getStartOffset(), fileDescriptor.getLength());
```

2. 播放存储设备的资源文件

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource("/mnt/sdcard/test.mp3");
```

3. 播放远程的资源文件

```
Uri uri = Uri.parse("http://**");
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource(Context, uri);
```