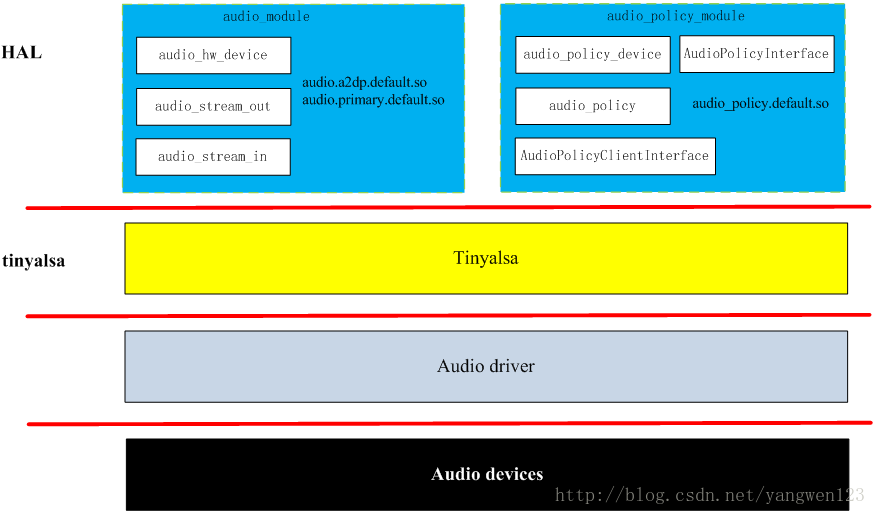
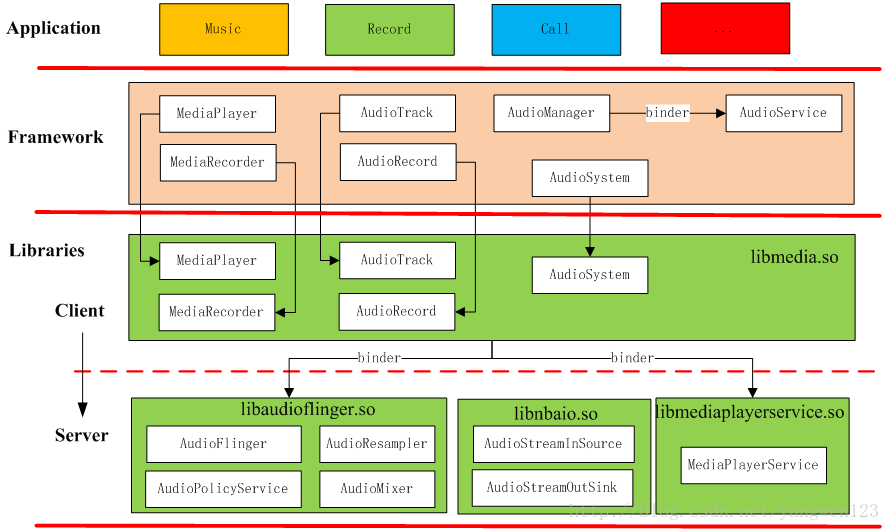
# Audio Playback explanation

Before we dive into audio playback, we need to understand how audio works in Android and modules involved in it.



Audio Application Framework: Audio Application Framework

AudioTrack: The output responsible for playback data, belonging to the Android Application Framework API class

AudioRecord: Responsible for the collection of recorded data, belonging to the Android Application Framework API class

AudioSystem: Responsible for the integrated management of audio transactions, belonging to the Android Application Framework API class

Audio Native Framework: Audio Local Framework

AudioTrack: The output responsible for playback data, belonging to the Android Local Framework API class

AudioRecord: Responsible for the collection of recorded data, belonging to the Android Local Framework API class

AudioSystem: Responsible for the integrated management of audio transactions, belonging to the Android Local Framework API class

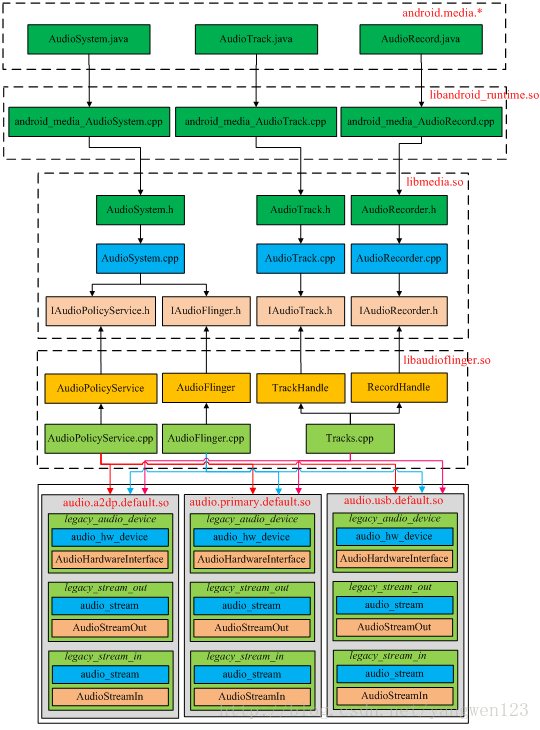
Audio Services: Audio Services

AudioPolicyService: the maker of the audio policy, responsible for the policy selection of the audio device switching, volume adjustment strategy, etc.

AudioFlinger: the executor of the audio policy, responsible for the management of the input and output stream devices and the processing and transmission of audio stream data.

Audio HAL: Audio hardware abstraction layer, responsible for interaction with audio hardware devices, directly called by AudioFlinger

AudioRecorder and AudioTrack are API classes that the Audio system provides externally. AudioRcorder is mainly used to complete the collection of audio data, and AudioTrack is responsible for the output of audio data. AudioFlinger manages the input and output audio streams in the system, and is responsible for the mixing of audio data, and realizes the input and output functions of audio data by reading and writing Audio hardware; AudioPolicyService is the policy control center of the Audio system, in charge of the selection and switching of sound devices in the system, Volume control, etc.



## AudioFlinger

System services in Android fall into two categories, namely Java Services and Native Services.

AudioFlinger and SurfaceFlinger belong to the Native Services.

Mediaserver starts all the native layer services [Viz: AudioFlinger, MediaPlayerService, CameraService, and AudioPolicyService] for achieving media related

As the audio hub in the Android system, it is also a system service, and it plays the role of starting up (providing access interface for the upper layer) and starting down (managing audio devices through HAL).

AudioFlinger inherits the template class BinderService, which is used to register native service.

BinderService is a template class, the publish function of this class is to complete the registration service with ServiceManager.

AudioFlinger registers a service named "media.audio\_flinger"

BnAudioFlinger is inherited from the RefBase layer, and the second parameter of IServiceManager::addService is actually a strong pointer reference (constsp&), so AudioFlinger has the program logic to call onFirstRef when the strong pointer is first referenced.

Although the AudioFlinger entity has been successfully created and initialized, it is still a static memory space so far, and no specific work is involved.

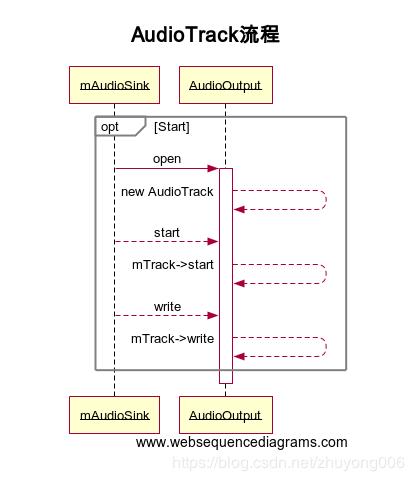
In terms of functional distribution,

AudioPolicyService is the policy maker, Such as when to open the audio interface device, what kind of device corresponds to a certain stream type of audio, etc.

AudioFlinger is the executor of the strategy, For example, how to communicate with audio equipment, how to maintain the audio equipment in the existing system, and how to deal with the mixing of multiple audio streams, etc. have to be completed by it.

# AUDIO TRACK

Audio Track



# Audio playback

Audio playback is divided into two programs of MediaPlayer and Audiotrack. MediaPlayer can play a variety of format sound files, such as MP3, WAV, OGG, AAC, MIDI, etc. However, Audiotrack can only play PCM data streams. Of course, there is still a close relationship between the two, when playing audio, when the Framework layer or the AudiotRack is created, and the decoded PCM digital stream is passed to the Audiotrack, and finally the audio is mixed, and the audio is played. Playing with Audiotrack is just skipping the decoding portion of MediaPlayer.

Audiotrack Audio = New AudiotRack (Audiomanager.Stream\_Music…)

byte[] buffer = new buffer[4096];

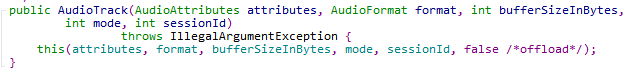
 audio.write(buffer, 0, 4096);

Audio.Play();

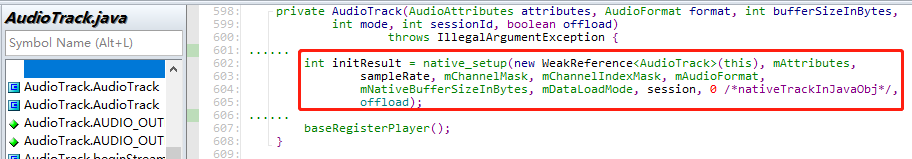
 audio.stop();

 audio.release();

Application creates an Audiotrack object by the following constructors



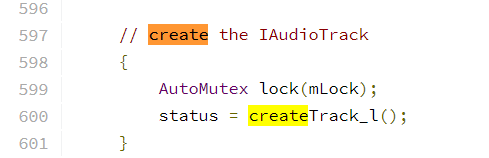
Then, by calling another constructor of itself, the setup method of Audiotrack.cpp is invoked through JNI in the construction method.



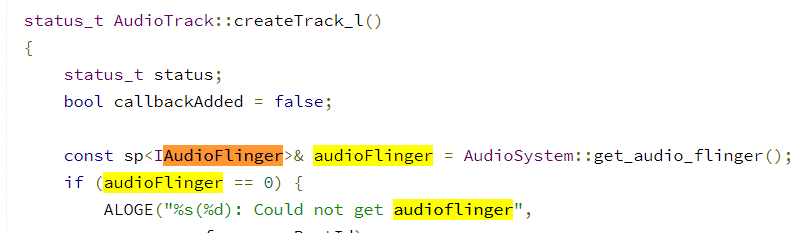
Create an *AudiotRack* object of the Native layer in Android\_Media\_AudiotRack\_Setup method and calls its set method.

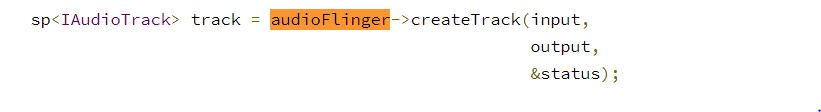
*AudiotrackJnistorage* is a container for audio data storage and is a package for anonymous sharing memory.

CreateTrack\_l method in the SET method of audiotrack.cpp



Create an Audioflinger object in CreateTrack\_L and call its CreateTrack method





CreateTrack method for calling Audioflinger

After the creation is successful, Audiotrack gets the PLAYBACKTHREAD :: Track, PCM's cross-trip transmission, the start / stop of the data stream is completed by Track, and the corresponding output OUTPUT is recorded in Audiotrack, and PlayBackthread can be determined by Output.

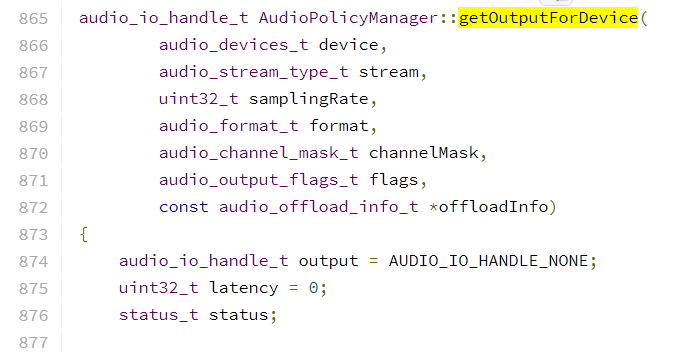
AUDIOSystem.getOutputForattr Gets the AudiopolicyService object and calls its GetputForattr method



AudioPolicyManager::getOutputForAttrInt(…)  
{….  
\*output = getOutputForDevices(…)  
…}

AudioPolicyManager::getOutputForDevice

Determine the final Flags in GetOutputDevices in GetOutputDevices.



If the Flags matches the Direct logo:

Look for matching Profile

Open corresponding OUTPUT

Create a new Output

After Creating Output, HAL will returned. The parameters of this output may be inconsistent with XML. If the parameters with the HAL do not match the non-HD path

AudioPolicyManager::selectOutput

AudioFlinger::openOutput

AudioFlinger::PlaybackThread::createTrack\_l

In Tracks.cpp AudioFlinger::PlaybackThread::Track::start

AudioFlinger::PlaybackThread::addTrack\_l

Execute StartOutput, notify AUDIPOLICY output, update status, and reselect the output device. At this point, the status of the track becomes Active, so PlaybackThread will add it in every THREADLOOP.

AudioFlinger::PlaybackThread::onAddNewTrack\_l

AudioFlinger::MixerThread::prepareTracks\_l

* Sampling in PrepareTracks\_L
* Calculate the volume (track -> SetFinalVolume())

Calling AUDIOMIXER's Process method in Threads.cpp

DUMP Mixed Audio Data

AudioMixer::process\_\_validate

AudioMixer::getProcessHook

AudioMixerBase::process\_\_noResampleOneTrack()

AudioFlinger::PlaybackThread::threadLoop\_write

AudioStreamOut::write

Write data from HAL