在 Android 中使用 JNI 和 C++ 实现 Python librosa.load 的功能，可以借助 miniaudio 库来进行音频文件的加载和解码。miniaudio 是一个轻量级的音频库，能够读取多种音频格式并提供方便的接口。

目标：

实现类似于 librosa.load 的功能：加载音频文件、进行采样率转换、将音频数据转换为浮动类型的 NumPy 数组（或其等效形式）。

步骤概述：

1. **集成 miniaudio 到 Android 项目中。**
2. **通过 JNI 使用 C++ 调用 miniaudio 读取音频文件。**
3. **进行采样率转换和音频数据的处理。**

1. 集成 miniaudio 到 Android 项目

miniaudio 是一个头文件库，可以直接在项目中使用。将 miniaudio 添加到你的项目中。

下载 miniaudio：

从 GitHub 下载 miniaudio 的源代码并将其放入你的项目中：

```
git clone https://github.com/mackron/miniaudio.git
```

将 miniaudio 的源代码放入你的 Android 项目的 cpp 文件夹中（例如：app/src/main/cpp/miniaudio）。

2. 使用 miniaudio 在 C++ 中加载音频文件

使用 miniaudio 的 API 加载音频文件并进行处理。下面是 C++ 中如何使用 miniaudio 读取音频文件并进行采样率转换的示例。

C++ 实现：

```
#include <jni.h>
#include <vector>
#include <iostream>
#include <miniaudio.h>
```

```
#define SAMPLE_RATE 22050  // 目标采样率

// 定义一个结构来存储音频文件的信息
typedef struct {
    std::vector<float> samples;
    int channels;
    int sample_rate;
} AudioData;

// 读取音频文件并返回音频数据
int read_audio_file(const char *filename, AudioData &audioData) {
    ma_decoder decoder;
    ma_decoder_config decoderConfig = ma_decoder_config_init(ma_format_f32, 0, SAMPLE_RATE);

    // 打开音频文件
    if (ma_decoder_init_file(filename, &decoderConfig, &decoder) != MA_SUCCESS) {
        return -1;  // 读取失败
    }

    // 获取音频的格式信息
    audioData.channels = decoder.outputChannelCount;
```

```
    audioData.sample_rate =
decoder.outputSampleRate;

    // 获取音频文件的长度（样本数）
    uint64_t totalFrameCount =
ma_decoder_get_length_in_pcm_frames(&d
ecoder);

    // 分配内存来存储音频样本

audioData.samples.resize(totalFrameCou
nt * audioData.channels);

    // 读取音频数据
    if
(ma_decoder_read_pcm_frames(&decoder,
audioData.samples.data(),
totalFrameCount, NULL) != MA_SUCCESS)
{
        ma_decoder_uninit(&decoder);
        return -2;  // 读取失败
    }

    ma_decoder_uninit(&decoder);  //
释放解码器资源
    return 0;  // 成功
}

// JNI 方法：加载音频文件并返回音频数据
```

```
extern "C"
JNIEXPORT jfloatArray JNICALL
Java_com_example_audio_AudioProcessor_
loadAudio(JNIEnv *env, jobject thiz,
jstring filePath) {
    // 获取音频文件路径
    const char *path = env-
>GetStringUTFChars(filePath, nullptr);

    // 读取音频数据
    AudioData audioData;
    int result = read_audio_file(path,
audioData);

    if (result != 0) {
        // 读取失败，抛出异常
        jthrowable exception = env-
>FindClass("java/io/IOException");
        env->ThrowNew(exception,
"Failed to load audio file.");
        env-
>ReleaseStringUTFChars(filePath,
path);
        return nullptr;
    }

    // 将音频数据转换为 float 数组
```

```
    jfloatArray resultArray = env->NewFloatArray(audioData.samples.size());
    env->SetFloatArrayRegion(resultArray, 0, audioData.samples.size(), audioData.samples.data());

    // 释放文件路径
    env->ReleaseStringUTFChars(filePath, path);

    return resultArray;
}
```

3. Java 端使用 JNI 调用

在 Java 端，你可以通过 JNI 调用 C++ 中的 loadAudio 方法，将音频文件路径传递给它，并获取音频样本数据。

Java 接口：

```
package com.example.audio;

public class AudioProcessor {
    static {
```

```
System.loadLibrary("audio_processor");
    }

    // JNI 方法加载音频文件
    public native float[]
loadAudio(String filePath);

    // 使用 JNI 方法
    public void
loadAndProcessAudio(String filePath) {
        float[] audioData =
loadAudio(filePath);
        if (audioData != null) {
            System.out.println("Audio
loaded with " + audioData.length + "
samples.");
        }
    }
}
```

4. 配置 CMake 和 Gradle

CMake 配置：

确保在 CMakeLists.txt 文件中添加了 miniaudio 头文件路径，并链接你的 C++ 代码。

```
cmake_minimum_required(VERSION 3.10.2)

project(audio_processor)

# 设置 NDK 和 Android 构建工具
set(CMAKE_TOOLCHAIN_FILE ${ANDROID_NDK}/build/cmake/android.toolchain.cmake)
set(ANDROID_NATIVE_API_LEVEL 21)

# 设置支持的架构
set(ANDROID_ABI "arm64-v8a")

# 包含 miniaudio 头文件
include_directories(src/main/cpp/miniaudio)

# 创建 JNI 库
add_library(audio_processor SHARED
    src/main/cpp/AudioProcessor.cpp
)

# 链接到 Android 的 log 库
target_link_libraries(audio_processor
    log)
```

Gradle 配置：

确保在 app/build.gradle 中启用了 NDK 构建，并配置了正确的 CMakeLists.txt 路径。

```
android {
    compileSdkVersion 30

    defaultConfig {
        applicationId
"com.example.audio"
        minSdkVersion 21
        targetSdkVersion 30
        ndk {
            moduleName
"audio_processor"
            externalNativeBuild {
                cmake {
                    path
"CMakeLists.txt"
                }
            }
        }
    }

    externalNativeBuild {
        cmake {
            path "CMakeLists.txt"
        }
    }
}
```

5. 其他步骤（可选）

采样率转换

如果你需要将音频的采样率转换为目标采样率（例如：librosa.load 默认的 22050Hz），你可以使用 miniaudio 的内置重采样功能或使用其他库（如 libsamplerate）来实现。

总结

通过上述步骤，你可以通过 JNI 和 miniaudio 在 Android 项目中实现类似于 Python librosa.load 的功能。关键步骤包括：

- 使用 miniaudio 读取音频文件。
- 使用 JNI 将 C++ 代码与 Java 交互。
- 配置 Android 的 CMakeLists.txt 和 Gradle 构建系统。