如果不借用第三方库，完全通过原生 C++ 解析 16,000 Hz 单通道 WAV 文件，可以手动解析 WAV 文件的头部并提取音频数据。这需要了解 WAV 文件的结构。

WAV 文件结构

WAV 文件格式基于 RIFF（Resource Interchange File Format），主要包括以下几个部分：

1. **RIFF Header**：文件标识符。
   - Chunk ID: RIFF (4 bytes)
   - Chunk Size: 文件总大小 - 8 (4 bytes)
   - Format: WAVE (4 bytes)
2. **Format Chunk**：描述音频格式。
   - Subchunk1 ID: fmt (4 bytes)
   - Subchunk1 Size: 通常为 16 或 18 (4 bytes)
   - Audio Format: PCM = 1 (2 bytes)
   - Num Channels: 声道数（单声道为 1，立体声为 2）(2 bytes)
   - Sample Rate: 采样率 (4 bytes)
   - Byte Rate: 每秒数据量 = 采样率 × 声道数 × 每样本字节数 (4 bytes)

- Block Align: 每采样块字节数 = 声道数 × 每样本字节数 (2 bytes)
- Bits Per Sample: 每样本位数 (2 bytes)

3. **Data Chunk**：包含音频数据。
   - Subchunk2 ID: data (4 bytes)
   - Subchunk2 Size: 音频数据大小 (4 bytes)
   - Audio Data: 原始音频数据。

实现步骤

1. 读取 WAV 文件头

解析头部信息，验证文件是否是合法的 WAV 文件。

2. 提取音频数据

读取音频数据部分，根据采样率、位深、声道数进行转换，将数据存入 std::vector<float>。

3. C++ 实现代码

以下是完整实现：

```
#include <jni.h>
```

```cpp
#include <vector>
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <cstdint>

struct WavHeader {
    char riff[4];                // "RIFF"
    uint32_t chunkSize;          // 文件大小 - 8
    char wave[4];                // "WAVE"
    char fmt[4];                 // "fmt "
    uint32_t subchunk1Size;      // 子块大小，通常为 16
    uint16_t audioFormat;        // 音频格式，1 = PCM
    uint16_t numChannels;        // 声道数
    uint32_t sampleRate;         // 采样率
    uint32_t byteRate;           // 每秒字节数
    uint16_t blockAlign;         // 每采样块字节数
```

```
    uint16_t bitsPerSample;     // 每样本位数
    char data[4];               // "data"
    uint32_t dataSize;          // 数据块大小
};

std::vector<float> loadWavFile(const std::string& filePath) {
    // 打开文件
    std::ifstream file(filePath, std::ios::binary);
    if (!file.is_open()) {
        throw std::runtime_error("Failed to open WAV file.");
    }

    // 读取 WAV 文件头
    WavHeader header;

file.read(reinterpret_cast<char*>(&header), sizeof(WavHeader));

    // 验证 WAV 文件
    if (std::string(header.riff, 4) != "RIFF" ||
```

```
            std::string(header.wave, 4) !=
"WAVE" ||
            std::string(header.fmt, 4) !=
"fmt " ||
            std::string(header.data, 4) !=
"data") {
            throw
std::runtime_error("Invalid WAV file
format.");
        }

        if (header.audioFormat != 1) {
            throw
std::runtime_error("Unsupported audio
format. Only PCM is supported.");
        }
        if (header.numChannels != 1) {
            throw std::runtime_error("Only
mono channel WAV files are
supported.");
        }
        if (header.bitsPerSample != 16) {
            throw std::runtime_error("Only
16-bit PCM WAV files are supported.");
        }

        // 读取音频数据
```

```
    std::vector<float>
audioData(header.dataSize /
sizeof(int16_t));
    for (size_t i = 0; i <
audioData.size(); ++i) {
        int16_t sample;

file.read(reinterpret_cast<char*>(&sam
ple), sizeof(int16_t));
        audioData[i] =
static_cast<float>(sample) / 32768.0f;
// 将 16-bit 转换为 [-1, 1]
    }

    file.close();
    return audioData;
}

// JNI 方法
extern "C" JNIEXPORT jfloatArray
JNICALL
Java_com_example_wavloader_WavLoader_l
oadWavToVector(JNIEnv* env, jobject,
jstring filePath) {
    // 转换 Java 字符串到 C++ 字符串
    const char* path = env-
>GetStringUTFChars(filePath, nullptr);
    std::string filePathStr(path);
```

```
    env->ReleaseStringUTFChars(filePath, path);

    try {
        // 加载 WAV 文件
        std::vector<float> audioData = loadWavFile(filePathStr);

        // 转换 std::vector<float> 为 Java float[]
        jfloatArray result = env->NewFloatArray(audioData.size());
        env->SetFloatArrayRegion(result, 0, audioData.size(), audioData.data());
        return result;
    } catch (const std::exception& e) {
        // 异常处理
        std::cerr << e.what() << std::endl;
        return nullptr;
    }
}
```

代码解析

1. **WAV 文件解析**：

- 使用 WavHeader 结构体读取和验证头部信息。
- 检查是否为 PCM 格式、16 位单通道音频。

2. **音频数据读取**：
   - 将每个 16-bit 样本归一化到 [-1.0, 1.0]。
   - 存储在 std::vector<float> 中。
3. **JNI 接口**：
   - 提供 loadWavToVector 方法，用于 Java 层调用。
   - 将 C++ 的 std::vector<float> 转换为 Java 的 float[]。

Java 层调用

在 Java 层定义 JNI 方法并调用：

JNI 接口： WavLoader.java

```
package com.example.wavloader;

public class WavLoader {
    static {

System.loadLibrary("wav_loader");
```

```
    }

    public native float[]
loadWavToVector(String filePath);
}
```

调用示例：

```
WavLoader loader = new WavLoader();
float[] audioData =
loader.loadWavToVector("/path/to/
audio.wav");

if (audioData != null) {
    System.out.println("Loaded audio,
samples: " + audioData.length);
} else {
    System.out.println("Failed to load
WAV file.");
}
```

总结

此实现不依赖第三方库，直接通过 C++ 手动解析 WAV 文件格式，适用于 16,000 Hz 单通道音频。此方法简单高效，但如果需要支持更多格式，建议使用专用的音频处理库如 libsndfile 或 FFmpeg。