封面

编译 kaldi-android

www.gaohaiyan.com

需求：将 kaldi 编译为 Android8.0-api26、ARMv8 使用的 so 动态库文件。

环境 1：XUbuntu20.04.3，python3.8.10、android_ndk_r20b、git、apt。（建议的编译环境）
环境 2：Mac10.14.3，python3、android-ndk、git、brew。

本文参考：
https://www.jianshu.com/p/a896bc4c3c14
https://www.jianshu.com/p/905214cedf97
http://jcsilva.github.io/2017/03/18/compile-kaldi-android/
https://medium.com/swlh/compile-kaldi-for-64-bit-android-on-ubuntu-18-70967eb3a308

安装一些必须的程序：

- linux 指令：

```
sudo apt-get install cmake g++ automake autoconf git sox gfortran libtool subversion python2.7 zlib1g-dev python3-distutils
```

- macos 指令：

```
brew install XXX
```

gfortran 可以从 https://github.com/fxcoudert/gfortran-for-macOS/releases 下载
也可以根据下文步骤说明安装

1.准备 toolchain

下载地址：https://developer.android.google.cn/ndk/downloads?hl=zh_cn ，
新版(r23b)的 dmg 解压出来是个 app，右键显示包内容，将"NDK"目录提取出来即可。
本例为 r20b，解压到用户目录下 ~/apps/android-ndk-r20b。
不要使用 ndk 内置的 ~/apps/android-ndk-r20b/toolchains，通过 make_standalone_toolchain.py 手动创建：

```
$ python3 ~/apps/android-ndk-r20b/build/tools/make_standalone_toolchain.py \
      --arch arm64 \ # 芯片位
      --api 26 \ # AndroidAPI 版本
      --install-dir ~/apps/android-ndk-r20b/arm64-toolchain # 生成目录
```

终端输入以下 3 句，或配置到环境变量：

```
export ANDROID_NDK_HOME=~/apps/android-ndk-r20b
export ANDROID_TOOLCHAIN_HOME=~/apps/android-ndk-r20b/arm64-toolchain
export PATH=$ANDROID_NDK_HOME:$ANDROID_TOOLCHAIN_HOME/bin:$PATH:.
```

linux 可以配置到 ~/.bashrc，macos 可以配置到 ~/.bash_profile。

2.编译 openblas

下载地址：https://github.com/xianyi/OpenBLAS/releases
本例为 OpenBLAS-0.3.18.tar.gz，解压到用户目录下 ~/apps/openblas_0.3.18。
编译参考 https://github.com/xianyi/OpenBLAS/wiki/How-to-build-OpenBLAS-for-Android

```
$ cd ~/apps/openblas_0.3.18/
$ make clean
$ make \
    TARGET=ARMV8 \
    BINARY=64 \
    ONLY_CBLAS=1 \
    CC="$ANDROID_TOOLCHAIN_HOME"/bin/aarch64-linux-android26-clang \
    AR="$ANDROID_TOOLCHAIN_HOME"/bin/aarch64-linux-android-ar \ # r23b 版 ndk 没有 ***_ar
    HOSTCC=gcc \
    ARM_SOFTFP_ABI=1 \
    -j4
echo OK.
OK.
rm -f linktest
 OpenBLAS build complete. (CBLAS)
  OS                ... Android
  Architecture      ... arm64
  BINARY            ... 64bit
  C compiler        ... CLANG  (cmd & version : Android (5220042 based on r346389c) clang version 8.0.7
(https://android.googlesource.com/toolchain/clang          b55f2d4ebfd35bf643d27dbca1bb228957008617)
(https://android.googlesource.com/toolchain/llvm 3c393fe7a7e13b0fba4ac75a01aa683d7a5b11cd) (based on
LLVM 8.0.7svn))
-n   Library Name     ... libopenblas_armv8p-r0.3.18.a
 (Multi-threading; Max num-threads is 8)
To install the library, you can run "make PREFIX=/path/to/your/installation install".
$
```

如下指令，将编译完成的 OpenBLAS 安装在当前路径的 install 目录。

```
$ make install NO_SHARED=1 PREFIX=`pwd`/install
/Library/Developer/CommandLineTools/usr/bin/make -j 8 -f Makefile.install install
Generating openblas_config.h in ~/apps/openblas_0.3.18/install/include
Generating f77blas.h in ~/apps/openblas_0.3.18/install/include
Generating cblas.h in ~/apps/openblas_0.3.18/install/include
Copying LAPACKE header files to ~/apps/openblas_0.3.18/install/include
Copying the static library to ~/apps/openblas_0.3.18/install/lib
Generating openblas.pc in ~/apps/openblas_0.3.18/install/lib/pkgconfig
Generating OpenBLASConfig.cmake in ~/apps/openblas_0.3.18/install/lib/cmake/openblas
Generating OpenBLASConfigVersion.cmake in ~/apps/openblas_0.3.18/install/lib/cmake/openblas
Install OK!
$
```

3.编译 clapack

下载地址：https://github.com/simonlynen/android_libs/tree/master/lapack/jni 直接使用，
也可以从 http://www.netlib.org/clapack 下载"clapack.tgz"进行修改，本例为 github 版本 3.2.1。
备选链接：https://pan.baidu.com/s/1RU-Md0Z5iK_2WW5-ITJkXA （提取码：jn4i）
或 https://download.csdn.net/download/Vigiles/12245335
解压到用户目录下 ~/apps/clapack_3.2.1。
修改 **Application.mk** 文件：

```
APP_STL := c++_static # c++_static 或 c++_shared
APP_CPPFLAGS := -frtti -fexceptions -mfloat-abi=softfp -mfpu=neon -std=c++ -Wno-deprecated
-ftree-vectorize -ffast-math -fsingle-precision-constant
APP_ABI := arm64-v8a # armeabi-v7a
APP_OPTIM := release
NDK_TOOLCHAIN_VERSION := clang++
APP_PLATFORM := android-26
```

使用 ndk-build 编译

```
$ cd ~/apps/clapack_3.2.1
$ ndk-build NDK_PROJECT_PATH=. APP_BUILD_SCRIPT=./Android.mk NDK_APPLICATION_MK=./Application.mk
[arm64-v8a] Compile       : f2c <= xwsne.c
[arm64-v8a] Compile       : f2c <= dtime_.c
[arm64-v8a] Compile       : f2c <= etime_.c
[arm64-v8a] StaticLibrary  : libf2c.a
[arm64-v8a] SharedLibrary  : liblapack.so
[arm64-v8a] Install       : liblapack.so => libs/arm64-v8a/liblapack.so
$
```

结束后，复制生成的 ~/apps/clapack_3.2.1/obj/local/armeabi-v8a/ 里的全部文件
libblas.a libclapack.a libf2c.a liblapack.so objs(文件夹)
到 **~/apps/openblas_0.3.18/install/lib** 目录。如果没有 **liblapack.a**，就复制 libClapack.a 为 liblapack.a 。

4.编译 kaldi

下载

```
$ git clone https://github.com/kaldi-asr/kaldi ~/apps/kaldi_android
```

本例 kaldi 的 commit id：66f5434d29e2a528b9363e0fa25f2793069602a3

使用如下指令检查依赖库和工具是否齐全。

```
$ cd ~/apps/kaldi_android/tools/
$ extras/check_dependencies.sh
```

根据提示，linux 使用 ap-get、macos 使用 brew 进行安装。

4.1.编译 openfst

本步骤可参考 ~/apps/kaldi_android/tools/Makefile 文件内容，约 55 行开始。

不要直接在 tools 目录下执行 make 编译，否则编译 kaldi 时可能有下面的 openfst 错误：

```
~/apps/kaldi_android/tools/openfst-1.7.2/lib/libfst.a: error adding symbols: File in wrong format
clang80++: error: linker command failed with exit code 1 (use -v to see invocation)
```

下载地址：https://www.openfst.org/twiki/bin/view/FST/FstDownload

本例为 openfst-1.7.2.tar.gz，解压到 kaldi 目录下 ~/apps/kaldi_android/tools/openfst-1.7.2。

```
$ cd ~/apps/kaldi_android/tools/openfst-1.7.2
$ ./configure \
    --prefix=`pwd` \
    --enable-static \
    --enable-shared \
    --enable-far \
    --enable-ngram-fsts \
    --host=aarch64-linux-android \
    --cache-file=aarch64-linux-android.cache \
    --enable-lookahead-fsts \
    --with-pic \
    LIBS="-ldl"
```

如果报错：

```
configure: error: cannot run test program while cross compiling
```

编辑 configure 文件，约 16776 行，注释掉 if 块内语句：

```
if test "$cross_compiling" = yes; then :
#  { { $as_echo "$as_me:${as_lineno-$LINENO}: error: in \`$ac_pwd':" >&5
#$as_echo "$as_me: error: in \`$ac_pwd':" >&2;}
#as_fn_error $? "cannot run test program while cross compiling
#See \`config.log' for more details" "$LINENO" 5; }
else
```

编译：

```
$ make -j4
$ make install
$ cd ../
$ ln -s openfst-1.7.2 openfst
```

4.2.编译 kaldi

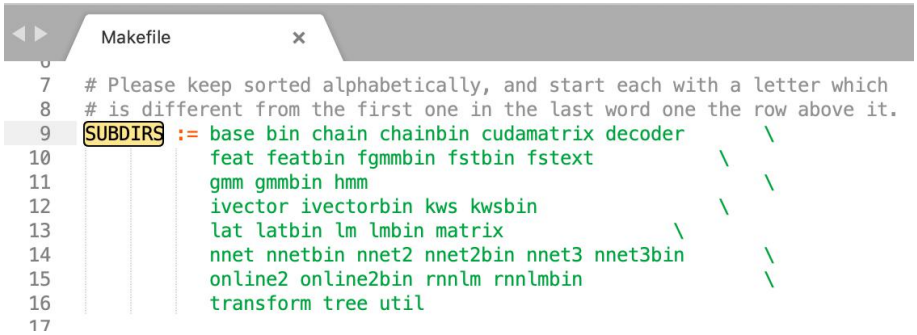
进入 ~/apps/kaldi_android/src/matrix，打开 Makefile 文件，

将 TESTFILES 一行注释。

进入 ~/apps/kaldi_android/src，

打开 **Makefile**，SUBDIRS 配置了哪些模块会被编译。可以根据需要删除手机上用不到的模块，也可以添加自己开发的模块。

可以看到本例的 kaldi 默认只有 online2，而**没有 online**。



打开 **configure** 文件，

（1）找到约 750 行，

--android-incdir=*) 一项配置，设置 dynamic_kaldi=true 。

（2）约 952 行，

if [["$use_cuda" = true && ! -f $CUBROOT/cub/cub.cuh]]; then 代码块，整个注释，关闭英伟达的 GPU 库 cuda。

创建 kaldi.mk：

```
$ cd ~/apps/kaldi_android/src
$ CXX=clang++ \
    ./configure --shared \
    --openblas-root=~/apps/openblas_0.3.18/install \ # 《2.编译 openblas》的路径
    --android-incdir=~/apps/android-ndk-r20b/arm64-toolchain/sysroot/usr/include \
    --host=aarch64-linux-android \ # arm-linux-androideabi 对应 32 位, aarch64-linux-android 对应 64 位 ARMv8。
    --use-cuda=no
Configuring KALDI to use OPENBLAS.
Checking compiler aarch64-linux-android-clang++ ...
Checking OpenFst library in ~/apps/kaldi/tools/openfst-1.7.2 ...
Performing OS specific configuration ...
Using OpenBLAS as the linear algebra library.
Successfully configured for Android with OpenBLAS from ~/apps/openblas_0.3.18/install.
Kaldi has been successfully configured. To compile:
  make -j clean depend; make -j <NCPU>
where <NCPU> is the number of parallel builds you can afford to do. If unsure,
use the smaller of the number of CPUs or the amount of RAM in GB divided by 2,
to stay within safe limits. 'make -j' without the numeric value may not limit
the number of parallel jobs at all, and overwhelm even a powerful workstation,
since Kaldi build is highly parallelized.
$
```

如果提示：

```
***configure failed: aarch64-linux-android-clang++ is not installed.
You need GNU g++ >= 5.0, Apple clang >= 6.0 or LLVM clang >= 3.5. ***
```

可能是 ***--android-incdir*** 设置的路径未访问到，检查《1.准备 toolchain》的 toolchain 的路径。

在 ~/apps/kaldi_android/src 目录生成 kaldi.mk 文件。打开修改：

约 57 行一项，末尾行注释并添加一条 -O3 -DNDEBUG。释义：https://www.cnblogs.com/x0rjchen/p/9310671.html 。

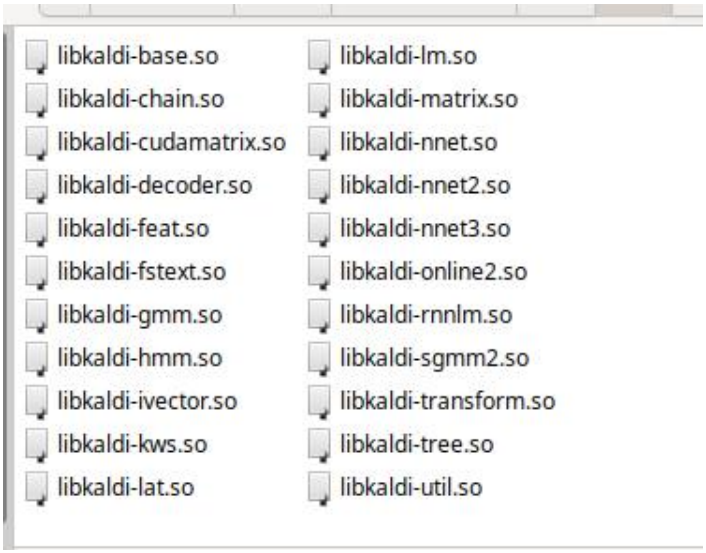
c++版本改为 11，默认 14.

```
CXXFLAGS = -std=c++11 -I.. -I$(OPENFSTINC) -O1 $(EXTRA_CXXFLAGS) \
           -Wall -Wno-sign-compare -Wno-unused-local-typedefs \
           -Wno-deprecated-declarations -Winit-self -Wno-mismatched-tags \
           -DKALDI_DOUBLEPRECISION=$(DOUBLE_PRECISION) \
           -DHAVE_CXXABI_H -DHAVE_OPENBLAS -DANDROID_BUILD \
           -I$(OPENBLASINC) -I$(ANDROIDINC) -ftree-vectorize -mfloat-abi=softfp \
           -mfpu=neon -pthread \
           -O3 -DNDEBUG # -g # -O0 -DKALDI_PARANOID
```

编译：

```
$ make -j4 clean depend
$ make -j4
```

生成的 so 文件可在 ~/apps/kaldi_android/src/lib 下找到。



这些都是软连接，可以使用 cp 命令拷贝 22 个对应的实际 so 文件到已存在的目录 ~/Desktop/kaldi-lib ：

```
$ cd ~/apps/kaldi_android/src/lib
$ cp -L ./* ~/Desktop/kaldi-lib
```

可以使用 aarch64-linux-android-readelf 查看 so 文件的信息。

~/apps/android-ndk-r20b/toolchains/aarch64-linux-android-4.9/prebuilt/darwin-x86_64/bin/aarch64-linux-android-readelf ，

或 ~/apps/android-ndk-r20b/toolchains/llvm/prebuilt/darwin-x86_64/bin/aarch64-linux-android-readelf。

5.cmake 集成 kaldi

本章节内容谨供参考，实际开发通常只在手机上做识别，用不到这么多头文件和库文件。

本例手机 MiNote3，处理器是骁龙 660，MIUI 12.0.1（Android 9）。

使用 adb 指令可以查看 cpu 架构。

$ adb shell getprop ro.product.cpu.abi

获取系统版本：

$ adb shell getprop ro.build.version.release

获取系统 api 版本

$ adb shell getprop ro.build.version.sdk

5.1.准备 h 文件

从 ~/apps/kaldi_android/src ，提取不包含“bin”结尾的文件夹，并且只提取 h 文件。

使用下面的指令和 python 代码，将目标文件拷贝到 ~/Desktop/kaldi_src_h 。

- 第 1 步 shell，获取 src 下的文件夹名，即模块名。也可直接在 python 中提取：

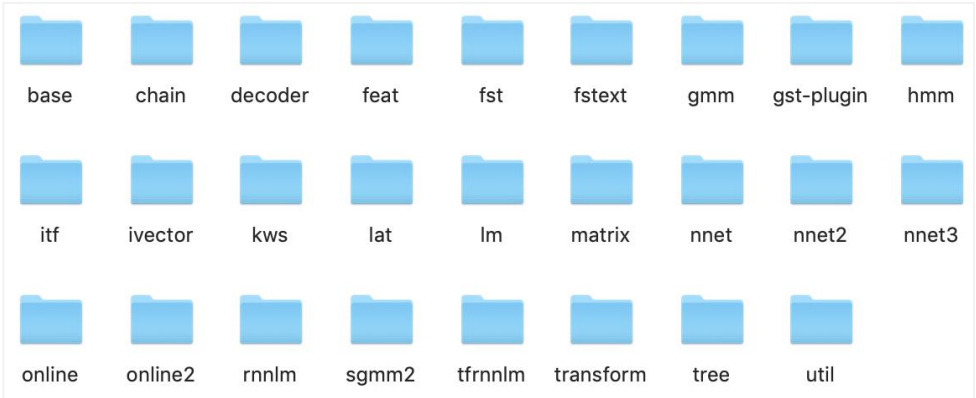
```
$ ls -l | grep ^d | grep -v "bin" | awk 'BEGIN {FS=" "} {print $9}' | awk 'BEGIN{RS="\n"; ORS="\", \"";}{print $0}'
```

- 第 2 步 python：

```
import os
import glob
from shutil import copyfile

fs = ["base", "chain", "cudadecoder", "cudafeat", "cudamatrix", "decoder", "doc", "feat",
     "fstext", "gmm", "gst-plugin", "hmm", "itf", "ivector", "kws",
     "lat", "lm", "makefiles", "matrix", "nnet", "nnet2", "nnet3", "online", "online2",
      "probe", "rnnlm", "sgmm2", "tfrnnlm", "transform", "tree", "util"]
src = "~/apps/kaldi_android/src"
dst = "~/Desktop/kaldi_src_h"

for folder in fs:
    dst_folder = os.path.join(dst, folder)
    if not os.path.exists(dst_folder):
        os.makedirs(dst_folder)
    src_dir = os.path.join(src, folder)
    h_dir = os.path.join(src_dir, '*.h')
    h_list = glob.glob(h_dir)
    if len(h_list) < 1:
        continue
    for h in h_list:
        name = os.path.basename(h)
        dist_file = os.path.join(dst_folder, name)
        copyfile(h, dist_file)
print("finish")
```

去除 cuda 开头的，

添加 fst 的 ~/apps/kaldi/tools/openfst-1.7.2/src/include/fst 。



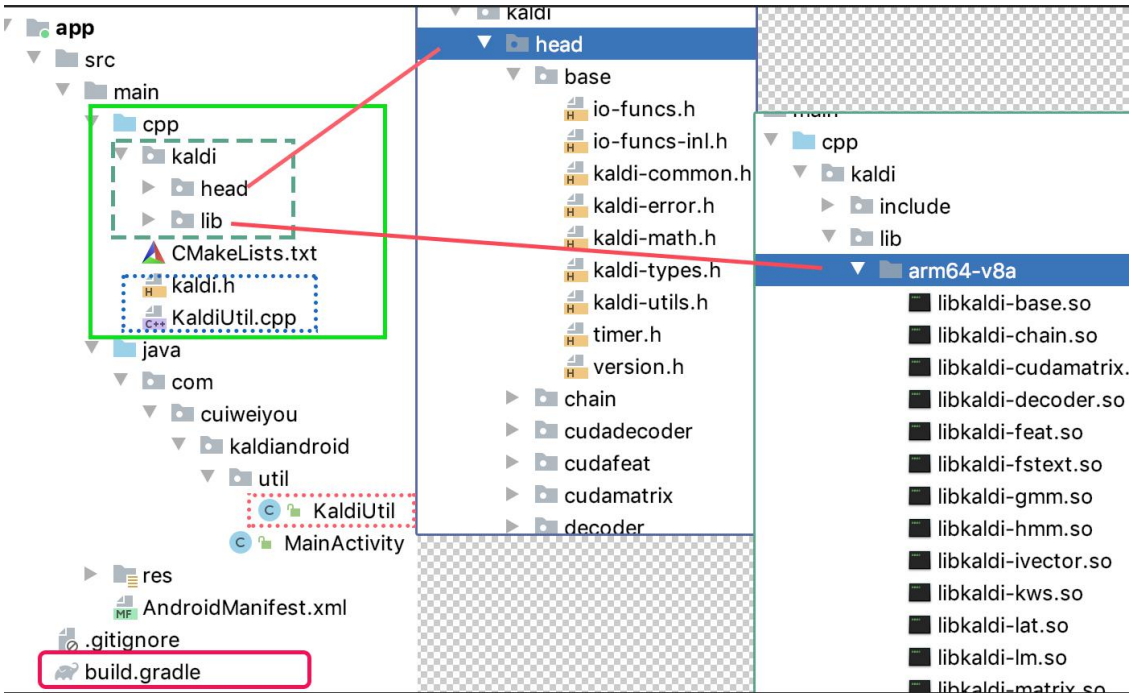
5.2.准备 so 文件

《4.2.编译 kaldi》步骤已经提取到 ~/Desktop/kaldi-lib 。

5.3.Android 项目配置

创建一个普通的安卓项目。

- 在 java 下像通常一样编写本地类方法 KaldiUtil.java。
- 在 main 下创建 cpp 文件夹,
  - ► 在 cpp 下创建 CMakeLists.txt 文件,
  - ► 在 cpp 下放入自己编写的本地方法 cpp 及自定义的 h 。
  - ► 在 cpp 下创建一个 kaldi 文件夹,
    - ★ 在 kaldi 下创建 head 和 lib 文件夹,
      - ◆ head 中放入 h 文件,
      - ◆ lib 中放入 so 文件。



5.3.1.本地方法 java 类

KaldiUtil.java :

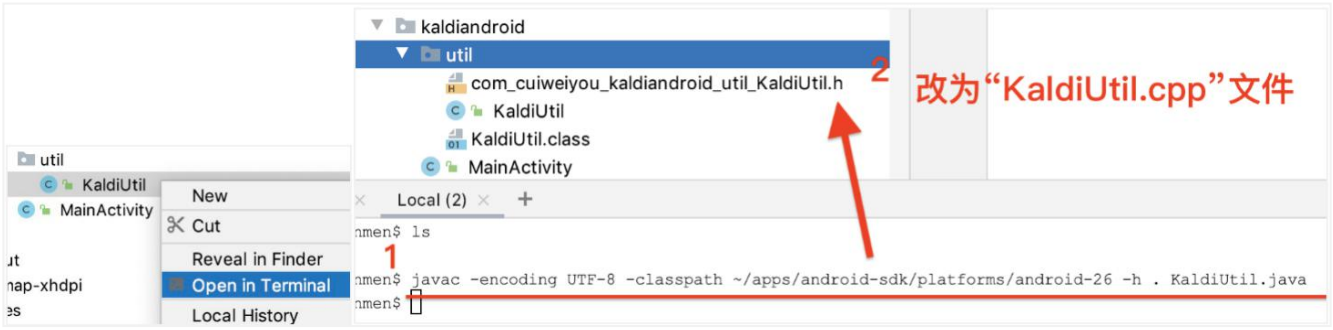
```
package com.cuiweiyou.kaldiandroid.util;


public class KaldiUtil {
    static {
        System.loadLibrary("KaldiUtil"); // 加载 cmake 生成的库。这个库名对应 CMakeLists.txt 中的配置
    }
    public native double KaldiMathLogAdd(double x, double y);  // 本地方法
}
```

右键 这个文件，Open in Terminal，使用 javac 指令生成对应的 h 文件，

把这个 h 文件改名为 KaldiUtil.cpp，放到 src/main/cpp 下。

```
$ javac -encoding UTF-8
    -classpath ~/apps/android-sdk/platforms/android-26
```

-h . Jni工具类.java



5.3.2.本地方法 cpp 实现

KaldiUtil.cpp :

```
//
// 这个原本是 javac 生成的 com_cuiweiyou_kaldiandroid_util_KaldiUtil.h 文件,
// 直接改后缀名为 cpp, 并修改代码。
//
// 此 cpp 文件在 jni 入口 CMakeLists.txt 中引入进行编译。
//

// 引用 Android 准备好的
#include <cstring>
#include <jni.h>
#include <cinttypes>
#include <android/log.h>

// 这里引用 kaldi 及自己写的
#include "kaldi.h" // 这里可以直接 include 自己需要的 h。

#define LOG_TAG "System.out"
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR, LOG_TAG, __VA_ARGS__) // 调用 Android 的 log。

#ifndef _Included_com_cuiweiyou_kaldiandroid_util_KaldiUtil
#define _Included_com_cuiweiyou_kaldiandroid_util_KaldiUtil
#ifdef __cplusplus
extern "C" {
#endif

/*
 * 这个是原 h 文件声明的函数, 这里直接修改为实现函数。
 * Class:     com_cuiweiyou_kaldiandroid_util_KaldiUtil
 * Method:    KaldiMathLogAdd
 * Signature: (DD)D
 */
JNIEXPORT jdouble JNICALL Java_com_cuiweiyou_kaldiandroid_util_KaldiUtil_KaldiMathLogAdd
  (JNIEnv *jniEnv, jobject, jdouble x, jdouble y){

    LOGE("进入 JNI 的范畴了", "");
    LOGE("调用 kaldi 的函数", "");
```

```
    double v = kaldi::LogAdd(x, y); // 调用 base/kaldi-math.h 中的函数

    LOGE("kaldi 计算结束，返回 java", "");

    return v;
};

#ifdef __cplusplus
}
#endif
#endif
```

kaldi.h 不是必须的，这里仅供演示：

```
//
// 统一说明及管理
//
//

#ifndef KALDIANDROID_KALDI_H
#define KALDIANDROID_KALDI_H

// kaldi 基础类
// #include "base/io-funcs-inl.h"
// #include "base/io-funcs.h"
// #include "base/kaldi-common.h"
// #include "base/kaldi-error.h"
 #include "base/kaldi-math.h" // ------------------------------------ 本例使用的 h
// #include "base/kaldi-types.h"
// #include "base/kaldi-utils.h"
// #include "base/timer.h"
// #include "base/version.h"

// Chain 模型基础类
// #include "chain/chain-datastruct.h"
// #include "chain/chain-den-graph.h"
// #include "chain/chain-denominator.h"
// #include "chain/chain-generic-numerator.h"
// #include "chain/chain-kernels-ansi.h"
// #include "chain/chain-numerator.h"
// #include "chain/chain-supervision.h"
// #include "chain/chain-training.h"
// #include "chain/language-model.h"

// cuda 开头的是英伟达显卡相关 api。目前手机都无法使用
// #include "cudadecoder/batched-static-nnet3-kernels.h"
// #include "cudadecoder/batched-static-nnet3.h"
// #include "cudadecoder/batched-threaded-nnet3-cuda-online-pipeline.h"
// #include "cudadecoder/batched-threaded-nnet3-cuda-pipeline.h"
// #include "cudadecoder/batched-threaded-nnet3-cuda-pipeline2.h"
```

```
// #include "cudadecoder/cuda-decodable-itf.h"
// #include "cudadecoder/cuda-decoder-common.h"
// #include "cudadecoder/cuda-decoder-kernels-utils.h"
// #include "cudadecoder/cuda-decoder-kernels.h"
// #include "cudadecoder/cuda-decoder.h"
// #include "cudadecoder/cuda-fst.h"
// #include "cudadecoder/cuda-online-pipeline-dynamic-batcher.h"
// #include "cudadecoder/cuda-pipeline-common.h"
// #include "cudadecoder/decodable-cumatrix.h"
// #include "cudadecoder/lattice-postprocessor.h"
// #include "cudadecoder/thread-pool-light.h"
// #include "cudadecoder/thread-pool.h"
// --
// #include "cudafeat/feature-online-batched-cmvn-cuda-kernels.h"
// #include "cudafeat/feature-online-batched-cmvn-cuda.h"
// #include "cudafeat/feature-online-batched-ivector-cuda-kernels.h"
// #include "cudafeat/feature-online-batched-ivector-cuda.h"
// #include "cudafeat/feature-online-batched-spectral-cuda-kernels.h"
// #include "cudafeat/feature-online-batched-spectral-cuda.h"
// #include "cudafeat/feature-online-cmvn-cuda.h"
// #include "cudafeat/feature-spectral-cuda.h"
// #include "cudafeat/feature-window-cuda.h"
// #include "cudafeat/lane-desc.h"
// #include "cudafeat/online-batched-feature-pipeline-cuda.h"
// #include "cudafeat/online-cuda-feature-pipeline.h"
// #include "cudafeat/online-ivector-feature-cuda-kernels.h"
// #include "cudafeat/online-ivector-feature-cuda.h"
// --
// #include "cudamatrix/cu-allocator.h"
// #include "cudamatrix/cu-array-inl.h"
// #include "cudamatrix/cu-array.h"
// #include "cudamatrix/cu-block-matrix.h"
// #include "cudamatrix/cu-common.h"
// #include "cudamatrix/cu-compressed-matrix.h"
// #include "cudamatrix/cu-device.h"
// #include "cudamatrix/cu-kernels-ansi.h"
// #include "cudamatrix/cu-kernels.h"
// #include "cudamatrix/cu-math.h"
// #include "cudamatrix/cu-matrix-inl.h"
// #include "cudamatrix/cu-matrix-lib.h"
// #include "cudamatrix/cu-matrix.h"
// #include "cudamatrix/cu-matrixdim.h"
// #include "cudamatrix/cu-packed-matrix.h"
// #include "cudamatrix/cu-rand.h"
// #include "cudamatrix/cu-sp-matrix.h"
// #include "cudamatrix/cu-sparse-matrix.h"
// #include "cudamatrix/cu-tp-matrix.h"
// #include "cudamatrix/cu-value.h"
// #include "cudamatrix/cu-vector.h"
// #include "cudamatrix/cublas-wrappers.h"
```

```
// 解码器相关
// #include "decoder/biglm-faster-decoder.h"
// #include "decoder/decodable-mapped.h"
// #include "decoder/decodable-matrix.h"
// #include "decoder/decodable-sum.h"
// #include "decoder/decoder-wrappers.h"
// #include "decoder/faster-decoder.h"
// #include "decoder/grammar-fst.h"
// #include "decoder/lattice-biglm-faster-decoder.h"
// #include "decoder/lattice-faster-decoder.h"
// #include "decoder/lattice-faster-online-decoder.h"
// #include "decoder/lattice-incremental-decoder.h"
// #include "decoder/lattice-incremental-online-decoder.h"
// #include "decoder/lattice-simple-decoder.h"
// #include "decoder/simple-decoder.h"
// #include "decoder/training-graph-compiler.h"

// 特征提取相关。手机不做相关工作，用不到
// #include "feat/feature-common-inl.h"
// #include "feat/feature-common.h"
// #include "feat/feature-fbank.h"
// #include "feat/feature-functions.h"
// #include "feat/feature-mfcc.h"
// #include "feat/feature-plp.h"
// #include "feat/feature-spectrogram.h"
// #include "feat/feature-window.h"
// #include "feat/mel-computations.h"
// #include "feat/online-feature.h"
// #include "feat/pitch-functions.h"
// #include "feat/resample.h"
// #include "feat/signal.h"
// #include "feat/wave-reader.h"

// OpenFST 有限自动机 相关
// #include "fst/accumulator.h"
// #include "fst/add-on.h"
// #include "fst/arc-arena.h"
// #include "fst/arc-map.h"
// #include "fst/arc.h"
// #include "fst/arcfilter.h"
// #include "fst/arcsort.h"
// #include "fst/bi-table.h"
// #include "fst/cache.h"
// #include "fst/closure.h"
// #include "fst/compact-fst.h"
// #include "fst/compat.h"
// #include "fst/complement.h"
// #include "fst/compose-filter.h"
// #include "fst/compose.h"
```

```
// #include "fst/concat.h"
// #include "fst/config.h"
// #include "fst/connect.h"
// #include "fst/const-fst.h"
// #include "fst/determinize.h"
// #include "fst/dfs-visit.h"
// #include "fst/difference.h"
// #include "fst/disambiguate.h"
// #include "fst/edit-fst.h"
// #include "fst/encode.h"
// #include "fst/epsnormalize.h"
// #include "fst/equal.h"
// #include "fst/equivalent.h"
// #include "fst/expanded-fst.h"
// #include "fst/expectation-weight.h"
// #include "fst/factor-weight.h"
// #include "fst/filter-state.h"
// #include "fst/flags.h"
// #include "fst/float-weight.h"
// #include "fst/fst-decl.h"
// #include "fst/fst.h"
// #include "fst/fstlib.h"
// #include "fst/generic-register.h"
// #include "fst/heap.h"
// #include "fst/icu.h"
// #include "fst/intersect.h"
// #include "fst/interval-set.h"
// #include "fst/invert.h"
// #include "fst/isomorphic.h"
// #include "fst/label-reachable.h"
// #include "fst/lexicographic-weight.h"
// #include "fst/lock.h"
// #include "fst/log.h"
// #include "fst/lookahead-filter.h"
// #include "fst/lookahead-matcher.h"
// #include "fst/map.h"
// #include "fst/mapped-file.h"
// #include "fst/matcher-fst.h"
// #include "fst/matcher.h"
// #include "fst/memory.h"
// #include "fst/minimize.h"
// #include "fst/mutable-fst.h"
// #include "fst/pair-weight.h"
// #include "fst/partition.h"
// #include "fst/power-weight.h"
// #include "fst/product-weight.h"
// #include "fst/project.h"
// #include "fst/properties.h"
// #include "fst/prune.h"
// #include "fst/push.h"
```

```
// #include "fst/queue.h"
// #include "fst/randequivalent.h"
// #include "fst/randgen.h"
// #include "fst/rational.h"
// #include "fst/register.h"
// #include "fst/relabel.h"
// #include "fst/replace-util.h"
// #include "fst/replace.h"
// #include "fst/reverse.h"
// #include "fst/reweight.h"
// #include "fst/rmepsilon.h"
// #include "fst/rmfinalepsilon.h"
// #include "fst/set-weight.h"
// #include "fst/shortest-distance.h"
// #include "fst/shortest-path.h"
// #include "fst/signed-log-weight.h"
// #include "fst/sparse-power-weight.h"
// #include "fst/sparse-tuple-weight.h"
// #include "fst/state-map.h"
// #include "fst/state-reachable.h"
// #include "fst/state-table.h"
// #include "fst/statesort.h"
// #include "fst/string-weight.h"
// #include "fst/string.h"
// #include "fst/symbol-table-ops.h"
// #include "fst/symbol-table.h"
// #include "fst/synchronize.h"
// #include "fst/test-properties.h"
// #include "fst/topsort.h"
// #include "fst/tuple-weight.h"
// #include "fst/types.h"
// #include "fst/union-find.h"
// #include "fst/union-weight.h"
// #include "fst/union.h"
// #include "fst/util.h"
// #include "fst/vector-fst.h"
// #include "fst/verify.h"
// #include "fst/visit.h"
// #include "fst/weight.h"
// --openfst扩展
// #include "fstext/context-fst.h"
// #include "fstext/deterministic-fst-inl.h"
// #include "fstext/deterministic-fst.h"
// #include "fstext/determinize-lattice-inl.h"
// #include "fstext/determinize-lattice.h"
// #include "fstext/determinize-star-inl.h"
// #include "fstext/determinize-star.h"
// #include "fstext/epsilon-property-inl.h"
// #include "fstext/epsilon-property.h"
// #include "fstext/factor-inl.h"
```

```
// #include "fstext/factor.h"
// #include "fstext/fst-test-utils.h"
// #include "fstext/fstext-lib.h"
// #include "fstext/fstext-utils-inl.h"
// #include "fstext/fstext-utils.h"
// #include "fstext/grammar-context-fst.h"
// #include "fstext/kaldi-fst-io-inl.h"
// #include "fstext/kaldi-fst-io.h"
// #include "fstext/lattice-utils-inl.h"
// #include "fstext/lattice-utils.h"
// #include "fstext/lattice-weight.h"
// #include "fstext/pre-determinize-inl.h"
// #include "fstext/pre-determinize.h"
// #include "fstext/prune-special-inl.h"
// #include "fstext/prune-special.h"
// #include "fstext/push-special.h"
// #include "fstext/rand-fst.h"
// #include "fstext/remove-eps-local-inl.h"
// #include "fstext/remove-eps-local.h"
// #include "fstext/table-matcher.h"
// #include "fstext/trivial-factor-weight.h"

// GMM (对角阵高斯混合) 模型基础类
// #include "gmm/am-diag-gmm.h"
// #include "gmm/decodable-am-diag-gmm.h"
// #include "gmm/diag-gmm-inl.h"
// #include "gmm/diag-gmm-normal.h"
// #include "gmm/diag-gmm.h"
// #include "gmm/ebw-diag-gmm.h"
// #include "gmm/full-gmm-inl.h"
// #include "gmm/full-gmm-normal.h"
// #include "gmm/full-gmm.h"
// #include "gmm/indirect-diff-diag-gmm.h"
// #include "gmm/mle-am-diag-gmm.h"
// #include "gmm/mle-diag-gmm.h"
// #include "gmm/mle-full-gmm.h"
// #include "gmm/model-common.h"
// #include "gmm/model-test-common.h"

// GStreamer 解码器插件
// #include "gst-plugin/gst-audio-source.h"
// #include "gst-plugin/gst-online-gmm-decode-faster.h"

// HMM 模型相关
// #include "hmm/hmm-test-utils.h"
// #include "hmm/hmm-topology.h"
// #include "hmm/hmm-utils.h"
// #include "hmm/posterior.h"
// #include "hmm/transition-model.h"
// #include "hmm/tree-accu.h"
```

```
// 扩展接口
// #include "itf/clusterable-itf.h"
// #include "itf/context-dep-itf.h"
// #include "itf/decodable-itf.h"
// #include "itf/online-feature-itf.h"
// #include "itf/optimizable-itf.h"
// #include "itf/options-itf.h"
// #include "itf/transition-information.h"

// 声纹识别(说话人识别)
// #include "ivector/agglomerative-clustering.h"
// #include "ivector/ivector-extractor.h"
// #include "ivector/logistic-regression.h"
// #include "ivector/plda.h"
// #include "ivector/voice-activity-detection.h"

// Keyword Search 关键词搜索
// #include "kws/kaldi-kws.h"
// #include "kws/kws-functions.h"
// #include "kws/kws-scoring.h"

// 构建词图
// #include "lat/arctic-weight.h"
// #include "lat/compose-lattice-pruned.h"
// #include "lat/confidence.h"
// #include "lat/determinize-lattice-pruned.h"
// #include "lat/kaldi-lattice.h"
// #include "lat/lattice-functions-transition-model.h"
// #include "lat/lattice-functions.h"
// #include "lat/minimize-lattice.h"
// #include "lat/phone-align-lattice.h"
// #include "lat/push-lattice.h"
// #include "lat/sausages.h"
// #include "lat/word-align-lattice-lexicon.h"
// #include "lat/word-align-lattice.h"

// 自带的 language model 语言模型
// #include "lm/arpa-file-parser.h"
// #include "lm/arpa-lm-compiler.h"
// #include "lm/const-arpa-lm.h"
// #include "lm/kaldi-rnnlm.h"
// #include "lm/kenlm.h"
// #include "lm/mikolov-rnnlm-lib.h"

// 矩阵计算
// #include "matrix/cblas-wrappers.h"
// #include "matrix/compressed-matrix.h"
// #include "matrix/jama-eig.h"
// #include "matrix/jama-svd.h"
```

```
// #include "matrix/kaldi-blas.h"
// #include "matrix/kaldi-matrix-inl.h"
// #include "matrix/kaldi-matrix.h"
// #include "matrix/kaldi-vector-inl.h"
// #include "matrix/kaldi-vector.h"
// #include "matrix/matrix-common.h"
// #include "matrix/matrix-functions-inl.h"
// #include "matrix/matrix-functions.h"
// #include "matrix/matrix-lib.h"
// #include "matrix/numpy-array.h"
// #include "matrix/optimization.h"
// #include "matrix/packed-matrix.h"
// #include "matrix/sp-matrix-inl.h"
// #include "matrix/sp-matrix.h"
// #include "matrix/sparse-matrix.h"
// #include "matrix/srfft.h"
// #include "matrix/tp-matrix.h"

// nnetX, 训练方面的工作手机干不动。
// DNN 训练声学模型实现方式 1, 仅支持单 GPU, 维护者 Karel
// #include "nnet/nnet-activation.h"
// #include "nnet/nnet-affine-transform.h"
// #include "nnet/nnet-average-pooling-component.h"
// #include "nnet/nnet-blstm-projected.h"
// #include "nnet/nnet-component.h"
// #include "nnet/nnet-convolutional-component.h"
// #include "nnet/nnet-frame-pooling-component.h"
// #include "nnet/nnet-kl-hmm.h"
// #include "nnet/nnet-linear-transform.h"
// #include "nnet/nnet-loss.h"
// #include "nnet/nnet-lstm-projected.h"
// #include "nnet/nnet-matrix-buffer.h"
// #include "nnet/nnet-max-pooling-component.h"
// #include "nnet/nnet-multibasis-component.h"
// #include "nnet/nnet-nnet.h"
// #include "nnet/nnet-parallel-component.h"
// #include "nnet/nnet-parametric-relu.h"
// #include "nnet/nnet-pdf-prior.h"
// #include "nnet/nnet-randomizer.h"
// #include "nnet/nnet-rbm.h"
// #include "nnet/nnet-recurrent.h"
// #include "nnet/nnet-sentence-averaging-component.h"
// #include "nnet/nnet-trnopts.h"
// #include "nnet/nnet-utils.h"
// #include "nnet/nnet-various.h"

// DNN 训练声学模型实现方式 2, 支持多 GPU、CPU, 维护者 Daniel
// #include "nnet2/am-nnet.h"
// #include "nnet2/combine-nnet-a.h"
// #include "nnet2/combine-nnet-fast.h"
```

```
// #include "nnet2/combine-nnet.h"
// #include "nnet2/decodable-am-nnet.h"
// #include "nnet2/get-feature-transform.h"
// #include "nnet2/mixup-nnet.h"
// #include "nnet2/nnet-component.h"
// #include "nnet2/nnet-compute-discriminative-parallel.h"
// #include "nnet2/nnet-compute-discriminative.h"
// #include "nnet2/nnet-compute-online.h"
// #include "nnet2/nnet-compute.h"
// #include "nnet2/nnet-example-functions.h"
// #include "nnet2/nnet-example.h"
// #include "nnet2/nnet-fix.h"
// #include "nnet2/nnet-functions.h"
// #include "nnet2/nnet-limit-rank.h"
// #include "nnet2/nnet-nnet.h"
// #include "nnet2/nnet-precondition-online.h"
// #include "nnet2/nnet-precondition.h"
// #include "nnet2/nnet-stats.h"
// #include "nnet2/nnet-update-parallel.h"
// #include "nnet2/nnet-update.h"
// #include "nnet2/online-nnet2-decodable.h"
// #include "nnet2/rescale-nnet.h"
// #include "nnet2/shrink-nnet.h"
// #include "nnet2/train-nnet-ensemble.h"
// #include "nnet2/train-nnet.h"
// #include "nnet2/widen-nnet.h"

// DNN 训练声学模型实现方式 3, nnet2 的改进, 维护者 Daniel
// #include "nnet3/am-nnet-simple.h"
// #include "nnet3/attention.h"
// #include "nnet3/convolution.h"
// #include "nnet3/decodable-batch-looped.h"
// #include "nnet3/decodable-online-looped.h"
// #include "nnet3/decodable-simple-looped.h"
// #include "nnet3/discriminative-supervision.h"
// #include "nnet3/discriminative-training.h"
// #include "nnet3/natural-gradient-online.h"
// #include "nnet3/nnet-am-decodable-simple.h"
// #include "nnet3/nnet-analyze.h"
// #include "nnet3/nnet-attention-component.h"
// #include "nnet3/nnet-batch-compute.h"
// #include "nnet3/nnet-chain-diagnostics.h"
// #include "nnet3/nnet-chain-diagnostics2.h"
// #include "nnet3/nnet-chain-example.h"
// #include "nnet3/nnet-chain-training.h"
// #include "nnet3/nnet-chain-training2.h"
// #include "nnet3/nnet-combined-component.h"
// #include "nnet3/nnet-common.h"
// #include "nnet3/nnet-compile-looped.h"
// #include "nnet3/nnet-compile-utils.h"
```

```
// #include "nnet3/nnet-compile.h"
// #include "nnet3/nnet-component-itf.h"
// #include "nnet3/nnet-computation-graph.h"
// #include "nnet3/nnet-computation.h"
// #include "nnet3/nnet-compute.h"
// #include "nnet3/nnet-convolutional-component.h"
// #include "nnet3/nnet-descriptor.h"
// #include "nnet3/nnet-diagnostics.h"
// #include "nnet3/nnet-discriminative-diagnostics.h"
// #include "nnet3/nnet-discriminative-example.h"
// #include "nnet3/nnet-discriminative-training.h"
// #include "nnet3/nnet-example-utils.h"
// #include "nnet3/nnet-example.h"
// #include "nnet3/nnet-general-component.h"
// #include "nnet3/nnet-graph.h"
// #include "nnet3/nnet-nnet.h"
// #include "nnet3/nnet-normalize-component.h"
// #include "nnet3/nnet-optimize-utils.h"
// #include "nnet3/nnet-optimize.h"
// #include "nnet3/nnet-parse.h"
// #include "nnet3/nnet-simple-component.h"
// #include "nnet3/nnet-test-utils.h"
// #include "nnet3/nnet-training.h"
// #include "nnet3/nnet-utils.h"

// online 是用来识别的，手机上用到。
// 语音识别第 1 版 api，停更了。
// #include "online/online-audio-source.h"
// #include "online/online-decodable.h"
// #include "online/online-faster-decoder.h"
// #include "online/online-feat-input.h"
// #include "online/online-tcp-source.h"
// #include "online/onlinebin-util.h"

// 语音识别第 2 版，较复杂
// #include "online2/online-endpoint.h"
// #include "online2/online-feature-pipeline.h"
// #include "online2/online-gmm-decodable.h"
// #include "online2/online-gmm-decoding.h"
// #include "online2/online-ivector-feature.h"
// #include "online2/online-nnet2-decoding-threaded.h"
// #include "online2/online-nnet2-decoding.h"
// #include "online2/online-nnet2-feature-pipeline.h"
// #include "online2/online-nnet3-decoding.h"
// #include "online2/online-nnet3-incremental-decoding.h"
// #include "online2/online-nnet3-wake-word-faster-decoder.h"
// #include "online2/online-speex-wrapper.h"
// #include "online2/online-timing.h"
// #include "online2/onlinebin-util.h"
```

```
// RNN 语言模型训练。手机干不动
// #include "rnnlm/rnnlm-compute-state.h"
// #include "rnnlm/rnnlm-core-compute.h"
// #include "rnnlm/rnnlm-core-training.h"
// #include "rnnlm/rnnlm-embedding-training.h"
// #include "rnnlm/rnnlm-example-utils.h"
// #include "rnnlm/rnnlm-example.h"
// #include "rnnlm/rnnlm-lattice-rescoring.h"
// #include "rnnlm/rnnlm-test-utils.h"
// #include "rnnlm/rnnlm-training.h"
// #include "rnnlm/rnnlm-utils.h"
// #include "rnnlm/sampler.h"
// #include "rnnlm/sampling-lm-estimate.h"
// #include "rnnlm/sampling-lm.h"

// SGMM 模型训练。手机干不动
// #include "sgmm2/am-sgmm2-project.h"
// #include "sgmm2/am-sgmm2.h"
// #include "sgmm2/decodable-am-sgmm2.h"
// #include "sgmm2/estimate-am-sgmm2-ebw.h"
// #include "sgmm2/estimate-am-sgmm2.h"
// #include "sgmm2/fmllr-sgmm2.h"

// 基于 tensorflow RNN 语言模型训练。手机干不动
// #include "tfrnnlm/tensorflow-rnnlm.h"

// 特征转换相关
// #include "transform/basis-fmllr-diag-gmm.h"
// #include "transform/cmvn.h"
// #include "transform/compressed-transform-stats.h"
// #include "transform/decodable-am-diag-gmm-regtree.h"
// #include "transform/fmllr-diag-gmm.h"
// #include "transform/fmllr-raw.h"
// #include "transform/fmpe.h"
// #include "transform/lda-estimate.h"
// #include "transform/lvtln.h"
// #include "transform/mllt.h"
// #include "transform/regression-tree.h"
// #include "transform/regtree-fmllr-diag-gmm.h"
// #include "transform/regtree-mllr-diag-gmm.h"
// #include "transform/transform-common.h"

// 内部决策树相关
// #include "tree/build-tree-questions.h"
// #include "tree/build-tree-utils.h"
// #include "tree/build-tree.h"
// #include "tree/cluster-utils.h"
// #include "tree/clusterable-classes.h"
// #include "tree/context-dep.h"
// #include "tree/event-map.h"
```

```
// #include "tree/tree-renderer.h"

// 基础工具类
// #include "util/basic-filebuf.h"
// #include "util/common-utils.h"
// #include "util/const-integer-set-inl.h"
// #include "util/const-integer-set.h"
// #include "util/edit-distance-inl.h"
// #include "util/edit-distance.h"
// #include "util/hash-list-inl.h"
// #include "util/hash-list.h"
// #include "util/kaldi-cygwin-io-inl.h"
// #include "util/kaldi-holder-inl.h"
// #include "util/kaldi-holder.h"
// #include "util/kaldi-io-inl.h"
// #include "util/kaldi-io.h"
// #include "util/kaldi-pipebuf.h"
// #include "util/kaldi-semaphore.h"
// #include "util/kaldi-table-inl.h"
// #include "util/kaldi-table.h"
// #include "util/kaldi-thread.h"
// #include "util/parse-options.h"
// #include "util/simple-io-funcs.h"
// #include "util/simple-options.h"
// #include "util/stl-utils.h"
// #include "util/table-types.h"
// #include "util/text-utils.h"

#endif //KALDIANDROID_KALDI_H
```

5.3.3.CMakeLists.txt

这个文件的写法很多，官方教程 https://cmake.org/cmake/help/latest/guide/tutorial/index.html 。

```
cmake_minimum_required(VERSION 3.4.1)

# ${CMAKE_SOURCE_DIR} 即本 CMakeLists.txt 文件所在目录
# 设置一个路径变量 KALDI_DIR, 对应与 CMakeLists.txt 同级的 ./kaldi
set(KALDI_DIR ${CMAKE_SOURCE_DIR}/kaldi)

# ${ANDROID_ABI} 对应 build.gradle 文件 android / defaultConfig /  ndk / abiFilters 的芯片架构
add_library(libkaldi-base SHARED IMPORTED)
set_target_properties(libkaldi-base PROPERTIES IMPORTED_LOCATION
      ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-base.so)

add_library(libkaldi-chain SHARED IMPORTED)
set_target_properties(libkaldi-chain PROPERTIES IMPORTED_LOCATION
      ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-chain.so)

add_library(libkaldi-decoder SHARED IMPORTED)
```

```
set_target_properties(libkaldi-decoder PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-decoder.so)

add_library(libkaldi-feat SHARED IMPORTED)
set_target_properties(libkaldi-feat PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-feat.so)

add_library(libkaldi-fstext SHARED IMPORTED)
set_target_properties(libkaldi-fstext PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-fstext.so)

add_library(libkaldi-gmm SHARED IMPORTED)
set_target_properties(libkaldi-gmm PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-gmm.so)

add_library(libkaldi-hmm SHARED IMPORTED)
set_target_properties(libkaldi-hmm PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-hmm.so)

add_library(libkaldi-ivector SHARED IMPORTED)
set_target_properties(libkaldi-ivector PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-ivector.so)

add_library(libkaldi-kws SHARED IMPORTED)
set_target_properties(libkaldi-kws PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-kws.so)

add_library(libkaldi-lat SHARED IMPORTED)
set_target_properties(libkaldi-lat PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-lat.so)

add_library(libkaldi-lm SHARED IMPORTED)
set_target_properties(libkaldi-lm PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-lm.so)

add_library(libkaldi-matrix SHARED IMPORTED)
set_target_properties(libkaldi-matrix PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-matrix.so)

add_library(libkaldi-nnet SHARED IMPORTED)
set_target_properties(libkaldi-nnet PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-nnet.so)

add_library(libkaldi-nnet2 SHARED IMPORTED)
set_target_properties(libkaldi-nnet2 PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-nnet2.so)

add_library(libkaldi-nnet3 SHARED IMPORTED)
set_target_properties(libkaldi-nnet3 PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-nnet3.so)
```

```
add_library(libkaldi-online2 SHARED IMPORTED)
set_target_properties(libkaldi-online2 PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-online2.so)

add_library(libkaldi-rnnlm SHARED IMPORTED)
set_target_properties(libkaldi-rnnlm PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-rnnlm.so)

add_library(libkaldi-sgmm2 SHARED IMPORTED)
set_target_properties(libkaldi-sgmm2 PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-sgmm2.so)

add_library(libkaldi-transform SHARED IMPORTED)
set_target_properties(libkaldi-transform PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-transform.so)

add_library(libkaldi-tree SHARED IMPORTED)
set_target_properties(libkaldi-tree PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-tree.so)

add_library(libkaldi-util SHARED IMPORTED)
set_target_properties(libkaldi-util PROPERTIES IMPORTED_LOCATION
       ${KALDI_DIR}/lib/${ANDROID_ABI}/libkaldi-util.so)

# 自己编写的本地方法类
add_library(KaldiUtil SHARED
          KaldiUtil.cpp) # 相对 CMakeLists.txt 的路径

target_include_directories(KaldiUtil PRIVATE
       ${KALDI_DIR}/head) # 【src/main/cpp/kaldi/head】，自定义本地方法中 include 头文件 拼接的路径

# build application's shared lib
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")

# 最终编译的 so 文件路径：项目/app/build/intermediates/merged_native_libs/debug/out/lib/arm64-v8a
target_link_libraries(KaldiUtil
                    android
                   libkaldi-base # 对应本例演示用的 base/kaldi-math.h
                 # libkaldi-chain
                 # libkaldi-decoder
                 # libkaldi-feat
                 # libkaldi-fstext
                 # libkaldi-gmm
                 # libkaldi-hmm
                 # libkaldi-ivector
                 # libkaldi-kws
                 # libkaldi-lat
                 # libkaldi-lm
                 # libkaldi-matrix
```

```
                # libkaldi-nnet
                # libkaldi-nnet2
                # libkaldi-nnet3
                # libkaldi-online2
                # libkaldi-rnnlm
                # libkaldi-sgmm2
                # libkaldi-transform
                # libkaldi-tree
                # libkaldi-util
                  log)
```

5.3.4.build.gradle

添加 cmake 相关项。

```
plugins {
    id 'com.android.application'
}

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.cuiweiyou.kaldiandroid"
        minSdkVersion 26 // 【对应编译 so 时的 api 版本】
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        externalNativeBuild {
            cmake {
                arguments '-DANDROID_STL=c++_shared' // 【解决 UnsatisfiedLinkError: dlopen failed: library "libc++_shared.so" not found】
            }
        }

        ndk {
            abiFilters "arm64-v8a" // 【对应本手机的芯片，及 so 的架构】
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
```

```
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }

    externalNativeBuild {
        cmake {
            path 'src/main/cpp/CMakeLists.txt' // 【JNI 入口】
        }
    }
}

dependencies {
    // 略
}
```

5.3.5.运行测试

进行前几步骤时，cpp 代码通常会报红，可以时不时的点击菜单栏 Build 下的“Make Project”。

在 Activity 中调用一下：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        double v = new KaldiUtil().KaldiMathLogAdd(1, 2);
        Log.e("ard", "这里是 java。kaldi 计算的结果： " + v);
    }
}
```

日志：

```
ylist, reflection)
cuiweiyou.kaldiandroid W/ou.kaldiandroi: Accessing hidden method Landroid/view/ViewGroup;->m
cuiweiyou.kaldiandroid W/ou.kaldiandroi: Accessing hidden method Landroid/widget/TextView;->
; (light greylist, linking)
cuiweiyou.kaldiandroid E/System.out: 进入JNI的范畴了
cuiweiyou.kaldiandroid E/System.out: 调用kaldi的函数
cuiweiyou.kaldiandroid E/System.out: kaldi计算结束，返回java
cuiweiyou.kaldiandroid E/ard: 这里是java。kaldi计算的结果：2.313261687518223
cuiweiyou.kaldiandroid I/Adreno: QUALCOMM build                   : ce8a911, I385ac5a262
 03/28/19
 EV031.25.03.01

 refs/tags/AU_LINUX_ANDROID_LA.UM.7.2.R1.09.00.00.442.052
 NONE
 NOTHING
cuiweiyou.kaldiandroid I/Adreno: Build Config                    : S L 6.0.7 AArch64
cuiweiyou.kaldiandroid D/vndksupport: Loading /vendor/lib64/hw/gralloc.sdm660.so from curren
```

- end