Androidで簡単JNI

～JNIを使って、HelloWorld～

日本アンドロイドの会
稲葉 敏一

はじめに

今回は、書籍やインターネット上にある情報をまとめた基本的な内容です。すでに、公開されている技術資料の内容と重複する内容も含まれています。それらも含め「Androidで簡単JNI」の視点でまとめたものです。

目次

1. JNIとは？
2. Android NDKとは？
3. javahによるヘッダファイル作成
4. ソースコード作成
5. soファイル(shared library)作成
6. HelloWorldの表示

1.JNIとは？(基本)

JNI(Java Navite Interface)

Java言語で開発されたプログラムからネイティブコードのプログラムを利用可能とする仕組み。

※JNIはAndroid特有の仕組みではない。

1.JNIとは？(なぜJNIを使うか？)

- 実行速度が速い
  - どうしてもJavaは遅い。
    ネイティブコードの使用が可能。
    (但しオーバヘッドに要注意)

- 今までの資産が流用可能
  - C/C++で作成したアルゴリズムなど
    Javaに書き換える必要がない。

1.JNIとは？(階層)

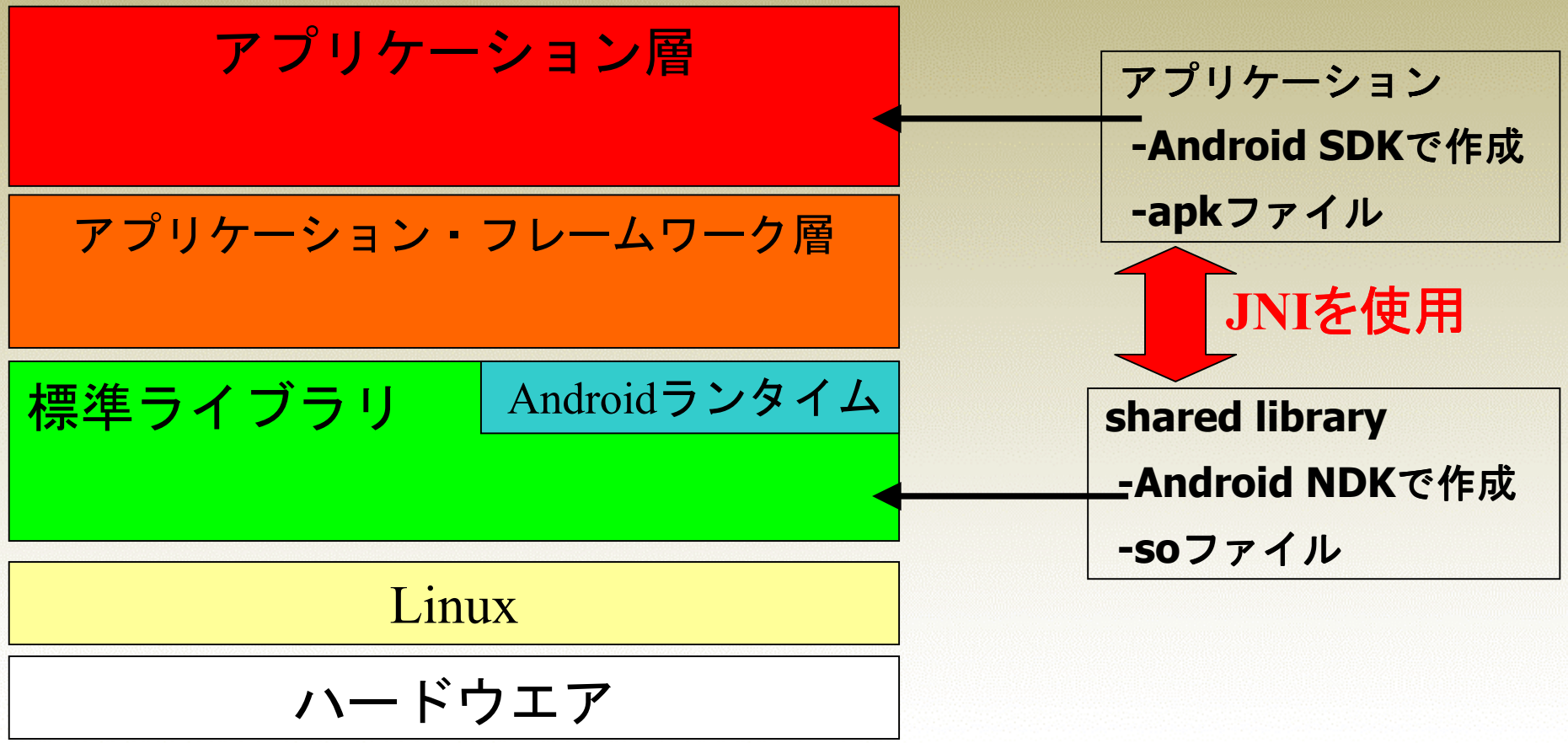


図1-1、Androidの階層におけるJNIの位置づけ

2.Android NDKとは？

Android NDK(Naitive Development Kit)

・Android SDKでは、Java言語にて
コーディングしたものをコンパイルし、
apkファイルを生成する。

・Android NDKでは、C/C++言語にて
コーディングしたものをコンパイルし、
soファイル(shared library)を生成する。

2.Android NDK(必要ツール)

・Android NDK
（今回：バージョン 1.6 r1）
・Android SDK

・cygwin
UNIX系OSのGNUプロジェクトによる
開発ツール郡をWindows環境に移植したもの
※フルインストールを推奨

2.Android NDK(Install)

```
Toshikazu@VAIO-P[5048] /cygdrive/c/user/develop/android/android-ndk-1.6_r1
check-awk.awk  gmsl          platforms  toolchains
core           host-setup.sh  prebuilt   tools
Toshikazu@VAIO-P$ ./host-setup.sh
Please run this script from the top-level NDK directory as in:
   cd $NDKROOT
   build/host-setup.sh
Toshikazu@VAIO-P$ ls
check-awk.awk  gmsl          platforms  toolchains
core           host-setup.sh  prebuilt   tools
Toshikazu@VAIO-P$ cd ..
Toshikazu@VAIO-P$ ls
GNUmakefile  README.TXT  apps  build  docs
Toshikazu@VAIO-P$ ./build/host-setup.sh
Checking host development environment.
NDK Root    : /cygdrive/c/user/develop/android/android-ndk-1.6_r1
GNU Make    : make (version 3.81)
Awk         : awk
Platform    : windows
Generate    : out/host/config.mk
Toolchain   : Checking for arm-eabi-4.2.1 prebuilt binaries

Host setup complete. Please read docs/OVERVIEW.TXT if you don't know what to do.


Toshikazu@VAIO-P$
```

buildフォルダ以下にて

./host-setup.shを実行

NDKROOTにて、実行せよとのこと。

フォルダ１つあがって

./build/host-setup.shを実行

必要なツール等チェックされ

「Host setup complete」が表示されたらOK！

2.Android NDK(環境設定)

・Javaソース(アプリ)は
　「apps」ディレクトリ以下に置く。
・Cソース(JNI)は、
　「sources」ディレクトリ以下に置く。

※シンボリックリンクで対応。
　lnコマンドにて設定可能
(UNIX系OSのファイルシステムの機能。
ファイルやディレクトリに別名を設定し、
ユーザがその名前をファイル本体と
同様に扱えるようにする仕組み。)

2.Android NDK(環境設定)

← のフォルダへ
シンボリックリンクを設定

apps

sources

JNI

作成するJNIヘッダとネイティブソース(Cソース)を、管理を一元化したいために「jni」フォルダを作成し、そこへ格納

【補足】上記のように、NDK 1.6 r1環境でもNDKROOTに[sources]フォルダを作成しても問題なし。
基本的には以下のフォルダ構成となっている。(同日の勉強会の日高さんの説明により気づきました)
●旧android-ndk-1.5_r1¥sources¥<projectname>
●新android-ndk-1.6_r1¥app¥<projectname>¥project¥jni

2.Android NDK(環境設定)

```
Toshikazu@VAIO-P[1872] /cygdrive/c/user/develop/android/android-ndk-1.6_r1/apps
Toshikazu@VAIO-P$
Toshikazu@VAIO-P$
Toshikazu@VAIO-P$ ls
AndroidManifest.xml  assets  bin  default.properties  gen  jni  res  src
Toshikazu@VAIO-P$
Toshikazu@VAIO-P$
Toshikazu@VAIO-P$ cd /cygdrive/c/user/develop/android/android-ndk-1.6_r1/
Toshikazu@VAIO-P$ ls
GNUmakefile  README.TXT  apps  build  docs  out  sources
Toshikazu@VAIO-P$ cd sources/
Toshikazu@VAIO-P$
Toshikazu@VAIO-P$
Toshikazu@VAIO-P$ ls
Toshikazu@VAIO-P$ pwd
/cygdrive/c/user/develop/android/android-ndk-1.6_r1/sources
Toshikazu@VAIO-P$ ln -s /cygdrive/c/user/develop/android_workspace/HelloAndroid
Jni/jni libHelloTestJNI
Toshikazu@VAIO-P$ ls
libHelloTestJNI
Toshikazu@VAIO-P$ cd ../apps/
Toshikazu@VAIO-P$ ln -s /cygdrive/c/user/develop/android_workspace/HelloAndroid
Jni HelloAndroidJni
Toshikazu@VAIO-P$ ls
HelloAndroidJni  hello-jni  san-angeles  two-libs  unit-tests
Toshikazu@VAIO-P$
```

ln –s 元フォルダ(リンクしたい先) シンボリックリンク名

←作成するライブラリ名でシンボリックリンク

←元のアプリのフォルダ名でシンボリックリンク

2.Android NDK (必要ファイル)

・アプリ：Application.mk
・JNI　　：Android.mk

上記の２つのファイルの作成が必要

※上記に記載されたルールに基づき
　コンパイルされる。
　サンプル等を確認し記載する。
　（記載ルールがあるため）

2.Android NDK (Application.mk)

```
APP_PROJECT_PATH := $(call my-dir) ←プロジェクトのパス
```

```
APP_MODULES      := HelloTestJNI   ←生成するモジュール名
```

例）**libHelloTestJNI.so**を作成する場合、
libと**.so**を抜いた「**HelloTestJNI**」を指定

2.Android NDK (Android.mk)

LOCAL_PATH := $(call my-dir)	←プロジェクトのパス
include $(CLEAR_VARS)	←初期化
LOCAL_SRC_FILES := HelloString.c	←コンパイル対象のソース
LOCAL_CFLAGS := -O3	←最適化オプション
LOCAL_CFLAGS += ¥	←コンパイルオプション
LOCAL_MODULE := libHelloTestJNI	←ターゲット名(作成する.soファイルの名前)
LOCAL_PRELINK_MODULE := false	←PRELINKはしない。
include $(BUILD_SHARED_LIBRARY)	←SHARED_LIBRARYとして作成

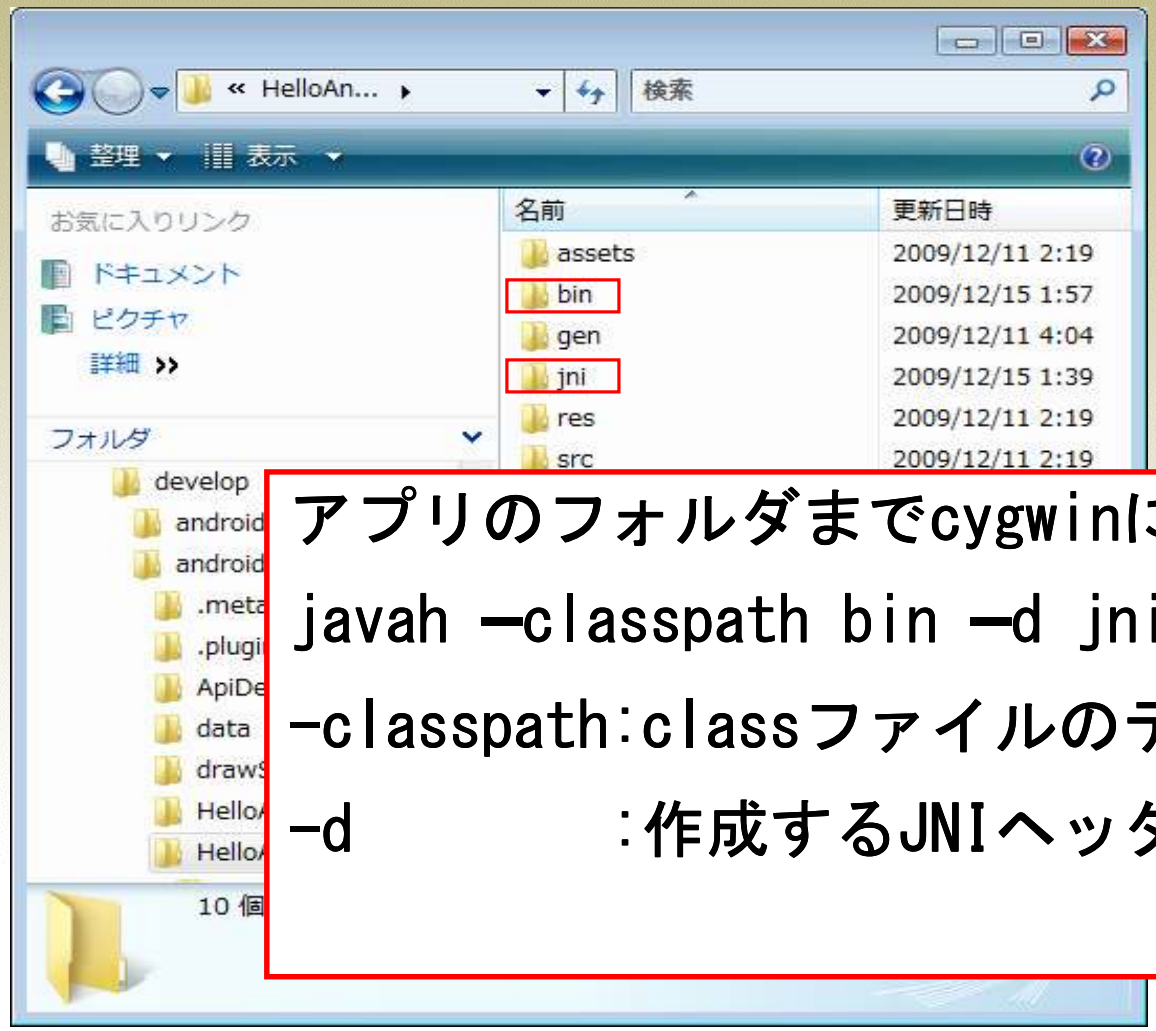
3. javahによるヘッダファイル作成

・javahコマンド
native宣言を含んだJavaのクラスファイルを指定し、JNIに必要なヘッダファイルを生成。

・javacコマンド
javaファイルよりclassファイルを生成。

※使用しているjavac、javahは、どのフォルダのものが使用されているか確認が必要！

3. javahコマンド詳細



アプリのフォルダまでcygwinにて移動。

```
javah -classpath bin -d jni inaba.test.JNItest
```

-classpath:classファイルのディレクトリ

-d　　:作成するJNIヘッダファイルの格納先

3. javahによるエラー

```
Toshikazu@VAIO-P[5928] /cygdrive/c/user/develop/android_workspace/HelloAndroidJni
AndTest.zip          ImageSwithcherTest.zip  VideoTest.zip      old
ApiDemos             NyARToolkitAndroid      WebcamBroadcaster  version.ini
HelloActivity        PullDown                data
HelloAndroidJni      PullDown.zip            drawSample
ImageSwithcherTest   VideoTest               eclipse.lnk
Toshikazu@VAIO-P$ cd HelloAndroidJni/
Toshikazu@VAIO-P$ ls
AndroidManifest.xml  assets  bin  default.properties  gen  jni  res  src
Toshikazu@VAIO-P$ javah -classpath bin -d jni inaba.test.JNItest
エラー:inaba.test.JNItest にアクセスできません。
クラスファイル bin¥inaba¥test¥JNItest.class は不正です。
クラスファイルのバージョン 50.0 は不正です。49.0 であるべきです。
削除するか、クラスパスの正しいサブディレクトリにあるかを確認してください。
com.sun.tools.javac.util.Abort
        at com.sun.tools.javac.comp
        at com.sun.tools.javadoc.Do
        at com.sun.tools.javadoc.Ro
        at com.sun.tools.javadoc.Ja
)
        at com.sun.tools.javadoc.St
        at com.sun.tools.javadoc.St
        at com.sun.tools.javadoc.Ma
        at com.sun.tools.javah.Main.main(Main.java:147)
javadoc: エラー - 致命的エラー
エラー 2 個
```

このようなエラーが出た場合、使用している。

javac、javahのバージョンを確認。

JDK5(jre1.5)へ統一。

※環境変数、cygwinのwhichコマンドなどで確認できる。(eclipseのフォルダjreが要注意！)

4.ソースコード作成

・アプリケーション(Javaソース)から
SHARED_LIBRARY(Cソース)を呼び出す

・SHARED_LIBRARYの関数が呼び出され、
アプリケーションに文字列を返す

・SHARED_LIBRARYから受け取った文字列を
アプリケーションにて画面に表示する

4.ソースコード作成

アプリケーション(抜粋)※文字列表示部分

```
public void onDraw(Canvas canvas)
{
super.onDraw(canvas);

   canvas.drawBitmap(bitmap, 0, 0, paint);

   String returnStr ;
   returnStr = JNItest.getHelloString();
   /* 文字列表示 */
   paint.setTextSize(32);
   paint.setColor(0xFFFF0000);/* 赤 */
   canvas.drawText(returnStr, 60, 170, paint);
}
```

4.ソースコード作成

アプリケーション(抜粋)※native宣言部分

```
package inaba.test;

import android.util.Log;

public class JNItest {

static {
   try {
        /* ライブラリを読み込み */
        System.load("libHelloTestJNI.so");
   } catch (UnsatisfiedLinkError e) {
        Log.d("HelloTestJNI", e.getMessage());
        Log.d("HelloTestJNI", "can't loadLibrary");
   }
}

public static native String getHelloString();
}
```

4.ソースコード作成

JNIヘッダ

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class inaba_test_JNItest */

#ifndef _Included_inaba_test_JNItest
#define _Included_inaba_test_JNItest
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     inaba_test_JNItest
 * Method:    getHelloString
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_inaba_test_JNItest_getHelloString
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

4.ソースコード作成

ネイティブコード(抜粋)

```
#include <stdio.h>
#include <stdlib.h>
#include "inaba_test_JNItest.h"

/*
 * Class:     inaba_test_JNItest
 * Method:    getHelloString
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL
  Java_inaba_test_JNItest_getHelloString
  (JNIEnv *jenv, jclass jobj){

  return (*jenv)->NewStringUTF(jenv,  "Hallo World");
}
```

4.ソースコード作成(補足)

・NewStringUTF
上記はJNIが定義している関数

※その他さまざまなJNIが提供している関数がある。それらを使用して実現したい機能を実装する。

～参考～

jtypeArray jinput

```
(*jenv)->GetArrayLength(jenv, jinput);
(*jenv)->GetTypeArrayElements(jenv, jinput, NULL);
```

Type は、Byte Int などが入り、型ごとに名前の違う関数が存在する

4.ソースコード(JNIでのつなぎ)

アプリ(宣言部分)

native `getHelloString()`

＝＝

Javah作成ヘッダ

`Java_inaba_test_JNItest_getHelloString`

※アプリのパッケージ名より関数名を自動生成

(同じものを指す)

アプリ(使用部分)

`JNItest.getHelloString();`

ネイディブにて実体を作成

```
Java_inaba_test_JNItest_getHelloString
 {
      /* 必要な処理 */
 }
```

JavaアプリからCコードの処理を実行

図4-1、naivite宣言(java)のメソッドとネイティブコード(Cコード)のつながり

5. soファイル(shared library)作成

- NDKのトップディレクトリより、apps以下のディレクトリ名を指定してmake実行

  make APP=HelloAndroidJni

- apps/libs/armeabi/libHelloTestJNI.so
  上記が作成される。
  (apk内にも同梱される。タイミングは
  apkを作成しなおしたときの様子)

5. soファイル(shared library)作成

```
Toshikazu@VAIO-P[5824] /cygdrive/c/user/develop/android/android-ndk-1.6_r1
Toshikazu@VAIO-P$ ls
GNUmakefile  README.TXT  apps  build  docs  out  sources
Toshikazu@VAIO-P$ make APP=HelloAndroidJni
Android NDK: Building for application 'HelloAndroidJni'
make: Nothing to be done for `all'.
Toshikazu@VAIO-P$ make APP=HelloAndroidJni
Android NDK: Building for application 'HelloAndroidJni'
make: Nothing to be done for `all'.
Toshikazu@VAIO-P$
Toshikazu@VAIO-P$
Toshikazu@VAIO-P$ ls
GNUmakefile  README.TXT  apps  build  docs  out  sources
Toshikazu@VAIO-P$ cd sources/
Toshikazu@VAIO-P$ ls
libHelloTestJNI
Toshikazu@VAIO-P$ cd ..
Toshikazu@VAIO-P$ make APP=HelloAndroidJni
Android NDK: Building for application 'HelloAndroidJni'
make: Nothing to be done for `all'.
Toshikazu@VAIO-P$ make APP=HelloAndroidJni
Android NDK: Building for application 'HelloAndroidJni'
Compile thumb  : HelloTestJNI <= apps/HelloAndroidJni/jni/HelloString.c
SharedLibrary  : libHelloTestJNI.so
Install        : libHelloTestJNI.so => apps/HelloAndroidJni/libs/armeabi
Toshikazu@VAIO-P$
```

対象のappsのフォルダ以下ではなく、NDKのトップより対象アプリのフォルダ名を指定

6. HelloWorldの表示

Javaアプリでは保持していなかった「Hello World」の文字列を取得し表示

6. HelloWorldの表示(補足)

・soファイルの置き場所は２箇所
-/system/lib
adb push 対象soファイル /system/lib
※エミュレータ起動後、adbの
pushコマンドにてアップ可能
-apkファイルの中に同梱
NDKで作成した場合、同梱される
(apkファイルは、zip形式にて生成)

※/system/libに同じものがあった場合
/system/libにあるものが優先的にロードされる

おわり

ご清聴ありがとうございました