NDK y JNI - Android

NDK - Android Native Development Kit

- Colección de herramientas dentro de Android SDK que permite la integración de código y librerías C/C++
- Java/Kotlin se compilan en bytecode pero el código C/C++ se compila directamente en código ejecutable nativo.
    + Eficiencia
    - Portabilidad.

NDK - Android Native Development Kit

Añade complejidad, aumentando además el tamaño de los APK y dificultando la depuración.

- Depende del desarrollador de Java escribir código seguro en C.
- Es posible que se produzcan sobrecargas y pérdidas de rendimiento
- Posibles problemas de seguridad en la memoria

Pero permite mayor eficiencia y facilita la ofuscación de código.

NDK - Android Native Development Kit

Para usar el NDK en Android necesitamos:

- CMake -> Herramienta de compilación para C/C++
- FFI (Foreign Functions Interface) ->
  -> JNI (Java Native Interface)
    - JNI es el mecanismo que utiliza JVM para integrarse con las funciones nativas.

FFI's

FFI (Foreign Functions Interface)

- Permiten llamar rutinas desde otro programa independientemente del lenguaje.
- La mayoría de los lenguajes modernos proporcionan esta característica de forma intuitiva
- Término originado a partir de LISP.

Existen diversas herramientas para JAVA:

- JNI, JNA, JNR, Panama

FFI's

Permiten:

- Acceder a funciones no disponibles en el lenguaje origen
- Utilizar bibliotecas nativas
- Acceder a funciones o programas del sistema operativo anfitrión
- Descarga de GPU y CPU (Cuda, OpenCL, OpenGL, Vulcan, DirectX...)
- Aritmética de precisión múltiple, multiplicaciones de matrices
- Aprendizaje profundo (Tensorflow, cuDNN, Blas...)

JNI

JNI - interfaz nativa de Java

Define una forma para que el código bytecode (escrito en Java o Kotlin) interactúe con el código nativo (escrito en C/C++).

JNI define dos estructuras de datos clave: "JavaVM" y "JNIEnv".

En esencia, ambas son punteros a punteros de tablas de funciones. (En la versión de C++, son clases con un puntero a una tabla de funciones y a una función de miembro para cada función de JNI que direcciona de manera indirecta por la tabla).

En teoría, se pueden tener múltiples JavaVM por proceso, pero Android permite solo una.

La estructura JNIEnv proporciona la mayoría de las funciones de JNI. Todas tus funciones nativas reciben una JNIEnv como primer argumento.

La JNIEnv se usa para almacenamiento local de subprocesos. Por esa razón, **no se puede compartir una JNIEnv entre subprocesos**. Si un fragmento de código no tiene otra forma de obtener JNIEnv, debes compartir JavaVM y usar `GetEnv` para descubrir la JNIEnv del subproceso.

JNI

Ejemplo función getpid()

```java
class Main {
    public static void main(String[] args) {
        System.out.println("my process id: " + getpid());
    }

    private static native int getpid();
}
```

Generate header

```
javac -h . Main.java
```

Main.h

Main.c

Implement C class

```c
#include <unistd.h>
#include "Main.h"

JNIEXPORT jint JNICALL Java_Main_getpid
  (JNIEnv *env, jclass cls) {
  // call the actual C function to get the process id!
  return getpid();
}
```

Compile C code to dynamic lib

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

```
java Main.java
```

JNI

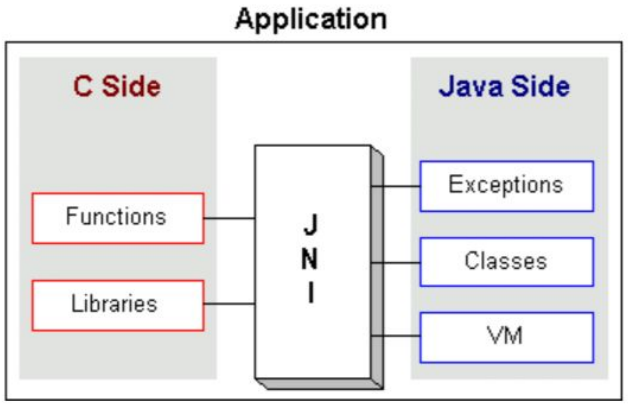
Proporciona métodos para interactuar con objetos de Java

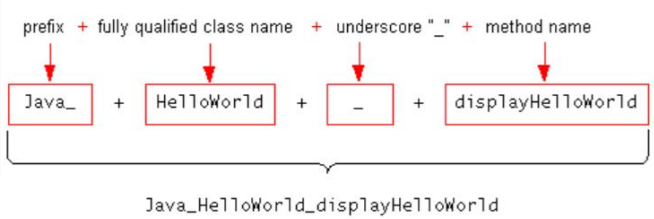
```java
package com.baeldung.jni;

public class UserData {

    public String name;
    public double balance;

    public String getUserInfo() {
        return "[name]=" + name + ", [balance]=" + balance;
    }
}
```

```java
...
public native UserData createUser(String name, double balance);

public native String printUserData(UserData user);
```

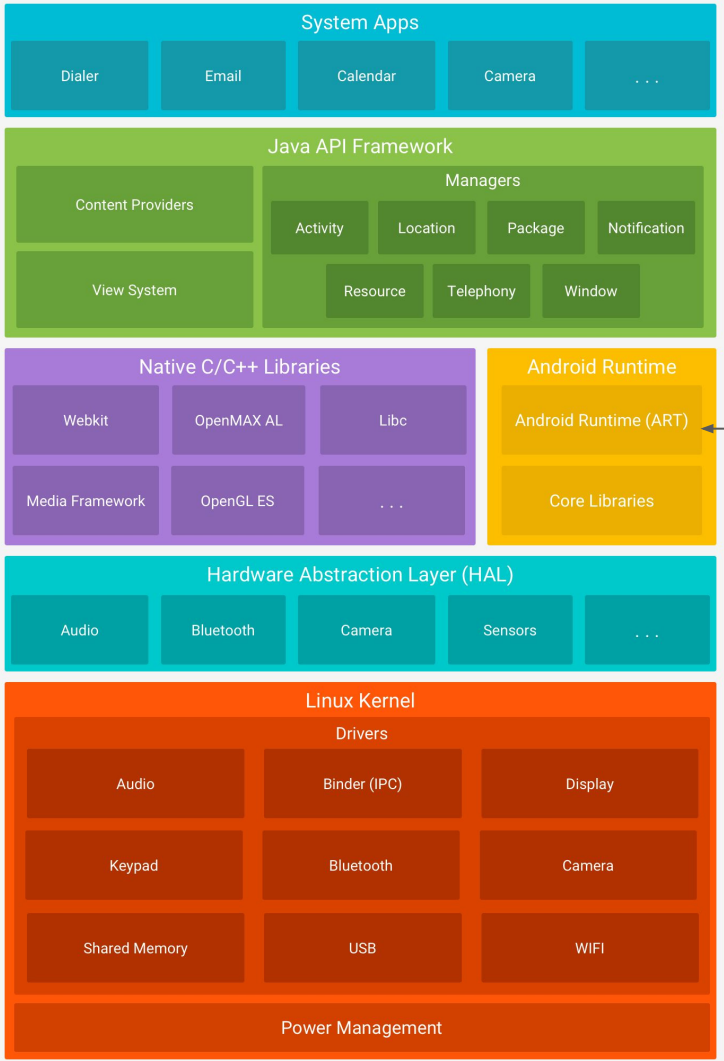
JNI

```
JNIEXPORT jobject JNICALL Java_com_baeldung_jni_ExampleObjectsJNI_createUser
  (JNIEnv *env, jobject thisObject, jstring name, jdouble balance) {

    // Create the object of the class UserData
    jclass userDataClass = env->FindClass("com/baeldung/jni/UserData");
    jobject newUserData = env->AllocObject(userDataClass);

    // Get the UserData fields to be set
    jfieldID nameField = env->GetFieldID(userDataClass , "name", "Ljava/lang/String;");
    jfieldID balanceField = env->GetFieldID(userDataClass , "balance", "D");

    env->SetObjectField(newUserData, nameField, name);
    env->SetDoubleField(newUserData, balanceField, balance);

    return newUserData;
}

JNIEXPORT jstring JNICALL Java_com_baeldung_jni_ExampleObjectsJNI_printUserData
  (JNIEnv *env, jobject thisObject, jobject userData) {

    // Find the id of the Java method to be called
    jclass userDataClass=env->GetObjectClass(userData);
    jmethodID methodId=env->GetMethodID(userDataClass, "getUserInfo", "()Ljava/lang/String;");

    jstring result = (jstring)env->CallObjectMethod(userData, methodId);
    return result;
}
```

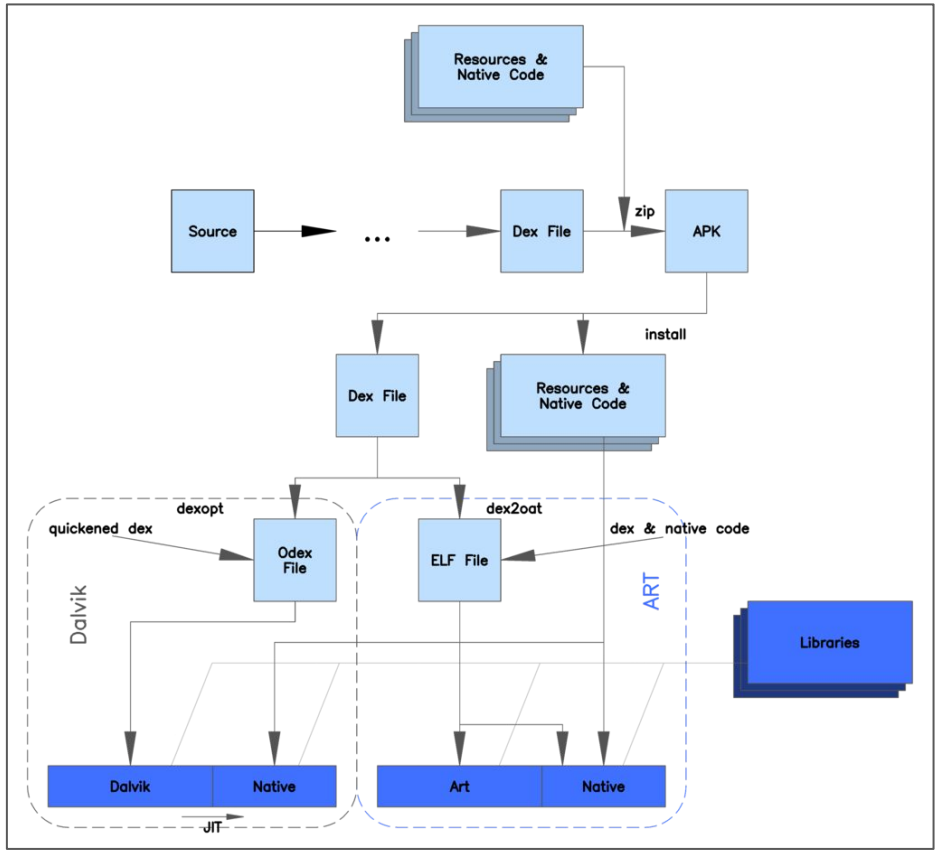




Arquitectura Android



"Dalvik" antes de la versión 5.0 (Lollipop)



Documentación

- Instalar NDK
  https://developer.android.com/studio/projects/install-ndk?hl=es-419
- Usar NDK
  https://developer.android.com/ndk/guides?hl=es-419
- Sugerencias JNI
  https://developer.android.com/training/articles/perf-jni?hl=es-419#java
- Tutorial NDK - Ejemplo Fibonacci (en Kotlin)
  https://www.techyourchance.com/android-ndk-tutorial/
- Ejemplo Hello-JNI
  https://developer.android.com/ndk/samples/sample_hellojni?hl=es-419
- Info JNI
  https://www.iitk.ac.in/esc101/05Aug/tutorial/native1.1/stepbystep/index.html
- Ejemplo JNI
  https://www.baeldung.com/jni