classes.dex

Ce fichier est le résultat de la compilation du code source.

La compilation se fait en deux temps:

- compilation en Byte-Code .class
- compilation en .dex

Les applications android sont généralement écrites en Java. La première phase de compilation est donc réalisée par 'javac'. Ensuite, le .class obtenu est transformé en code spécifique à la JVM d'android.

```
dx --dex --output=Bonjour.dex Bonjour.class
```

Le code obtenu est plus efficace qu'un Byte-code standard, mais il n'est plus portable et ne peut être facilement décompilé.

Exemple de décompilation:

```
iget v1, p0, Lfr/pythagorefd/apps/Bonjour;->next_block_y:I
add-int/lit8 v1, v1, 0x1e
iput v1, p0, Lfr/pythagorefd/apps/Bonjour;->next_block_y:I

sub-int v0, v1, v2
const/4 v1, 0x4
if-ge v0, v1, :cond_a
return v0
```

Ce code est modifiable et recompilable

DalvikVM

La JVM utilisée est une DalvikVM.

Cette JVM ne supporte pas awt, swing ni J2ME. Elle est optimisée pour les applications embarquées et pour l'exécution en plusieurs instances.

La gestion des tâches et la segmentation mémoire sont déléguées au système d'exploitation sous-jacent.

Le code obtenu après la compilation `javac`, est du byte-code. Ce dernier est transformé en `.dex` natif à l'aide de l'outil `platforms/android-x.x.x/tools/dx`

Cela permet de réduire fortement la consommation mémoire et d'obtenir de bonnes performances, tout en garantissant l'étanchéité des applications.

Paquets disponibles

Les paquets essentiels du JDK standard sont disponibles:

```
java.lang:     String, Thread, System, ...
java.sql:      Connection, ResultSet, ...
java.text:     DateFormat, ParsePosition, ...
java.io:       FileInputStream, PrintWriter, ...
java.security: KeyStore, Principal, PrivateKey, PublicKey, ...
java.net:      ServerSocket, HttpURLConnection, NetworkInterface, ...
java.nio:      Channels, Pipe, ...
java.util:     ArrayList, Set, Calendar, .zip, .regex, ...
```

Le SDK contient aussi:

```
org.apache.http: HttpServerConnection, HttpClient, HttpRequest, ...
org.json (JavaScript Object Notation): JSONObject, JSONArray, JSONTokener, ...
```

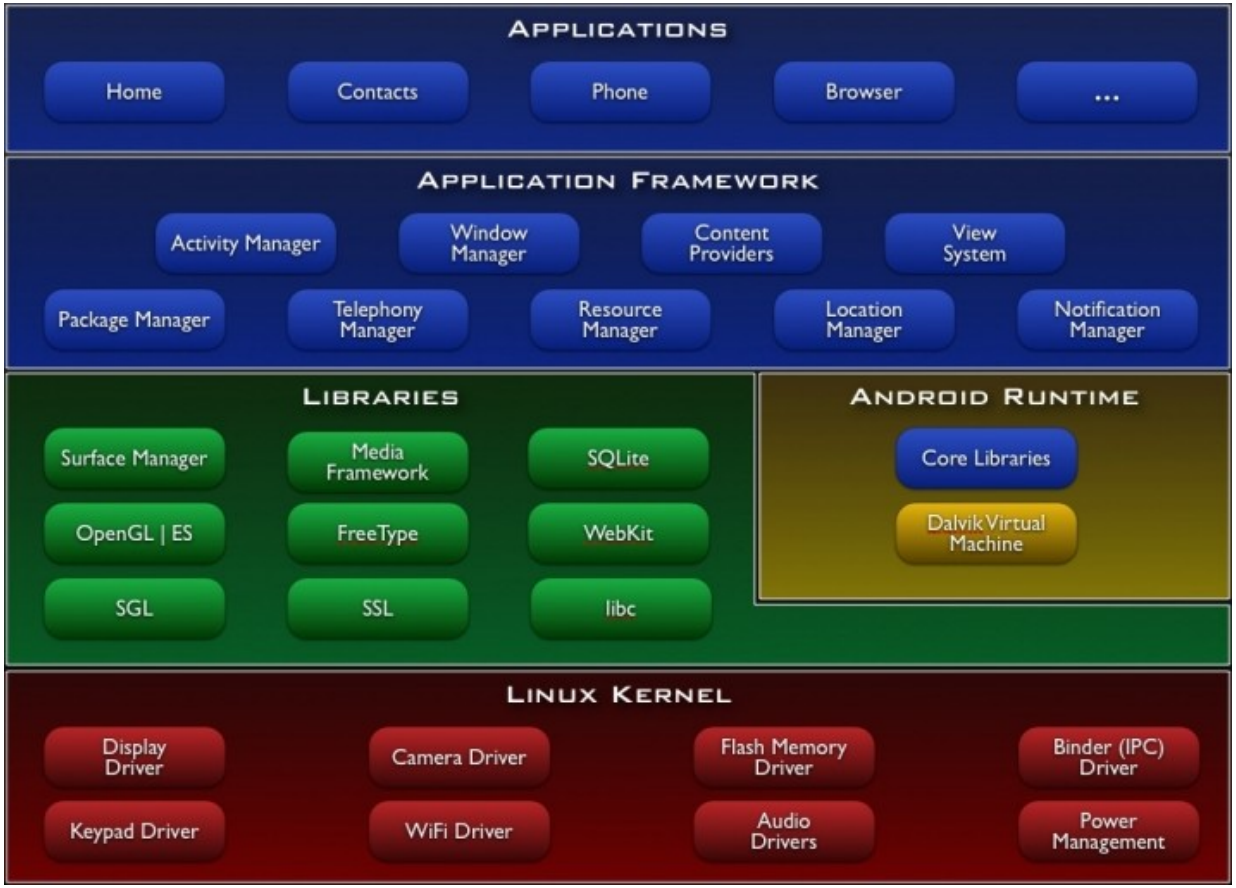
Ainsi que tous les sous-paquets spécifiques à Android:

dalvik.*, qui contiennent des outils pour gérer les classes .dex

et

android.*, qui contiennent tous les outils "Android": Activity, Service, View,

Architecture Android



Source: http://developer.android.com/images/system-architecture.jpg

Compilation avancée

Compilation native

Dans certains cas, il est souhaitable d'avoir un accès direct au langage natif du système d'exploitation.

Pour Android, il s'agit du langage C ou du C++

Ce code natif peut être compilé et installé sur la plateforme grâce au NDK. Dans ce cas, les bibliothèques natives sont stockées dans un répertoire lib.

L'accès natif peut se faire soit en accès direct, comme la création d'une nouvelle commande sur l'android, soit en accès applicatif au travers d'une application Java.

Dans le premier cas, l'application obtenue est en fait un code exécutable que l'on transfère par un "adb push".

Dans le second cas, il faut utiliser le mécanisme Java permettant d'appeler des méthodes natives depuis une classe: JNI.

Installation du NDK

Il faut télécharger le NDK depuis le site de Google.

Par exemple, pour télécharger la version 5:

```
http://dl.google.com/android/ndk/
```

Décompactez l'archive:

```
tar xzf android-ndk-r5-linux-x86.tar.bz2
```

Attention, ce ndk s'appuie sur la glibc 2.11. Il faut donc vous assurer que votre système est compatible:

```
$ /lib/libc.so.6
GNU C Library development release version 2.12.90, by Roland McGrath et al.
...
```

Si ce n'est pas le cas, vous pouvez mettre à niveau votre libc:

```
tar xjf glibc-2.12.2.tar.bz2
cd glibc-2.12.2
mkdir glibc-build
cd glibc-build/
cp /usr/lib/gcc/.../.../include/cpuid.h /usr/include
export CFLAGS="-march=i686 -O2"
../glibc-2.12.2/configure --prefix=/usr
make -j4 && make install
puis reboot de la machine
```

C natif

Pour commencer, nous allons tester les exemples fournis:

```
cd android-ndk-r5
tests/run-tests.sh --full
```

Il faut ensuite lancer un shell sur le périphérique:

```
$ adb shell
$ cd /data/local/ndk-tests
$ ls -l
-rwxr-xr-x shell    shell       37832 2010-12-07 17:47 test_gnustl_1
-rwxr-xr-x shell    shell        4840 2010-12-07 17:47 test_basic_rtti
-rwxr-xr-x shell    shell       37160 2010-12-07 17:47 test_basic_exceptions
$ ./test_gnustl_1
OK: Hello, world (with full C++ support) !
$ ./test_basic_rtti
OK: 'foo' is pointing to a Bar class instance.
$ ./test_basic_exceptions
OK: Exception raised: Memory allocation failure!
$
```

Exemple "Bonjour"

Edition

Dans un répertoire de base, "Bonjour" par exemple, créez un sous-répertoire "jni" et enregistrez y les fichiers suivants:

Fichier source bonjour.c

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

int main(void)
{
        printf("Bonjour tout le monde! \n");
        printf("uid=%d gid=%d pid=%d\n",getuid(),getgid(),getpid());
        return 0;
}
```

Fichier Andrdoid.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := Bonjour
LOCAL_SRC_FILES := bonjour.c
include $(BUILD_EXECUTABLE)
```

Compilation

```
make -f ~/domotique/android/android-ndk-r5/build/core/build-local.mk
```

Bonjour

Installation

```
adb shell mkdir /data/local/jni
adb push obj/local/armeabi/Bonjour /data/local/jni
```

Execution

```
adb shell
cd /data/local/jni
./Bonjour
```

JNI

JNI est un système permettant d'interfacer le code Java avec du code natif.

Le principe est de charger une bibliothèque à l'aide de System.loadLibrary et d'appeler ses méthodes comme si elles faisaient parties de la classe.

Les fonctions enregistrées dans la bibliothèque, proviennent d'une compilation de code écrit en C ou en C++.

Le mécanisme JNI permet de faire la liaison entre l'appel Java vers le code C en prenant en charge la transmission des arguments, ainsi que la transmission de la valeur de retour de la fonction.

Il est aussi possible d'utiliser les objets de la classe, directement depuis le code natif.

Exemple

Nous allons commencer par écrire une classe Java qui charge une bibliothèque "libtest.so".

```
package fr.pythagorefd.jni.bonjour;
public class Bonjour extends Activity {

    public native String  donneDate();
         /*
         on charge la bibliotheque de fonctions compilee avant
         Cette bibliotheque est nommee libtest.so
         et est rangee dans /data/data/fr.pythagorefd.jni.bonjour/lib/libtest.so
         */
    static {
                System.loadLibrary("test");
    }

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView  tv = new TextView(this);
        tv.setText( "date="+ donneDate() );
        setContentView(tv);
    }
}
```

Nous stockerons ce fichier dans un répertoire src du répertoire projet

Compilez le projet:

```
android_compile .
```

Exemple

Il faut ensuite écrire le code de la bibliothèque libtest.so

Nous avons besoin de la signature de la méthode telle que comprise par Java:

Par exemple:

```
public native String  donneDate()
```

va s'écrire en C:

```
JNIEXPORT jstring JNICALL Java_fr_pythagorefd_jni_bonjour_Bonjour_donneDate
                                                (JNIEnv *tthis, jobject)
```

L'obtention de la signature des méthodes java, peut être réalisé à l'aide de l'outil: javah

Signatures des fonctions

Création des entêtes:

```
cd bin/classes
javah -jni fr.pythagorefd.jni.bonjour.Bonjour
```

Visualisez le fichier obtenu

```
$ cat fr_pythagorefd_jni_bonjour_Bonjour.h

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

#ifndef _Included_fr_pythagorefd_jni_bonjour_Bonjour
#define _Included_fr_pythagorefd_jni_bonjour_Bonjour
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     fr_pythagorefd_jni_bonjour_Bonjour
 * Method:    donneDate
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_fr_pythagorefd_jni_bonjour_Bonjour_donneDate
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Code en C

Il reste à ajouter le code C retournant la date:

```
#include <string.h>
#include <jni.h>
/*
 * Class:     fr_fr_pythagorefd_jni_bonjour_Bonjour
 * Method:    donneDate
 * Signature: ()Ljava/lang/String;
 */


jstring Java_fr_pythagorefd_jni_bonjour_Bonjour_donneDate(JNIEnv *env, jobject tthis )
{
        long t=0;
        time(&t);
        return (*env)->NewStringUTF(env, ctime(&t) );
}
```

Ce fichier sera stocké dans un répertoire "jni" du projet

Compilation

Dans le répertoire jni du projet, ajoutez le fichier de description Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
LOCAL_MODULE    := fhutils
LOCAL_SRC_FILES := prog.c
include $(BUILD_SHARED_LIBRARY)
```

Ce fichier indique qu'il faut construire une bibliothèque dynamique et non plus un exécutable.

Puis compilez la bibliothèque

```
make -f ...../android-ndk-r5/build/core/build-local.mk
```

Compilez le projet et installez le sur la plateforme

```
android_compile . && android_install_app .
```