Librairie partagée sous Android

Nous allons proposer une solution technique permettant de partager un JAR d'un APK avec les autres APK installé sur le terminal Android.

Par Philippe PRADOS - 2011
www.prados.fr

Proposer un composant pour toutes les applications Android

La plupart des applications pour Android sont autonome. Elles se suffisent à elles-mêmes. Il arrive parfois que l'on souhaite proposer un composant applicatif, destinés à plusieurs applications. Il y a plusieurs approches pour faire cela. La plus simple consiste à proposer un simple Jar à intégrer dans chaque application.

Nous souhaitons proposer un composant plus complexe, portées par un APK et utilisés par plusieurs applications APK simultanément.

Le framework Android est conçu comme cela. Une application spécifique **system_app** est lancé au boot, puis la librairie **android.jar** s'occupe de communiquer avec, à l'aide des mécanismes d'invocation cross processus (AIDL).

Ne pouvant enrichir la librairie **android.jar**, nous devons procéder autrement.

Imaginons un composant qui expose une API pour vérifier la licence d'une application, ou pour proposer un mécanisme d'achat pendant l'exécution de l'application. Ces services sont délégués à la place de marché intégrée.

Notre composant expose des interfaces à l'aide d'un AIDL, un fichier de description d'interface, permettant l'invocation de méthodes entres deux applications (Cf. Hors Série Android).

Prenons l'exemple de l'AIDL suivant :

```
package org.checklicence.lib.shared;

interface ICheckLicence
{
        boolean checkLicence();
}
```

Eclipse va invoquer l'utilitaire **aidl** pour générer du code pour la partie cliente et la partie serveur. Nous obtenons dans le répertoire **/gen**, l'interface **org.checklicence.lib.shared.ICheckLicence**, avec la classe interne **Stub**, elle-même, avec la classe interne **Proxy**.

Stub s'occupe de l'implémentation du service coté serveur. **Proxy** s'occuper de la vision du service coté client.

La méthode **checkLicence()** est implémenté dans le composant applicatif à l'aide d'une classe héritant de **ICheckLicence.Stub**.

```
package org.checklicence;

import org.checklicence.lib.shared.ICheckLicence;

import android.os.RemoteException;
```

```
public class CheckLicenceImpl extends ICheckLicence.Stub
{

  @Override
  public boolean checkLicence() throws RemoteException
  {
    // TODO Vérifier la licence de l'utilisateur
    return true;
  }

}
```

Les classes générées par l'AIDL sont nécessaires au client et au serveur. Chaque client devra donc avoir ces classes. Comment dans ces conditions faire évoluer l'interface ? Si on ajoute une nouvelle méthode à l'interface, il est nécessaire de re-générer le code et de le distribuer à toute les applications. Ce n'est pas très évolutif !

Partage de la librairie cliente

La librairie cliente est fortement adhérente avec la partie serveur. Si le code du serveur évolue, la librairie cliente doit également évoluer. Oui mais voilà, tous les APK qui intègre cette librairie doivent alors être mis à jour ! C'est justement ce que nous souhaitons éviter.

Notre objectif et de publier la librairie cliente dans l'APK du composant, et de permettre à chaque application de la récupérer dynamiquement. Ainsi, une mise à jour de l'APK permettra d'adapter la librairie cliente de toutes les applications. Nous voulons en fait, une librairie partagée. Android ne propose pas cela. Qu'a cela ne tienne.

Pour cela, nous devons organiser le code pour que cela soit possible.

Nous allons découper notre projet en trois parties :

- La librairie d'interface (**CheckLicence-lib**). C'est une librairie pratiquement vide, permettant aux applications d'invoquer le service. La phase d'initialisation va récupérer dynamiquement l'implémentation des interfaces. Le code généré par l'AIDL n'est pas exposé à ce niveau.
- La librairie cliente d'implémentation (**CheckLicence-sharedlib**). Cette librairie propose une implémentation des interface de la librairie d'interface. Elle encapsule le code généré par l'AIDL. Elle sera empaquetée dans l'APK du composant applicatif.
- L'application APK offrant le service (**CheckLicence.apk**). Cette application propose des services accessibles à distance, par d'autres applications, et la libraire cliente d'invocation.

La librairie d'interface est une librairie la plus petite possible, ne possédant que des interfaces ou des classes abstraites. Cette librairie expose les API utilisés par les applications exploitant le composant. Comme il s'agit d'interface, il est facile de maintenir une compatibilité ascendante. L'ajout d'une méthode n'invalide pas le code existant.

Pour initialiser le composant, nous proposons alors une méthode statique primitive, permettant d'obtenir une implémentation de la librairie cliente.

```
package org.checklicence.lib;

import android.content.Context;

public abstract class CheckLicenceForMyMarket
{
  // API publique pour les applications
  public abstract boolean checkLicenceForMyMarket();

  static CheckLicenceForMyMarket getManager(Context context)
  {
    // TODO
  }
}
```

Toute la difficulté consiste à implémenter la méthode **getManager()**. Nous allons voir comment faire cela.

Commençons par utiliser secrètement notre code généré depuis l'AIDL, dans une classe **CheckLicenceForMyMarketImpl**, surchargeant **CheckLicenceForMyMarket**. Notre implémentation va

initier la communication avec l'implémentation présente dans la place de marché, via un **bindService()**.

```
public class CheckLicenceForMyMarketImpl extends CheckLicenceForMyMarket
{
 private static final String ACTION_LICENCE="org.checklicence.LICENCE";
 private ICheckLicence mRemoteLicence;
 public CheckLicenceForMyMarketImpl(Context context)
 {
  final Intent intent=new Intent(ACTION_LICENCE);
  context.bindService(intent, new ServiceConnection()
  {

   @Override
   public void onServiceDisconnected(ComponentName name)
   {
    mRemoteLicence=null;
   }

   @Override
   public void onServiceConnected(ComponentName name, IBinder binder)
   {
    mRemoteLicence=ICheckLicence.Stub.asInterface(binder);
   }
  }, Context.BIND_AUTO_CREATE);
 }
 @Override
 public boolean checkLicenceForMyMarket()
 {
  try
  {
   return mRemoteLicence.checkLicence();
  }
  catch (RemoteException e)
  {
   throw new Error(e);
  }
 }
}
```

Cette classe est à placer dans la librairie d'implémentation.

Une première implémentation de la méthode **getManager()** peut être celle-ci

```
static CheckLicenceForMyMarket getManager(Context context)
{
 return new CheckLicenceForMyMarketImpl(context);
}
```

Mais dans ce cas, la classe **CheckLicenceForMyMarketImpl** doit être placé dans **CheckLicence-lib**, ceux que nous voulons éviter. Nous souhaitons la placer dans **CheckLicence-sharedlib** car l'implémentation est fortement lié aux services publié par **CheckLicence.apk**.

La prochaine chose à faire, et de modifier le build du projet **CheckLicence-sharedlib** pour générer un Jar signé, aligné et utilisant le format DEX.

Il faut revenir un peu sur la machine virtuelle Dalvik utilisée par Android. Cette machine virtuelle utilise un byte-code différent du byte code standard de Java. La particularité de ce dernier est qu'il prépare la liaison entre toutes les classes de l'archive. Si vous dépliez un APK (qui est un fichier ZIP), vous trouverez un fichier **classes.dex** à la place des nombreux fichiers **.class** d'une archive classique. Ce fichier possède l'union de toutes les classes de l'archive, avec un byte code spécifique. Le format est conçu pour pouvoir être directement mappé en mémoire à l'aide d'un **mmap**. Pour optimiser cela, l'archive doit être aligné sur des frontières de mots assembleur. Cela fait partie du processus de build d'android lors de la génération de la release. Le Jar doit également être signé, car les classes sont vérifiés lors de leurs installations en mémoire.

Notre objectif est de réaliser une archive JAR compatible Dalvik, qui sera publié par **CheckLicence.apk**.

Le mécanisme de build d'eclipse n'est pas suffisant pour cela. Nous devons convertir le projet pour utiliser **ant**, bien connu des développeurs Java.

```
$ cd CheckLicence-sharedlib
```

$ android update project --path . --name CheckLicence-sharedlib

Cela nous génère un fichier **build.xml** que nous allons modifier.

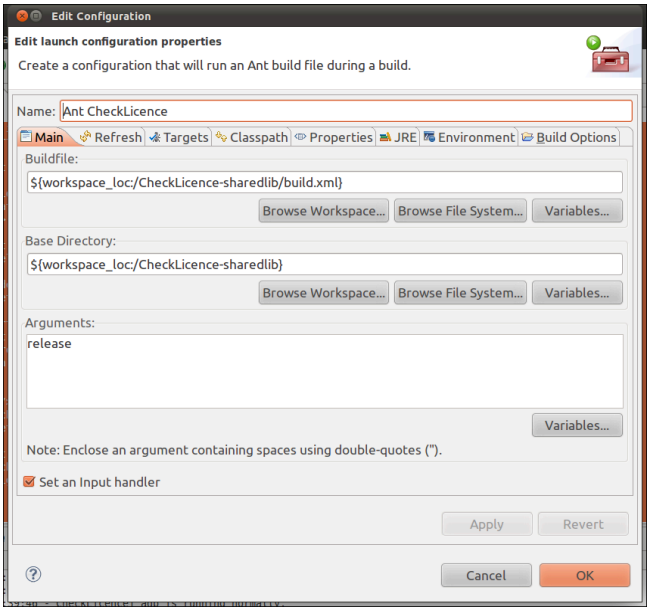
Nous modifions le marqueur **<setup/>** par ceci :

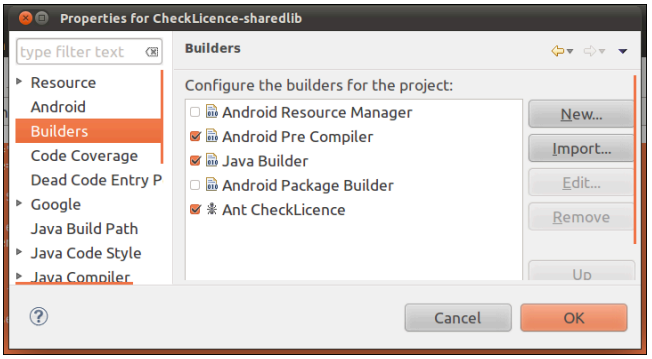
```
  <setup import="false"/>
 <property name="verbose" value="true" />

  <import file="${sdk.dir}/tools/ant/main_rules.xml" />

  <path id="project.libraries.src">
  <pathelement location="${android.library.reference.1}/src"/>
  </path>

 <target name="-post-compile">
  <delete>
     <fileset dir="${out.dir}/classes/org/checklicence/lib" includes="*.class"/>
   </delete>
 </target>

 <property name="out.unsigned.file" value="${out.release.file}" />
 <target name="package" depends="-dex">
  <property name="out.unsigned.file" value="${out.release.file}" />
  <jar destfile="${out.unsigned.file}"
     basedir="${out.dir}/"
     includes="classes.dex"
   />
  <echo>Shared librairy done</echo>
 </target>

 <target name="release" depends="package">
    <echo>Signing final jar...</echo>
    <signjar
         jar="${out.unsigned.file}"
         signedjar="../CheckLicence/assets/sharedlib.jar"
         keystore="${key.store}"
         storepass="${key.store.password}"
         alias="${key.alias}"
         keypass="${key.alias.password}"
         verbose="${verbose}" />
 </target>
```

Cette modification va générer un fichier **sharedlib.jar** dans le répertoire **asset** du projet **CheckLicence**. Ce fichier possède les classes à partager. N'oubliez pas de créer un fichier **local.properties** avec les informations personnel, comme la localisation du SDK, le fichier des clefs et les mots de passes.

Le fichier **classes.dex** possède toutes les classes de l'archives. Lors du chargement par un classe loader, il ne va pas vérifier s'il existe déjà une version de la classe dans un classe loader supérieur. Contrairement au Java classique, nous devons supprimer toutes les classes de **${out.dir}/classes/org/checklicence/lib** pour ne pas créer d'ambiguïté par la suite.

Nous devons ensuite modifier le build d'eclipse pour lui demander d'utiliser le fichier **ant**.





Nous avons créé une librairie Android que nous plaçons dans le répertoire **../CheckLicence/assets/sharedlib.jar**.

Notre application **CheckLicence.apk** peut alors l'exposer à toutes les autres applications. Mais comme les **assets** ne sont pas réellement des fichiers, juste un accès à l'APK de l'application, il faut auparavant en faire une copie accessible par toutes les applications. Nous faisons cela dans la méthode **onCreate()** d'une instance **Application**, bien entendu en tâche de fond, car il est interdit d'avoir des IO dans le main thread.

```
public class MyApplication extends Application
{
  private static final String TAG="app";
  private static final boolean USE_SHAREDLIB=true;
  private static final String SHARED_LIB="sharedlib.jar";

  @Override
  public void onCreate()
  {
    super.onCreate();
    if (USE_SHAREDLIB)
    {
      // Copy a public version of shared library
      new Thread("Copy shared library")
      {
        public void run()
        {
          final SharedPreferences
prefs=getSharedPreferences("sharedlib",Context.MODE_PRIVATE);
          final long lastCopied=prefs.getLong("copy", -1);
          final long packageLastModified=new
File(getApplicationInfo().publicSourceDir).lastModified();
          if (packageLastModified>lastCopied)
          {
            InputStream in=null;
```

```
        OutputStream out=null;
        try
        {
         in=getAssets().open(SHARED_LIB);
         out=openFileOutput(SHARED_LIB, Context.MODE_WORLD_READABLE);
         byte[] buf=new byte[1024*4];
         for (;;)
         {
          int s=in.read(buf);
          if (s<1) break;
          out.write(buf,0,s);
         }
         prefs.edit().putLong("copy",packageLastModified).commit();
        }
        catch (IOException e)
        {
         Log.e(TAG,"Impossible to copy shared library",e);
        }
        finally
        {
         if (in!=null)
         {
          try
          {
           in.close();
          }
          catch (IOException e)
          {
           Log.e(TAG,"Impossible to close input stream",e);
          }
          try
          {
           out.close();
          }
          catch (IOException e)
          {
           Log.e(TAG,"Impossible to close input stream",e);
          }
         }
        }
       }
      }.start();

   }
  }
}
```

Enfin, nous avons un **sharedlib.jar** de type Android, disponible en lecture par toutes les applications. Nous pouvons maintenant utiliser un **ClassLoader** pour charger la librairie et implémenter notre méthode **getManager()**.

```
{
  public static synchronized CheckLicenceForMyMarket getManager(Context context)
{

   if (mSingleton==null)
   {
    try
    {
      ClassLoader classLoader=CheckLicenceForMyMarket.class.getClassLoader();
      if (USE_SHAREDLIB)
      {
      File dir=context.getApplicationContext().getDir("dexopt", Context.MODE_PRIVATE);
      final String packageName="org.checklicence";
      PackageInfo info=context.getPackageManager().getPackageInfo(packageName, 0);
      String jar=info.applicationInfo.dataDir+"/files/"+SHARED_LIB;
      InputStream in=new FileInputStream(jar); in.read(); in.close(); // Check if is readable
      classLoader=
       new DexClassLoader(jar,
```

```
            dir.getAbsoluteFile().getAbsolutePath(),null,
            classLoader
            );
        }
       mSingleton=(CheckLicenceForMyMarket)classLoader.loadClass(BOOTSTRAP_CLASS)
        .getConstructor(Context.class).newInstance(context);
      }
      catch (Exception e)
      {
       throw new Error("Install the Remote Android package",e);
      }
    }
    return mSingleton;
  }
  private static final String BOOTSTRAP_CLASS=
    "org.checklicence.lib.shared.CheckLicenceForMyMarketImpl";
  static final boolean USE_SHAREDLIB=true;
  static final String SHARED_LIB="sharedlib.jar";
}
```

Nous utilisons un **ClassLoader** spécifique pour les fichiers DEX. Ce dernier à besoin d'un nom de fichier source et d'un répertoire valide en écriture, qui va permettre de stocker une version optimisée de notre archive. Comme nous avons bien supprimé les classes partagées entre notre archive et l'archive pré-installé dans chaque application, il n'y a aucun problème de confusion dans les classes. Sinon, une erreur abscon va trahir la présence d'une Dalvik par rapport à une JVM classique.

Nous pouvons enfin proposer un exemple d'application Android qui utilise notre librairie partagée. L'application dois utiliser la librairie légère **CheckLicence-lib**, cela permet d'utiliser en réalité **CheckLicence-sharedlib** dont l'archive viens de l'application **CheckLicence.apk**.

```
public class TestCheckLicenceActivity extends Activity
{
  private CheckLicenceForMyMarket mLicenceManager;

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

       mLicenceManager=CheckLicenceForMyMarket.getManager(this);

       Button button=new Button(this);
       button.setText("Check licence");
       button.setOnClickListener(new OnClickListener()
    {

      @Override
      public void onClick(View v)
      {
       checkLicence();
      }
    });
       setContentView(button);
    }
    private void checkLicence()
    {
     boolean licence=mLicenceManager.checkLicenceForMyMarket();
     new AlertDialog.Builder(this)
        .setMessage("Licence ="+licence)
        .setOnCancelListener(
           new AlertDialog.OnCancelListener()
           {
            public void onCancel(DialogInterface dialog)
            {
             finish();
            }
           })
        .show();
    }
```

```
}
```

Avec les sources, vous trouverez quatre packages :

- **CheckLicence** qui est l'application portant la librairie partagée ainsi que l'implémentation du service de vérification de la licence ;
- **CheckLicence-lib** qui est la libraire la plus légère possible, ne proposant que des interfaces et la méthode **getManager()**.
- **CheckLicence-sharedlib** qui est la librairie partagées par toutes les applications, via le classloader ;
- **CheckLicence-test**, l'application de test de tous cela.

Tous est disponible sur **www.prados.fr**.

Conclusion

Voici un nouveau composant pour notre place de marché. Nous pouvons maintenant proposer des APIs spécifiques pour permettre de vérifier dynamiquement la licence du programme, ou pour permettre l'achat d'extension pendant l'exécution de l'application. Si une vulnérabilité est découverte dans l'API, une simple mise à jour du market suffit. Il n'est pas nécessaire de modifier toutes les applications qui propose un achat dans l'application !

Avec beaucoup d'effort, on arrive à dépasser les limites du framework proposé. Rien n'est simple dans le petit monde d'Android;-)

Philippe PRADOS article@prados.fr
Architecte Senior. AtoS.