Programmation Nomade (Android)

Dominique Archambault
Master *"Technologie et Handicap"*
Université Paris 8

Cours n – Sauvegarder de données – 2012/2013
(Version corrigée le 19 décembre 2012)

1 Introduction

3 situations
- Paires clé/valeur de type simple, dans un fichier de préférence partagées.
- Sauvegarder des fichiers dans le système de fichier
- Utiliser la base SQLite

2 Fichier de préférence partagées

2.1. Récupérer une référence vers un objet `SharedPreferences`

2 méthodes
- **`SharedPreferences getSharedPreferences(String name, int mode)`**
  Si plusieurs fichiers de préférences sont nécessaires.
- **`SharedPreferences getPreferences(int mode)`**
  Si on a besoin d'un seul fichier de préférences.

Dans les 2 cas le **`mode`** permet de contrôler les permissions, le mode principal est **`MODE_PRIVATE`**, 2 autres possibilités : **`MODE_WORLD_READABLE`** et **`MODE_WORLD_WRITEABLE`**.

```
Context context = getActivity();
SharedPreferences sharedPref =
    context.getSharedPreferences(
        getString(R.string.preference_file_key),
        Context.MODE_PRIVATE);
```

ex. de nom de fichier
`com.example.myapp.PREFERENCE_FILE_KEY`

```
SharedPreferences sharedPref =
    getActivity().getPreferences(
        Context.MODE_PRIVATE);
```

2.2. Écrire dans un objet `SharedPreferences`

- Créer un **`Editor`** sur cet objet
  `SharedPreferences.Editor editor = sharedPref.edit();`
- Passer les paires clé/valeur en utilisant des méthodes de type **`putInt(String key, int value)`** ou **`putString(String key, String value)`**.
  ```
  editor.putInt(getString(R.string.saved_high_score),
                newHighScore);
  ```
- Appeler la méthode **`commit()`** pour sauvegarder les modifications.
  `editor.commit();`

Exemple

```
SharedPreferences sharedPref;
sharedPref=getActivity()
            .getPreferences(Context.MODE_PRIVATE);

SharedPreferences.Editor editor=sharedPref.edit();

editor.putInt(getString(R.string.highscore_val),
              newHighScore);
editor.putString(getString(R.string.highscore_name),
              newHighScoreName);

editor.commit();
```

2.3. Lire dans un objet `SharedPreferences`

- On utilise les méthodes de type :
  `int getInt (String key, int defValue)`
  ou **`String getString(...)`**.
  Attention une valeur par défaut doit être donnée.
  ```
  long highscore = sharedPref.getInt(
                getString(R.string.saved_high_score),
                default_highscore);
  ```

Exemple

```
SharedPreferences sharedPref=getActivity()
            .getPreferences(Context.MODE_PRIVATE);

def_highscore=getResources()
        .getInteger(R.string.high_score_default));
def_HS_name=getResources()
        .getString(R.string.HS_name_default));

highscore=sharedPref.getInt(
              getString(R.string.highscore_val),
              def_highscore);
HS_name=sharedPref.getInt(
              getString(R.string.highscore_name),
              def_HS_name);
```

3 Sauvegarder dans le système de fichiers

1. Choisir entre stockage "Interne" ou "Externe"
2. Obtenir les permissions pour le stockage "Externe"
3. Enregistrer un fichier sur le stockage "Interne" ou "Externe"
4. Connaître l'espace disponible
5. Effacer un fichier

3.1. Stockage "Interne" ou "Externe"

Stockage Interne
- Toujours présente
- Fichiers réservés à notre l'application (par défaut)
- Fichiers supprimés quand un désinstalle l'application

Stockage Externe
- Pas forcément présente, peut être démontée (et physiquement retirée), peut aussi avoir été changée.
- Pas de contrôle de droits de lecture : toutes les applications y ont accès.
- en cas de désinstallation, ne sont supprimées que si elles sont dans le répertoire prévu (**`getExternalFilesDir()`**).

3.2. Obtenir des permission

Stockage Externe
Pour écrire sur le stockage externe il est nécessaire d'en obtenir la permission, via le manifest.

```
<manifest ...>
  <uses-permission android:name=
     "android.permission.WRITE_EXTERNAL_STORAGE" />
  ...
</manifest>
```

Stockage Interne
Aucune permission n'est nécessaire.

3.3a. Enregistrer sur le stockage Interne

Récupérer le répertoire de l'application

- **`File getFilesDir()`**
  Renvoie un objet **`File`** représentant un répertoire interne.
- **`File getCacheDir()`**
  Renvoie un objet **`File`** représentant un répertoire interne pour des fichiers temporaires.

Créer un fichier (1)

Utiliser le constructeur **`File(File dir, String name)`** pour créer un objet de type fichier.
par exemple :

```
File f = new File(context.getFilesDir(), nomfich);
```

Créer un fichier (2)

On peut aussi utiliser
`FileOutputStream openFileOutput(String name, int mode)`
pour récupérer un flux d'écriture sur un fichier.
Le système décide de l'emplacement du fichier (en fonction du mode).

Modes :

- **`MODE_APPEND`**
- **`MODE_PRIVATE`**
- **`MODE_WORLD_READABLE`**
- **`MODE_WORLD_WRITEABLE`**

Exemple

```
String nomfich = "monfich.txt";
String machaine = "Hello␣world!";
FileOutputStream outstream;

try {
  outstream=openFileOutput(nomfich,
                           Context.MODE_PRIVATE);
  outstream.write(machaine.getBytes());
  outstream.close();
}
catch (Exception e) {
  ...
}
```

Exemple

```
public File getTempFile(Context cx, String url) {
  File f;
  try {
      String fnme = Uri.parse(url)
                       .getLastPathSegment();
      f = File.createTempFile(fname, null,
                              cx.getCacheDir());
  }
  catch (IOException e) {
    // Erreur lors de la création du fichier
  }
  return f;
}
```

3.3b. Enregistrer sur le stockage Externe
Toujours vérifier l'état du stockage

```
public boolean isExternalStorageWritable() {
  String state=Environment.getExternalStorageState();
  if (state.equals(Environment.MEDIA_MOUNTED)) {
    return true;
  }
  return false;
}

public boolean isExternalStorageReadable() {
  String state=Environment.getExternalStorageState();
  if (state.equals(Environment.MEDIA_MOUNTED) ||
    state.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
    return true;
  }
  return false;
}
```

2 types de fichiers

- Fichiers publics
- Fichiers privés

`File getExternalStoragePublicDirectory (String type)` renvoie un répertoire pour les fichiers publics. Où **`type`** correspond au type de fichier qu'on veut stocker :

- **`Environment.DIRECTORY_MUSIC`**
- **`Environment.DIRECTORY_PODCASTS`**
- **`Environment.DIRECTORY_RINGTONES`**
- **`Environment.DIRECTORY_ALARMS`**
- **`Environment.DIRECTORY_NOTIFICATIONS`**
- **`Environment.DIRECTORY_PICTURES`**
- **`Environment.DIRECTORY_MOVIES`**
- **`Environment.DIRECTORY_DOWNLOADS`**
- **`Environment.DIRECTORY_DCIM`**

Exemple

```
public File getAlbumStorageDir(String albumName) {
  File f=new File(Environment.
              getExternalStoragePublicDirectory(
                  Environment.DIRECTORY_PICTURES),
              albumName);
  if (!f.mkdirs()) {
    Log.e(LOG_TAG, "Le␣dossier␣n'a␣pas␣été créé");
  }
  return f;
}
```

`File getExternalFilesDir (String type)` renvoie un répertoire pour les fichiers privés. Où **`type`** correspond au type de fichier qu'on veut stocker, ou bien **`null`**.

3.4. Connaître l'espace disponible

Pour éviter de causer une **`IOException`**, on peut utiliser les méthodes suivantes de la classe **`File`** :

- **`long getFreeSpace()`**
  renvoie l'espace disponible sur le volume correspondant au dossier courant.
- **`long getTotalSpace()`**
  renvoie l'espace total sur ce volume.

3.5. Effacer un fichier

- Appeler la méthode **`delete()`** sur l'objet à détruire :
  `myFile.delete();`
- si le fichier est sur le stockage interne : **`deleteFile(String name)`**
  `myContext.deleteFile(fileName);`

Lors de la désinstallation de l'application, le système efface :

- Les fichiers installés sur le stockage interne
- Les fichiers installés sur le stockage externe créés avec **`getExternalFilesDir()`**

4 Utiliser la base SQLite

1. Définir un schéma et un contrat
2. Créer la base à l'aide d'un "SQL Helper"
3. Enregistrer des données dans la base
4. Lire des données de la base
5. Effacer des données de la base

4.1. Définir un schéma et un contrat

Schéma

L'organisation de la base de données.

- Les tables
- Les champs
- (Éventuellement) les relations

Contrat
Le schéma est constitué d'un ensemble de clauses SQL, que l'on va regrouper dans une classe **Contrat**, qui contiendra des constantes correspondant aux noms des tables et des champs. Si plusieurs tables sont nécessaires, on pourra utiliser des sous classes pour leurs champs.

```
public static class EntryContract
                    implements BaseColumns {
  public static final String TABLE_NAME = "entry";
  public static final String FIELD_ENTRY_ID =
                               "entryid";
  public static final String FIELD_TITLE =  "title";
  public static final String FIELD_SUBTITLE =
                               "subtitle";
  ...
```

```
  // Pour eviter de l'instancier par erreur
  private EntryContract() {}
  ...
```

4.2. Créer la base : un SQL Helper

SQL helper
Il s'agit d'une classe dans laquelle on va encapsuler les appels à la base de données.
- Des chaînes constantes pour les requêtes SQL
- Des méthodes pour créer/retrouver la base

On peut la faire dériver de **`SQLiteOpenHelper`** qui fournit des méthodes pour ouvrir la base :
- **`SQLiteDatabase getWritableDatabase()`**
- **`SQLiteDatabase getReadableDatabase()`**.

Utilisation de `SQLiteOpenHelper`
Il faut implémenter les méthodes :
- **`void onCreate(SQLiteDatabase db)`**
- **`void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`**
- **`void onOpen (SQLiteDatabase db)`**

et *(facultatif)*
- **`void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)`**

Exemple

```
public class MyAppDbHelper extends SQLiteOpenHelper {
  // A chaque fois que le schema change,
  // incrementer le numero de version.
  public static final int DATABASE_VERSION = 1;
  public static final String DATABASE_NAME = "MyApp.db";

  private static final String TEXT_TYPE = "␣TEXT";
  private static final String COMMA_SEP = ",";
```

```
  private static final String SQL_CREATE_ENTRIES =
    "CREATE␣TABLE␣" +EntryContract.TABLE_NAME +"␣("
    +EntryContract._ID +"␣INTEGER␣PRIMARY␣KEY,"
    +EntryContract.FIELD_ENTRY_ID +TEXT_TYPE +COMMA_SEP
    +EntryContract.FIELD_TITLE +TEXT_TYPE +COMMA_SEP
    +... "␣)";
```

```
  private static final String SQL_DELETE_ENTRIES =
    "DROP␣TABLE␣IF␣EXISTS␣" + EntryContract.TABLE_NAME;
```

```
  public MyAppDbHelper(Context cx) { // Constructeur
    super(cx, DATABASE_NAME, null, DATABASE_VERSION);
  }
  public void onCreate(SQLiteDatabase db) {
    db.execSQL(SQL_CREATE_ENTRIES);
  }
  public void onOpen(SQLiteDatabase db) {
    // skip
  }
```

```
  public void onUpgrade(SQLiteDatabase db,
                        int oldVersion, int newVersion) {
    // Ici on efface tout et on recommence (pas top!)
    db.execSQL(SQL_DELETE_ENTRIES);
    onCreate(db);
  }
  public void onDowngrade(SQLiteDatabase db,
                        int oldVersion, int newVersion) {
    onUpgrade(db, oldVersion, newVersion);
  }
```

Accéder à la base
Il suffit maintenant de créer une instace de notre SQLiteOpenHelper.

```
MyAppDbHelper mDbHelper=new MyAppDbHelper(getContext());
```

Utiliser la base
Selon qu'on veut lire ou écire on utilisera l'une des méthodes **`getReadableDatabase()`** ou **`getWritableDatabase()`** :

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();
```

ou

```
SQLiteDatabase db = mDbHelper.getWritableDatabase();
```

4.3. Enregistrer des données dans la base

Enregistrer
- Créer un objet **`ContentValues`**
- L'insérer dans la base en le passant à la méthode **`insert(String table, String nullColumnHack, ContentValues values)`**

```
// Creer l'objet ContentValues
ContentValues values = new ContentValues();
values.put(EntryContract.FIELD_ENTRY_ID, id);
values.put(EntryContract.FIELD_TITLE, title);
values.put(EntryContract.FIELD_CONTENT, content);
```

```
// Inserer la nouvelle entree, en renvoyant la cle
// primaire
long newRowId;
newRowId = db.insert(
         EntryContract.TABLE_NAME,
         EntryContract.FIELD_NULLABLE,
         values);
```

4.4. Lire des données de la base

Lire
On va utiliser la méthode **`query(...)`**, qui renvoie un objet **`Cursor`**. On va filtrer la requête, et ne récupérer que les champs qui nous intéressent.

```
public Cursor query(
  String table,
  String[] columns, Liste des champs à retourner
  String selection, filtre (clause SQL WHERE)
  String[] selectionArgs,
  String groupBy,
  String having,
  String orderBy)
```

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Les champs dont on a besoin
String[] projection = {
    EntryContract._ID,
    EntryContract.FIELD_TITLE,
    EntryContract.FIELD_UPDATED,
    ...
```

```
    };

// Ordre de tri
String sortOrder =
    EntryContract.FIELD_UPDATED + "␣DESC";
```

```
Cursor c = db.query(
    EntryContract.TABLE_NAME,
    projection,
    selection, // a definir
    selectionArgs, // idem
    null,  // pas de groupe de lignes
    null,  // pas de filtre sur les groupes de ligne
    sortOrder
    );
```

4.5. Effacer des données de la base

Pour effacer des entrées, il faut donner un critère de sélection des entrées à effacer.

```
String selection = EntryContract.FIELD_ENTRY_ID
                              +"␣LIKE␣?";

String[] selelectionArgs = { String.valueOf(rowId) };

db.delete(table_name, mySelection, selectionArgs);
```