

Introduction

En programmation, la gestion de la persistance réfère au mécanisme responsable de la sauvegarde et de la restauration des données. Ce mécanisme fait en sorte qu'un programme puisse se terminer sans que ses données ne soient perdues. La persistance des données peut être locale ou distante.

La première consiste à stocker les données dans la mémoire interne du terminal mobile à travers l'utilisation : des préférences partagées, des fichiers ou encore du système de gestion des bases de données relationnelles SQLite.

La seconde permet de stocker les données dans une base de données externe. Cependant, vu qu'Android ne peut pas communiquer directement avec la base de données distante, cela est rendu possible grâce à l'utilisation d'une application web utilisée comme intermédiaire entre Android et la base.

Le choix de la solution à utiliser dépend de la nature des données à stocker, de leur quantité et du nombre d'utilisateurs :

- Si les données ne sont pas structurées alors on utilise les préférences partagées ou les fichiers.
- Si les données sont structurées alors on utilise une base de données :
 - Si leur quantité n'est pas importante et uniquement le propriétaire du terminal mobile utilise les données alors la persistance locale est la plus adéquate
 - Sinon on fait recours à une persistance distante.

I. Les préférences partagées

Afin d'utiliser les préférences partagées dans une application Android, on est amené à utiliser la classe `SharedPreferences` du package `android.content` permettant de gérer les préférences partagées sous forme d'un fichier xml qui contient un ensemble de paires clé/valeur.

Pour initialiser un `SharedPreferences` on peut utiliser l'une de ces deux solutions :

- `SharedPreferences p=PreferenceManager.getDefaultSharedPreferences(getApplicationContext());`
Le fichier crée aura un nom qui dépend du package de l'application et son chemin complet est : `"/data/data/[application package name]/shared_prefs/[application package name]_preferences.xml"`
- `SharedPreferences p=getSharedPreferences("conn", Context.MODE_PRIVATE);`
Le nom du fichier sera `conn.xml` et son chemin complet est : `"/data/data/[application package name]/shared_prefs/conn.xml"`.
Le paramètre `MODE` peut être :
 - `MODE_PRIVATE` : le mode par défaut, seul l'application qui a créé les préférences peut y accéder.
 - `MODE_APPEND` : permet d'ouvrir les préférences partagées en mode ajout.

Ecriture

Afin de créer une préférence partagée, il faut :

1. Créer une instance de la classe `Editor` du package `android.content.SharedPreferences` : `Editor ed= p.edit();`
2. Ajouter des paires clé/valeur aux préférences partagées à travers l'une de ces méthodes :
`Editor putBoolean(String key, boolean value)`
`Editor putFloat(String key, float value)`
`Editor.putInt(String key, int value)`
`Editor.putLong(String key, long value)`
`Editor.putString(String key, String value)`
3. Valider le changement des valeurs utiliser :
`boolean commit()`

Cours : Développement Mobile

Remarque :

Pour effacer tous les paires utiliser : Editor clear()

Exemple :

```
//SharedPreferences p=getSharedPreferences("conn", Context.MODE_PRIVATE);  
//Editor ed= p.edit();  
//ed.putString("utilisateur", "Mohamed");  
//ed.commit();
```

Lecture

Pour lire une valeur à partir des préférences partagées, utiliser une de ces méthodes suivant le type de la valeur :

- boolean getBoolean(String key, boolean defValue)
- float getFloat(String key, float defValue)
- int getInt(String key, int defValue)
- long getLong(String key, long defValue)
- String getString(String key, String defValue)

Exemple :

```
//SharedPreferences p=getSharedPreferences("conn", Context.MODE_PRIVATE);  
//String utilisateur = p.getString("utilisateur", "");
```

II. Les fichiers

La lecture et l'écriture des fichiers sous Android est identique à ce que l'on pourrait faire en Java. Plusieurs solutions existent, parmi lesquelles nous allons exposer la solution la plus basique qui consiste à :

- Déclarer des objets de types FileInputStream ou FileOutputStream
- Lire ou écrire les données avec les méthodes read ou write
- Libérer le flux avec close() une fois terminé

II.1. Lecture

Voici un exemple de code permettant de lire le contenu d'un fichier " file.txt" :

```
String file = "file.txt";  
FileInputStream fin;  
try {  
    fin = openFileInput(file);  
    int c;  
    String data = "";  
    while ((c = fin.read()) != -1) {  
        data = data + Character.toString((char) c);  
    }  
    fin.close();  
    Toast.makeText(getApplicationContext(), "Données récupérées",  
    Toast.LENGTH_SHORT)  
        .show();  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (IOException e) { e.printStackTrace(); }
```

L'exemple suivant permet d'écrire dans le fichier " file.txt" :

```
String data;
String file = "file.txt";
try {
    FileOutputStream fOut = openFileOutput(file,
    MODE_WORLD_READABLE);
    data = "Mohamed";
    fOut.write(data.getBytes());
    fOut.close();
    Toast.makeText(getApplicationContext(), "Données enregistrées",
    Toast.LENGTH_SHORT)
        .show();
}
catch (Exception e) { e.printStackTrace(); }
```

III. Base de données locale : SQLite

SQLite est une bibliothèque écrite en C qui propose un moteur de base de données relationnelle accessible par le langage SQL.

Chaque application Android peut créer une base de données SQLite : C'est un fichier .db placé dans le dossier /data/data/PAQUETAGE/databases/NOM_BDD. Les étapes à suivre afin d'utiliser une base de données sous Android sont :

1. Déclarer une classe équivalente à chaque table de la base SQLite. Le nombre d'attributs de chaque classe est égal à celui des champs de la table.
2. Déclarer une sous classe de la classe SQLiteOpenHelper et redéfinir le constructeur, la méthode onCreate et la méthode onUpgrade.
3. Utiliser les méthodes de la classe SQLiteDatabase pour exécuter les requêtes.
4. Interroger la base :
 - o Utiliser la classe ContentValues pour remplir les paires clé/valeur des requêtes INSERT et UPDATE.
 - o Utiliser la classe Cursor pour parcourir les résultats des requêtes de type SELECT.

La suite du cours utilise une table **Livre** appartenant à la base de données **bibliotheque** et possédant le schéma suivant :

Livre (**id**, titre, nbpage)

III.1. Création des classes équivalentes aux tables de la base

Dans cette étape, nous allons créer une classe Livre

```
public class Livre {
    private int id;
    private String titre;
    private int nbpage;
    public Livre(int id, String titre, int nbpage) {
        this.id = id;
        this.nbpage = nbpage;
        this.titre = titre;
    }
    public int getId() {
```

```
        return id;
    }
    public String getTitre() {
        return titre;
    }
    public int getNbpage() {
        return nbpage;
    }
    public String toString() {
        return id + "-" + titre + "-" + nbpage;
    }
}
```

III.2. Création d'une sous-classe de SQLiteOpenHelper

SQLiteOpenHelper c'est une classe abstraite du package android.database.sqlite qui permet de gérer la création de la base et ses versions, généralement on utilise une sous classes qui implémente les deux méthodes abstraites onCreate et onUpgrade.

Pour notre exemple, nous allons créer une classe intitulée SQLiteBib qui hérite de la classe SQLiteOpenHelper

Exemple :

```
package com.bib;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;
public class SQLiteBib extends SQLiteOpenHelper{
    public SQLiteBib(Context context, String name, CursorFactory
factory, int version) { super(context, name, factory, version); }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String Sql="create table livre (id INTEGER PRIMARY KEY
AUTOINCREMENT,titre text NOT NULL,nbpage INTEGER );";
        db.execSQL(Sql);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    }
}
```

III.3. SQLiteDatabase

SQLiteDatabase c'est une classe du package android.database.sqlite qui permet de gérer une base de données SQLite.

- SQLiteDatabase getWritableDatabase () : permet d'ouvrir la base de données en mode lecture/écriture
- SQLiteDatabase getReadableDatabase () : permet d'ouvrir la base de données en mode lecture

Ensuite, nous allons utiliser les opérations sur la BD à travers les méthodes suivantes :

- long insert (String table, String nullColumnHack, ContentValues values)
- int update (String table, ContentValues values, String whereClause, String[] whereArgs)
- int delete (String table, String whereClause, String[] whereArgs)
- Cursor query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

Après avoir effectué les méthodes CRUD¹, il faudrait fermer la base de donnée avec la méthode `close()`.

```
//SQLiteBib b= new SQLiteBib(getApplicationContext(), "bib.db", null, 1);
//SQLiteDatabase db = b.getWritableDatabase();
// Méthodes CRUD
db.close();
```

III.4. Méthodes CRUD

a. Ajout

Avant d'insérer un enregistrement dans la table, nous devons mettre les données dans un objet de type `ContentValues`. `ContentValues` est une classe du package `android.content` qui permet de stocker un ensemble de paires (clé, valeur). Parmi ses méthodes :

- `void put(String key, Integer value)`
- `void put(String key, Float value)`
- `void put(String key, String value)`

Ensuite, nous utilisons la méthode `insert()` de la classe `SQLiteOpenHelper`:

- `long insert (String table, String nullColumnHack, ContentValues values)`

Exemple :

Pour insérer un livre intitulé Android de 300 pages, nous utilisons le code suivant :

```
//ContentValues cv = new ContentValues();
//cv.put("titre", "Android Nougat");
//cv.put("nbpage", 300);
//db.insert("livre", null, cv);
```

Remarque :

Les clés utilisées avec la méthode `put()` doivent avoir les mêmes noms des champs de la table

b. Consultation

Afin de consulter des enregistrements de la table, il est impératif de créer un objet de type `Cursor` qui contient le résultat de la requête obtenu avec la méthode `query` appliquée à l'objet de type `SQLiteOpenHelper` utilisé précédemment:

```
Cursor query (String table, String[] columns, String selection, String[]
String groupBy, String having, String orderBy, String limit)           selectionArgs,
```

`Cursor` est une interface du package `android.database.sqlite` qui permet l'accès à un résultat d'une requête sur une base SQLite. Parmi ses méthodes :

- `abstract void close()`
- `abstract int getCount()`
- `abstract float getFloat (int columnIndex)`
- `abstract int getInt (int columnIndex)`
- `abstract String getString (int columnIndex)`
- `abstract boolean moveToFirst()`
- `abstract boolean moveToLast ()`
- `abstract boolean moveToNext ()`
- `abstract boolean moveToPosition (int position)`
- `abstract boolean moveToPrevious ()`

¹ CRUD : Create Read Update Delete

Exemple:

```
SQLiteBib b = new SQLiteBib(this, "bib.db", null, 1);
SQLiteDatabase db = b.getWritableDatabase();
Cursor c = db.query("livre", new String[] { "id", "titre",
"nbpage" }, "", null, null, null);
while (c.moveToFirst()) {
    int id = c.getInt(0); //c.getInt("id");
    String titre = c.getString(1);
    int nbpage = c.getInt(2);
    Livre l = new Livre(id, titre, nbpage);

    Toast.makeText(getApplicationContext(), l.toString(), Toast.LENGTH_LONG)
.show();
}
c.close();
```

c. Modification

Avant de modifier un enregistrement donné de la table, nous devons utiliser, comme pour l'ajout, un objet de type ContentValues. Ensuite, nous utilisons la méthode update() de la classe SQLiteOpenHelper:

- *int update (String table, ContentValues values, String whereClause, String[] whereArgs)*

Exemple

Pour modifier le titre et le nombre de page du livre d'identifiant égale à 1 :

```
ContentValues v = new ContentValues();
v.put("titre", "Android Oreo");
v.put("nbpage", 400);
SQLiteBib b = new SQLiteBib(this, "bib.db", null, 1);
SQLiteDatabase db = b.getWritableDatabase();
db.update("livre", v, "id=" + 1, null);
```

d. Suppression

Afin de supprimer des enregistrements de la table, nous utilisons la méthode delete() de la classe SQLiteOpenHelper:

- *int delete (String table, String whereClause, String[] whereArgs)*

Exemple

Pour supprimer le livre d'identifiant égale à 1 :

```
SQLiteBib b = new SQLiteBib(this, "bib.db", null, 1);
SQLiteDatabase db = b.getWritableDatabase();
db.delete("livre", "id=" + 1, null);
```

IV. Base de données distante

Contrairement à une base de données locale SQLite, il existe plusieurs méthodes pour gérer une base de données distante. Parmi lesquels, nous pouvons citer :

- L'utilisation d'une application purement web,
- L'utilisation d'une application hybride,
- L'utilisation d'une application native,

Cours : Développement Mobile

IV.1. Application Web

Dans cette solution aucune ligne de code n'est écrite. L'utilisateur lance le navigateur du Smartphone puis tape l'adresse de l'application Web. Cette solution nécessite une connexion internet et elle fonctionne sur tous les terminaux mobiles.

IV.2. Application hybride

Cette solution consiste à ajouter un `WebView` à l'activité et appeler l'adresse de l'application Web.

Exemple :

```
//import ...
public class MainActivity extends Activity {
    private WebView wv1;
    String url="http://192.100.30.12:80/bib/index.html";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        init();
    }
    private void init() {
        wv1=(WebView)findViewById(R.id.webView1);
        wv1.getSettings().setLoadsImagesAutomatically(true);
        wv1.getSettings().setJavaScriptEnabled(true);
        wv1.loadUrl(url);
    }
}
```

IV.3. Application native

Cette approche consiste à travers le développement d'une application native qui envoie des requêtes HTTP et la réception des réponses correspondantes. La base de données distante peut être configurée de 2 manières distinctes :

1. **Hébergée sur un serveur SGBD dédié** : Hébergée sur un serveur SGBD dédié : Pour mettre en œuvre cette option, il est impératif de concevoir un webservice, généralement sous la forme d'une API REST. L'utilisation de bibliothèques HTTP telles que Volley ou Retrofit s'avère incontournable pour effectuer des appels réseau efficaces. Ce processus permet une communication transparente entre l'application et la base de données, en garantissant une gestion optimale des requêtes et des réponses.

2. **Utilisation d'un service cloud** : Utilisation d'un service cloud : Les bases de données distantes peuvent également être externalisées vers des services cloud tels que Firebase, AWS, Azure ou MongoDB. Ces plates-formes fournissent des SDK adaptés à différentes plateformes, simplifiant ainsi l'interaction entre l'application et la base de données distante. Cette approche offre une flexibilité accrue, avec la possibilité de tirer parti des fonctionnalités avancées offertes par ces services cloud, tout en réduisant la complexité du développement grâce à des interfaces simplifiées et des outils adaptés.

Conclusion

Dans ce chapitre, nous avons présenté les différentes solutions de persistance de données commençant par les préférences partagées passant par les fichiers jusqu'à arriver aux bases de données locales SQLite et distantes avec JSON et la bibliothèque volley.

Dans le chapitre suivant, nous allons voir la géolocalisation et les cartes sous android avec l'API google maps.