HelHa
Tournai
Bachelor en Informatique

2019 - 2020
Android

Développement Android

Persistance de données

Version du document : 7 août 2019

1 Conserver ses données en local : SQLite

Dans les applications mobiles, il est souvent nécessaire de stocker des données localement afin de pouvoir les récupérer, consulter et mettre à jour même après le redémarrage de l'appareil. Par exemple, on pourrait vouloir récupérer des préférences de l'utilisateur, des historiques, des données fréquemment consultées, etc.

Pour cela, on pourrait simplement utiliser des fichiers. Mais ce n'est souvent pas pratique ni efficace car mettre à jour des données dans un fichier demande en général de mettre en mémoire l'entièreté du fichier, le mettre à jour et enregistrer le tout à nouveau dans le fichier.

La librairie standard d'Android contient le Framework *SQLite*, un système simple et léger de base de données qui offre un service performant et souple permettant de gérer la persistance des données localement.

Pour des gestions de données plus complexes, on se tournera vers d'autres Framework, plus complets, permettant de définir les types des données, définir des contraintes sur les données (Foreign Keys), ajouter des indexes, ou encore offrant un ORM, Object Relational Mapping, par exemples ORMLite ou GreenDAO.

Dans ce TD vous allez utiliser *SQLite* pour conserver les *Crimes* de votre application *CriminalIntent*.

2 Schema DB

Avant toute chose, modifions légèrement les constructeurs de la classe `Crime` afin de pouvoir construire un `Crime` ayant déjà un identifiant.

```
public Crime() {
    this(UUID.randomUUID());
}

public Crime(UUID id) {
    mId = id;
    mDate = new Date();
}
```

Définissons ensuite les éléments de notre schéma de base de données. Une manière intéressante d'implémenter cela est de définir les tables et colonnes en tant que constantes dans une imbrication de classes.

```
1 public class CrimeDbSchema {
2     public static final class CrimeTable {
3         public static final String NAME = "crimes";
4         public static final class cols {
5             public static final String UUID = "uuid";
6             public static final String TITLE = "title";
7             public static final String DATE = "date";
8             public static final String SOLVED = "solved";
9         }
```

```
    }
}
```

3 SQLiteOpenHelper

La classe `SQLiteOpenHelper` permet de simplifier la mise en place de la base de données dans une application Android.

```
public class CrimeBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "crimeBase.db";
    public CrimeBaseHelper(Context context) {
        super(context, DATABASE_NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("create table "+ CrimeTable.NAME + "("
        + "_id integer primary key autoincrement, "
        + CrimeTable.cols.UUID + ", " + CrimeTable.cols.TITLE + ", "
        + CrimeTable.cols.DATE + ", " + CrimeTable.cols.SOLVED + ")"
        );
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
}
```

Question 1

D'après la javadoc de la classe `SQLiteOpenHelper`, à quel moment sont appelées les méthodes `onCreate` et `onUpgrade` ?
A quoi sert le numéro de version ?

4 Lecture des données : les curseurs

La librairie utilise des curseurs pour parcourir les résultats de requêtes, ce sont des itérateurs sur les lignes des tables.

La classe `CrimeCursorWrapper` permet de faire le lien entre curseur et la classe `Crime` de notre modèle.

```
public class CrimeCursorWrapper extends CursorWrapper {

    public CrimeCursorWrapper(Cursor cursor)
```

```
4       {
5           super(cursor);
6       }
7
8       public Crime getCrime()
9       {
10          String uuidString = getString(getColumnIndex(CrimeDbSchema.CrimeTable.cols.UUID));
11          String  title  = getString(getColumnIndex(CrimeDbSchema.CrimeTable.cols.TITLE));
12          long date = getLong(getColumnIndex(CrimeDbSchema.CrimeTable.cols.DATE));
13          int isSolved  = getInt(getColumnIndex(CrimeDbSchema.CrimeTable.cols.SOLVED));
14
15          Crime crime = new Crime(UUID.fromString(uuidString));
16          crime.setTitle( title );
17          crime.setDate(new Date(date));
18          crime.setSolved(isSolved!=0);
19          return crime;
20      }
21  }
```

Question 2

Quelles sont les méthodes de `CursorWrapper` utilisées dans la classe `CrimeCursorWrapper` ?
Quel est le package de `CursorWrapper` ?

5 Lier `CrimeLab` et la base de données

Il vous reste à lier `CrimeLab`, la liste des crimes, à la base de données.

```
1   public class CrimeLab {
2       private static CrimeLab sCrimeLab;
3
4       private Context mContext;
5       private SQLiteDatabase mDatabase;
6
7       public static CrimeLab get(Context context) {
8           if(sCrimeLab == null) {
9               sCrimeLab = new CrimeLab(context);
10          }
11          return sCrimeLab;
12      }
13      private CrimeLab(Context context) {
14          mContext = context.getApplicationContext();
15          mDatabase = new CrimeBaseHelper(mContext).getWritableDatabase();
16      }
17
18      public void addCrime(Crime crime) {
19          mDatabase.insert(CrimeDbSchema.CrimeTable.NAME, null, getContentValues(crime));
20      }
21
22      public void updateCrime(Crime crime) {
23          String uuidString = crime.getId().toString();
```

```
24          ContentValues values = getContentValues(crime);
25
26          mDatabase.update(CrimeDbSchema.CrimeTable.NAME,
27                           values,
28                           CrimeDbSchema.CrimeTable.cols.UUID + " = ?",
29                           new String[] {uuidString});
30      }
31      public Crime getCrime(UUID id) {
32
33          CrimeCursorWrapper cursor =
34                  queryCrimes(CrimeDbSchema.CrimeTable.cols.UUID+ " = ? ",
35                              new String[] {id.toString()}
36                  );
37          try {
38              if(cursor.getCount() == 0)
39                  return null;
40
41              cursor.moveToFirst();
42              return cursor.getCrime();
43          } finally {
44              cursor.close();
45          }
46      }
47
48      public List<Crime> getCrimes() {
49          ArrayList<Crime> crimes = new ArrayList<>();
50
51          CrimeCursorWrapper cursor = queryCrimes(null, null);
52          try {
53              cursor.moveToFirst();
54              while(!cursor.isAfterLast()) {
55                  crimes.add(cursor.getCrime());
56                  cursor.moveToNext();
57              }
58          } finally {
59              cursor.close();
60          }
61          return crimes;
62      }
63
64      private ContentValues getContentValues(Crime crime) {
65          ContentValues values = new ContentValues();
66          values.put(CrimeDbSchema.CrimeTable.cols.UUID, crime.getId().toString());
67          values.put(CrimeDbSchema.CrimeTable.cols.TITLE, crime.getTitle());
68          values.put(CrimeDbSchema.CrimeTable.cols.DATE, crime.getDate().getTime());
69          values.put(CrimeDbSchema.CrimeTable.cols.SOLVED, crime.isSolved()? 1: 0);
70          return values;
71      }
72
73      private CrimeCursorWrapper queryCrimes(String whereClause, String[] whereArgs) {
74          return new CrimeCursorWrapper(mDatabase.query(
75                  CrimeDbSchema.CrimeTable.NAME,
76                  null, //all columns
77                  whereClause,
78                  whereArgs,
79                  null,null,null));
80      }
81  }
```

Les méthodes publiques `getCrime`, `getCrimes`, `addCrime` et `updateCrime` interagissent avec la base de données.

Les méthodes `getCrime` et `getCrimes` permettent de récupérer un crime et tous les crimes, respectivement. Elles utilisent la méthode privée `queryCrimes` qui retourne un curseur permettant d'accéder aux résultats.

Les méthodes de mise à jour, `addCrime` et `updateCrime`, permettent d'ajouter ou de mettre à jour un crime dans la base de données. Les valeurs, nouvelles ou à mettre à jour, sont contenues dans un objet de type `ContentValues`, c'est un ensemble de pairs clef/valeur, la clef étant le nom de colonne.

Vous avez maintenant tout ce qu'il vous faut pour mettre à jour votre application CriminalIntent pour qu'elle enregistre les crimes dans une base de données.

6 Exercices

Mettez à jour votre application afin que l'on puisse ajouter un nouveau crime, les mettre à jour et les visualiser, les crimes étant enregistrés dans une base de données *SQLite*.

Enregistrez la gravité de chaque crime (Mild (léger), Moderate (moyen), Severe (sérieux)).

Ajoutez également la date de résolution du crime.