Android
Gestion des données

Rémi Forax

Types de stockage

Android fourni plusieurs types de stockage

- Données actives d'une activité (Bundle)
- Fichier ressources read-only (répertoire res)
- Préférence partageable (SharedPreference)
- Le presse papier (ClipboardManager)
- Système de fichiers
  - Donnée interne de l'application (mémoire flash)
    ou données en cache (résultat calcul intermédiaire)
  - Donnée externe partagée (carte SD)
  - En ligne sur le cloud
- Base de donnée (SQLite3)

Préférences Partageables

Lire des couples clé (String)/valeur

Récupère les préférences de l'application

- SharedPreference context.getPreferences(mode)

Récupère les préférences partagée

- SharedPreference context.getSharedPreferences(name,mode)

mode: MODE_[PRIVATE|WORD_READABLE| WORD_WRITABLE]

Récupère les valeurs

- get[Boolean|Float|Int|Long|String(String key, X defaultValue)

Préférences Partageables

Ecrire des couples clé (String)/valeur

Modification transactionnelle des données

- On récupère un editeur avec SharedPrefrences.Editor edit()
- On fait des editor.putX(key, value)
- On valide atomiquement les changement editor.commit()

  ATTENTION: commit peut renvoyer false !

Listener sur les modifications (même celles faite par une autre application

- registerOnSharedPreferenceChangeListener()

Exemple avec des préférences

```
public class MainActivity3 extends Activity {
  private transient Data data;
  static class Data {
    private String emailAddress;
    private String message;
    ...
  }
```

```
@Override protected void onCreate(Bundle savedInstanceState) {
  ...
  Data data = new Data();
  SharedPreferences preference = getPreferences(MODE_PRIVATE);
  data.load(preference);
  this.data = data;
  EditText emailAddressEdit = (EditText)findViewById(R.id.editText1);
  emailAddressEdit.setText(data.emailAddress);
  ...
}
@Override protected void onStop() {
  super.onStop();
  EditText emailAddressEdit = (EditText)findViewById(R.id.editText1);
  data.emailAddress = emailAddressEdit.getText().toString();
  ...
  data.store(getPreferences(MODE_PRIVATE));
}
```

Exemple avec des préférences

```
public class MainActivity3 extends Activity {
  private transient Data data;

  static class Data {
    private String emailAddress;
    private String message;

    public void load(SharedPreferences preferences) {
      emailAddress = preferences.getString("emailAddress", null);
      message = preferences.getString("message", null);
    }

    public void store(SharedPreferences preferences) {
      Editor editor = preferences.edit();
      editor.putString("emailAddress", emailAddress);
      editor.putString("message", message);
      editor.commit();  // apply() if API level >= 9
    }
  }
```

Presse papier & copier/coller

Un ClipboardManager représente le presse papier

getSystemService(Context.CLIPBOARD_SERVICE)

Copier dans le presse papier

Créé un ClipData.Item que l'on met dans un ClipData

- ClipData.Item item = ClipData.new*
- new ClipData(ClipDescription, ClipData.Item)

Mettre dans le presse papier

- clipboardManager.setPrimaryClip(ClipData data)

Coller du presse papier

On récupère le ClipData du presse papier

- ClipData clipData = ClipboardManager.getPrimaryClip()

On regarde la description

- ClipData.getDescription()

On récupère l'item

- ClipData.Item item = ClipData.get(0)

Différent type de donnée

Il est possible de transmettre autre chose que du texte

Un ClipData.Item est de type:

- Texte ou HTML
  - ClipData.Item ClipData.new[Plain|Html]Text(label, CharSequence text)
  - Item.get[Plain|Html]Text()
- URI
  - ClipData.Item ClipData.newUri(ContentResolver resolver, label, URI uri)
  - ClipData.Item ClipData.newRawUri(label, URI uri)
- Intent
  - ClipData.newIntent(label, Intent intent)

Fichiers internes de l'application

Android sandbox les fichiers de l'application

1 seul répertoire contient tous les fichiers propre à l'application File context.getFilesDir()

- Les fichiers ne sont accessibles que par l'application
- Le répertoire est créé à l'installation de l'application
- Le répertoire est détruit à la désinstallation de l'application

Système de fichiers interne est en UTF8 et peut être crypté donc impossible d'y accéder hors de l'API

Fichiers internes de l'application

Opérations sur un Context (relatif à l'application)

FileInputStream openFileInput(String name)

FileOutputStream openFileOutput(String name, int mode)

File deleteFile(String name)

File getDir(String name, int mode)

- création du répertoire si nécessaire (avec mode)

String[] fileList()

- Listes de tous les fichiers

Flag pour les modes de créations

- MODE_PRIVATE: accessible par l'application uniquement (ou autre application ayant le même user ID)
- MODE_APPEND: ajout à la fin (par défaut, écrasement du fichier)

Repertoire cache

Utile pour stocker les données temporaires

Résultat de calcul, donnée en cache accessible si serveur indisponible, etc...

Répertoire de cache propre à l'application

File context.getCacheDir()

Les données peuvent être effacées par le système

- En cas de désinstallation de l'application
- En cas de pénurie de mémoire de stockage
  - Le système vire en priorité les données des gros consommateurs :)

Fichiers externes

Les fichiers externes sont

publique visible par tout le monde

- Habituellement, FAT non crypté

sur carte SD (par USB ou non)

Les fichiers sont

soit organisés par application

soit organisés par type de donnée (music, pictures, etc)

Fichiers externes de l'application

Obtenir un répertoire racine externe

peut être null si pas de carte SD par exemple !

Global à toutes les applications

context.getExternalStoragePublicDirectory()

par type de donnée

context.getExternalStoragePublicDirectory(String type)

Spécifique à l'application

context.getExternalFilesDir(String type) (type peut être null)

En cache

context.getExternalCacheDir()

type de fichiers organisés par sous-répertoire

DIRECTORY_* ([MUSIC|MOVIES|PICTURES|...)

Sur le cloud !

- Possibilité de spécifier un agent de sauvegarde des données vers le cloud
  - L'utilisateur du téléphone indique le could à utiliser
    - souvent Google Drive
- Balise <application> au niveau du fichier AndroidManifest.xml
  - android:allowBackup
  - android:backupAgent: classe Java qui hérite de BackupAgent

BackupAgent

La classe BackupAgent appel pour

réaliser une sauvegarde **incrémentale** de oldState vers newState en écrivant les données binaires dans data

- onBackup(ParcelFileDescriptor oldState, BackupDataOutput data, ParcelFileDescriptor newState)

restaurer une sauvegarde

- onRestore(BackupDataInput data, int appVersion, ParcelFileDescriptor newState)

Il existe des BackupAgentHelpers pour chaque type de données courantes (fichier, préférence, etc)

BackupAgent

De plus, l'application peut elle même demander un backup incrémental

- BackupManager.dataChanged()

Attention à ne pas appelée cette méthode trop souvent !

SQLite3

- Moteur de base de données relationnel sur fichier sans support de concurrence
- Persistence de structure de donnée en tables
  - Habituellement, 1 ligne 1 objet, un champ par colonne
- Jointure, index de tri, etc.

Gestions des tables

- Création

  CREATE TABLE IF NOT EXISTS gpstrace(date INTEGER PRIMARY KEY, latitude REAL NOT NULL, longitude REAL NOT NULL, altitude REAL) ;

- Création d'index

  CREATE INDEX IF NOT EXISTS latitude ON gpstrace (latitude)

- Effacement de table

  DROP TABLE gpstrace

Requètes

- Selection

  SELECT col1,col2,...,coln FROM table WHERE expr GROUP BY expr HAVING expr {UNION, INTERSECT, EXCEPT} SELECT ...

- Insertion

  INSERT INTO table (col1,col2,...,coln) VALUES (val1,val2,...,valn)

- Mise à jour

  UPDATE table SET col1=val1, col2=val2, ...,coln=valn WHERE expr

- Suppression

  DELETE FROM table WHERE expr

  Expressions

  - Opérateurs classiques : || * / % + - << >> & | < <= > >= == != NOT
  - Like et regex: cloname LIKE x: (% pour .* et _ pour ?) ou colname REGEXP r

SQLiteOpenHelper

Helper dont on hérite pour ouvrir une base

1 constructeur

- SQLiteOpenHelper(context, dbName, null, version)

2 méthodes à redéfinir

- exécuter le scripte de création de la base (table + index)
  - onCreate(SQLiteDatabase) pour
- Mettre à jour une base de la version v1 à la version v2
  - onUpgrade(SQLiteDatabase, int v1, int v2)

Obtenir la base en lecture ou lecture/écriture

- get[Readable|Writable]Database()

SQLiteDatabase

Executer une requète

- Cursor rawQuery(String query, String[] selectionArgs)
- Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
  - Bien séparer sélection et selectionArgs pour éviter l'injection de commande SQL
- Exemple:
  - Cursor c = db.rawQuery("SELECT date, latitude, longitude FROM gpstraces WHERE latitude > ? ORDER BY date DESC limit 100", new String[] {"45"})
  - Cursor cursor = db.query("gpstraces", new String[] {"date", "latitude", "longitude"}, "latitude > ?", new String[]{"45"}, null, null, "date DESC", 100)

Cursor

On parcourt les résultats d'une requête avec un Cursor (pas thread safe)

Méthodes usuelles

- nombre de lignes
  - int getCount()
- Valeur de la colonne à l'index de la ligne courante
  - X get[Short|Int|Long|Float|Double|String|Blob](index)
- Bouger à l'élement prochain ou renvoie false
  - boolean moveToNext()

Ecriture

- Insertion
  - db.insert(String table, String nullColumnHack, ContentValues values)
- Mise à jour
  - db.update(String table, ContentValues values, String whereClause, String[] whereArgs)
- Suppression
  - db.delete(String table, String whereClause, String[] whereArgs)

  avec ContentValues est une Map (column name -> value)