Broadcast Receivers

Broadcast Receivers

- un *broadcast receiver* permet de recevoir et traiter des *intents* diffusés
- la diffusion d'un *intent* est réalisée par la méthode **sendBroadcast**
- on le définit en héritant de la classe BroadcastReceiver
  - implanter la méthode **onReceive**
- un *broadcast* receiver peut traiter un *intent* même si l'application associée n'est pas démarrée
- durée de vie d'un *broadcast receiver* :
  - une instance n'existe que le temps de traiter un *intent*
- le système envoie un *intent* à tous les *broadcast receivers* abonnés par ordre de priorité (telle que définie dans le fichier manifest)
- un *broadcast receiver* peut interrompre la réception d'un *intent* aux éléments de priorité inférieur en appelant la méthode **abortBroadcast()**.

Enregistrement d'un *broadcast receiver*

On ajoute un noeud `<receiver>` au fichier manifest, celui-ci doit inclure une balise `intent-filter` spécifiant l'*intent* à traiter.

```
<receiver name=".MyBroadcastReceiver">
  <intent-filter>
    <action android:name="com.minfo.action.My_ACTION" />
  </intent-filter>
</receiver>
```

Broadcast Receivers

Actions de diffusions Android natives

- ACTION_BOOT_COMPLETED
- ACTION_CAMERA_BUTTON
- ACTION_DATE_CHANGED
- ACTION_MEDIA_BUTTON
- ACTION_MEDIA_EJECT
- ...

Des permissions peuvent être nécessaire pour intercepter un *intent* diffusé.

Liste complète

http://developer.android.com/reference/android/content/Intent.html

Broadcast Receivers

Exemple

Un *broadcast receiver* traitant un message à afficher à l'écran.

Fichier manifest

```
<application>
  <receiver
      android:name=
        "com.minfo.broadcastreceiverdemob.ToastReceiver">
      <intent-filter>
          <action android:name="com.minfo.action.showtoast"/>
      </intent-filter>
  </receiver>
</application>
```

On précise le nom de la classe traitant l'action et le nom de l'action.

ToastReceiver

```
public class ToastReceiver extends BroadcastReceiver{

  @Override
  public void onReceive(Context context, Intent intent) {
    String msg = intent.getExtras().getString("message");
    Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
  }
}
```

`getExtras()` retourne une instance de la classe Bundle qui permet d'enregistrer et consulter des paires clé-valeur

Broadcast Receivers

Utilisation depuis une activité d'une autre application

```
public void onClick(View v){
  Intent intent = new Intent();
  intent.setAction("com.minfo.action.showtoast");
  intent.putExtra("message", edittext.getText().toString());
  this.sendBroadcast(intent);
}
```

Persistance de données

- `Bundle`
- *Shared preferences*
- Fichiers
- Bases de données `SQLite`
- *Content Provider*

Bundle

Bundle

Objet permettant d'enregistrer des paires clés-valeurs, utilisé en particulier par les activités pour restaurer leur état sauvegardé

- état récupéré dans la méthode `onCreate(Bundle savedInstanceState)`
- état enregistré par la méthode `onSaveInstanceState(Bundle onSaveInstanceState))`

De base, l'activité enregistre ses éléments internes, il est possible d'ajouter des éléments en redéfinissant la méthode `onSaveInstanceState`

Attention : ces données ne sont pas persistantes, elle ne sont conservée que pour le cycle de vie d'une instance de l'application.

Bundle

Fichiers

Fichiers

Classes et méthodes Java I/O standard disponibles

Accès simplifié aux fichiers locaux

- à chaque application est associé un répertoire `data/data/package.of.the.application`
- l'accès aux fichiers se fait par les méthodes `openFileInput(String s)` et `openFileOutput(String s, int perm)` du contexte de l'application.
- `s` est le nom du fichier
- `perm` la permission associée
  - `MODE_PRIVATE` : privé
  - `MODE_WORLD_READABLE` : accès en lecture aux autres applications
  - `MODE_WORLD_WRITABLE` : accès en écriture aux autres applications
  - `MODE_WORLD_READABLE | MODE_WORLD_WRITABLE`

Autoriser d'autres applications à accéder aux fichiers est déconseillé on utilisera plutôt des *content providers*.

Fichiers

Exemple, application A

```
try {
  FileOutputStream fos =
    openFileOutput("file.tmp", MODE_WORLD_READABLE);
  fos.write("Hello World".getBytes());
  fos.close();
}
```

Exemple, application B

```
try {
  Context context = createPackageContext("com.minfo.filedemoa",0);
  FileInputStream fos = context.openFileInput("file.tmp");
  byte[] buffer = new byte[11];
  fos.read(buffer);
  fos.close();
}
```

`createPackageContext` permet de récupérer le context de l'application A à partir du nom de son *package*.

Shared Preferences

Shared Preferences

Principe

Permettre aux applications de manipuler des paires clé-valeurs persistentes.

La classe `SharedPreferences`

- les paires sont accessibles au travers d'instances de la classe

  `SharedPreferences`

- une instance de la classe `SharedPreferences` est obtenue en appelant la méthode `getSharedContext` d'une instance de la classe `Context`, généralement celle associée à l'application

  ```
  getApplicationContext().getSharedPreferences("MyPref", mode)
  ```

  - `"MyPref''` est une chaîne de caractère quelconque identifiant un fichier dans lequel les préférences sont enregistrées [a]
  - `mode` type de permission, on peut accéder aux préférences d'une autre application en utilisant la même méthode que pour les autres fichiers

[a] Ce fichier appartient à l'application et se situe dans [APP DIR]/shared_prefs/MyPref.xml

Shared Preferences

Depuis une activité

La classe `Activity` héritant de la classe `Context` elle dispose également d'une méthode `getSharedContext`. Par défaut, celle-ci donne accès aux préférences partagées de l'application, les deux appels suivants sont donc équivalent depuis une activité

- `getApplicationContext().getSharedPreference("MyPref'', MODE_PRIVATE);`
- `[this.]getSharedPreferences("MyPref'', MODE_PRIVATE)`

Préférences d'une activité

La classe activité dispose également d'une méthode `getPreferences` permettant d'accéder à des préférences propres à une activité donnée. Depuis une activité dont le nom est "MyActivity", les deux appels suivants sont équivalents :

- `getSharedPreferences(MODE_PRIVATE)`
- `getSharedPreferences("MyActivity",MODE_PRIVATE)`

Les préférences d'une activité ne sont qu'un cas particulier de préférences dont l'identifiant est le nom de la classe.

Shared Preferences

Lecture

La lecture de valeurs depuis une instance de la classe `SharedPreference` se fait au travers de méthodes de cette instance de la forme

$$\texttt{get}\mathit{Type}\texttt{(}\mathit{entry\_id}\texttt{,default\_}\mathit{value}\texttt{)}$$

- *Type* : type de donnée
- *entry_id* : clée
- *default_value* : valeur retournée si la clée n'est pas présente

Ecriture

L'écriture ne se fait pas directement mais au travers de l'instance de la classe `SharedPreferences.Editor` associée à l'instance de la classe `SharedPreference`.

- une telle instance est obtenue par un appel à la méthode d'instance `edit()` de la classe `SharedPreferences`
- la classe `SharedPreferences.Editor` dispose de méthodes d'instance de la forme

$$\texttt{put}\mathit{Type}(\mathit{entry\_id}, \mathit{value})$$

  - *Type* : type de donnée
  - *entry_id* : clée
  - *value* : valeur écrite

Une fois les valeurs "écrites", on réalise un appel à la méthode `commit` de l'objet de type `SharedPreferences.Editor` qui réalise l'enregistrement des données de manière atomique.

Shared Preferences

Exemple

Une application disposant de deux activité

- la première `MainActivity.java` permet de lire et écrire une préférence `"MyPref_entry"` associée au fichier `MyPref`
- la seconde `SecondActivity.java`, démarrée depuis la première, affiche la valeur enregistrée

Shared Preferences

Exemple : MainActivity.java

```
EditText edittext;
SharedPreferences preferences;

public void onCreate(Bundle savedInstanceState) {
   ...
   preferences = getSharedPreferences("MyPref", MODE_PRIVATE);
}

// read, write and launch button
public void read(View v){
   String msg=preferences.getString("MyPref_entry", "No value");
   edittext.setText(msg);
}

public void write(View v){
   SharedPreferences.Editor editor = preferences.edit();
   editor.putString("MyPref_entry",
   edittext.getText().toString());
   editor.commit();
}

public void launch(View v){
   startActivity(new Intent(this,SecondActivity.class));
}
```

Shared Preferences

Exemple : SecondActivity.java

```
SharedPreferences preferences;
TextView textview;

public void onCreate(Bundle savedInstanceState) {
  ...
  preferences = getSharedPreferences("MyPref", MODE_PRIVATE);
  String msg = preferences.getString("MyPref_entry", "Unknown");
  textview.setText(msg);
}
```

Depuis une autre application

Dans certains cas très limités, une application peut accéder au préférences d'une autre application.

- les deux applications doivent partarger le même identifiant utilisateur ce qui suppose qu'elle soient signées par le même développeur et en faire explicitement la demande en ajoutant une option dans la section `manifest` de leur fichier *Manifest*

  ```
  android:sharedUserId="minfo.uid.shared"
  ```

- l'application ouvrant ces préférences doit associer la bonne permission au fichier associé

  ```
  getSharedPreferences("MyPref'', MODE_WORLD_READABLE) ou
  getSharedPreferences("MyPref'', MODE_WORLD_WRITEABLE)
  ```

  en fonction du type d'accès souhaité.

Shared Preferences

Exemple

Deux applications `SharedPreferencesDemo2` et `SharedPreferencesDemo3`.

- la première enregistre une préférence `"MyPref_entry"` associée au fichier `"MyPref"`

Accès au contexte d'une autre application

Pour accéder aux préférences de la première application la seconde doit récupérer son contexte

```
Context context =
createPackageContext("minfo.examples.sharedpreferencesdemo2",
                     CONTEXT_IGNORE_SECURITY);
```

- `"minfo.examples.sharedpreferencesdemo2''` est le nom du package de la première application
- `CONTEXT_IGNORE_SECURITY` voir classe `Context`

Shared Preferences

Exemple : fichiers Manifests

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="minfo.examples.sharedpreferencesdemo2"
android:versionCode="1"
android:versionName="1.0"
android:sharedUserId="minfo.uid.shared">
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="minfo.examples.sharedpreferencesdemo3"
android:versionCode="1"
android:versionName="1.0"
android:sharedUserId="minfo.uid.shared">
```

Shared Preferences

Exemple SharedPreferencesDemo2.MainActivity

```
SharedPreferences preferences;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  preferences = getSharedPreferences("MyPref", Context.MODE_WORLD_READ

  SharedPreferences.Editor editor = preferences.edit();

  editor.putString("MyPref_entry", "Shared Data!");

  editor.commit();
}
```

Shared Preferences

Exemple SharedPreferencesDemo3.MainActivity

```
TextView textview;
SharedPreferences preferences;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  textview = (TextView) findViewById(R.id.textview);
  Context context;
  try {
    context = createPackageContext("minfo.examples.sharedpreferencesdemo2"

    preferences = context.getSharedPreferences("MyPref",MODE_PRIVATE);

    String msg = preferences.getString("MyPref_entry", "Unknown");
    textview.setText(msg);
  } catch (NameNotFoundException e) {
    e.printStackTrace();
  }
}
```

Bases de données

SQLite

Chaque application dispose de sa propre base de donnée `SQLite` qui n'est accessible que depuis cette application.

- version limitée de `SQL` sans serveur
- implantation très efficace

Creation d'un schema de base de donnée

- Héritage de la classe `SQLiteOpenHelper`
- Redéfinition de la méthode `onCreate(SQLiteDatabase db)`
  - création des tables
- Redéfinition de la méthode `onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`
  - Changement de version, mise à jour des schémas

SQLite

```
public class MySQLiteHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "mydb";
    private static final int DATABASE_VERSION = 1;
    public static final String TABLE_NAME = "mytable";
    public static final String UID = "_id";
    public static final String NAME = "name";

    private static final String CREATE =
    "CREATE TABLE " + TABLE_NAME + " (" +
    UID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
    NAME + " VARCHAR(255) );";
    private  static final String DROP =
    "DROP TABLE IF EXISTS " + TABLE_NAME + ";";

    public MySQLiteHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);}

    public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREATE);}

    public void onUpgrade(SQLiteDatabase db,
                int oldVersion, int newVersion) {
    db.execSQL(DROP);}
}
```

SQLite

```
MySQLiteHelper helper = new MySQLiteHelper(this);

// Acces en lecture
SQLiteDatabase db = helper.getReadableDatabase();

// Acces en ecriture
SQLiteDatabase db = helper.getWritableDatabase();
```

SQLite

```
// Insertion
String insertQuery = "INSERT INTO " + MySQLiteHelper.TABLE_NAME +
" (" + MySQLiteHelper.NAME + ") VALUES ('nom2');";
db.execSQL(insertQuery);

// Selection
String query = "SELECT " + MySQLiteHelper.UID + ", " +
MySQLiteHelper.NAME + " FROM " +
MySQLiteHelper.TABLE_NAME + ";";
Cursor c2 = db.rawQuery(query, null);
while (c2.moveToNext()){
  int id = c2.getInt(c2.getColumnIndex(MySQLiteHelper.NAME));
  String name =
  c2.getString(c2.getColumnIndex(MySQLiteHelper.NAME));
  Log.d("DB0","id : " + id + " name : " + name);
}
c2.close();
```

SQLite

```
// Insertion
ContentValues cv = new ContentValues();
cv.put(MySQLiteHelper.NAME, "F. Dabrowski");
db.insert(MySQLiteHelper.TABLE_NAME, MySQLiteHelper.NAME, cv);

//Selection
String table = MySQLiteHelper.TABLE_NAME;
String[] colums =
    new String[]{MySQLiteHelper.UID, MySQLiteHelper.NAME};
Cursor c = db.query(table, columns, null, null, null, null, null);

c.moveToFirst();
while (c.moveToNext()){
  int id = c.getInt(c.getColumnIndex(MySQLiteHelper.NAME));
  String name = c.getString(c.getColumnIndex(MySQLiteHelper.NAME));
  Log.d("DB","id : " + id + " name : " + name);
}
c.close();
```

android.database.Cursor

Type de retour des requêtes fonctionnant comme un itérateur.

- méthodes de parcours

  `move[ , ToFirst, ToLast, ToNext, ToPosition, ToPrevious]`

- obtenir les valeurs de la ligne courante

  `get[Double, Float, Int, Long, Short, String]`

  prend en paramètre l'index de la colonne

- obtenir l'index correpondant à un nom de colonne

  `getColumnindex[ , orThrow](String ColumnName)`

- nombre de colonnes

  `getColumnCount()`

- Libérer les ressources

  `close()`

`android.content.ContentValues`

Abstraction simplifiant l'insertion, la mise à jour ou la suppression de lignes.

- constructeur de base `ContentValues()`
- ajout de valeurs

  `put(String key, Type value)`
- lecture et test de présence d'une valeur

  `getAsType(String key)` `containsKey(String key)`

Insertion

```
SQLDatabase::insert(String table, String nullColumnHack,
                    ContentValues cv):long
```

- table : nom de la table
- nullColumnHack : nom de colonne recevant la valeur null si la valeur est vide
- cv : ligne à insérer

Mise à jour

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

- table : la table
- values : nouvelles valeurs
- whereClause : lignes à modifier, toutes si valeur `null`
- whereArgs : arguments de la clause where

Supression

Semblable aux autres méthodes

SQLite

- students

_id	student_name	state	grade

- courses

_id	course_name

- classes

_id	student_id	course_id

```
public class ClassTable {

  public static final String ID = "_id";
  public static final String STUDENT_ID = "student_id";
  public static final String COURSE_ID = "course_id";
  public static final String TABLE_NAME = "classes";
}

public class CourseTable{
  ...
}

public class ClassTable{
  ...
}
```

```
public class MySQLiteHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "course_db";
    private static final int DATABASE_VERSION = 1;

    public MySQLiteHelper(Context context){
    super(context,DATABASE_NAME, null,DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

    db.execSQL("CREATE TABLE " + StudentTable.TABLE_NAME + "(" +
            StudentTable.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            StudentTable.NAME + " TEXT, " +
            StudentTable.GRADE + " INTEGER " + ");");
    // ...

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newV
    // TODO Auto-generated method stub
    db.execSQL("DROP TABLE IF EXISTS" + StudentTable.TABLE_NAME +";")
    //...
    onCreate(db);
    }


}
```

```
public  long addStudent(String name, int grade){
    ContentValues cv = new ContentValues();
    cv.put(StudentTable.NAME, name);
    cv.put(StudentTable.GRADE, grade);
    SQLiteDatabase db = getWritableDatabase();
    long result = db.insert(StudentTable.TABLE_NAME, StudentTable.NAM
    return result;
}

public long addCourse(String name){
    //...
}

public long enrollStudentClass(int studentId, int courseId){
    // ...
}
```

```
    public Set<Integer> getStudentsByGradeForCourse(int courseId, int
SQLiteDatabase db = getReadableDatabase();
                String[] cols;
                String[] selectionArgs;
                Cursor c;
                Set<Integer> returnIds, gradeIds;

                // Premiere requete

                // Seconde requete

returnIds.retainAll(gradeIds);
return returnIds;
}
```

```
    cols = new String[] {ClassTable.STUDENT_ID};
    selectionArgs = new String[] {String.valueOf(courseId)};

    c = db.query(ClassTable.TABLE_NAME, cols, ClassTable.COURSE_ID + " = ?"
    null, null, null);

    returnIds = new HashSet<Integer>();
    while (c.moveToNext()){
      int id = c.getInt(c.getColumnIndex(ClassTable.STUDENT_ID));
      returnIds.add(id);
    }
```

```
    cols = new String[]{StudentTable.ID};
    selectionArgs = new String[] {String.valueOf(grade)};

    c = db.query(StudentTable.TABLE_NAME, cols, StudentTable.GRADE + "= ?"
    null,null,null);
    gradeIds = new HashSet<Integer>();
    while (c.moveToNext()){
      int id = c.getInt(c.getColumnIndex(StudentTable.ID));
      gradeIds.add(id);
    }
```

Query

```
cols = new String[] {ClassTable.STUDENT_ID};
selectionArgs = new String[] {String.valueOf(courseId)};

c = db.query(ClassTable.TABLE_NAME, cols, ClassTable.COURSE_ID +
" = ?", selectionArgs, null, null, null);
```

- cols : colonnes cibles de la selection
- selectionArgs : tableau d'arguments pour la clause `WHERE` (`ClassTable.COURSE_ID + "= ?''`)

```
SELECT student_id FROM classes WHERE course_id = courseId
```

Query

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

- groupe les résultat selon la colonne donnée
- selection sur les groupes
- résultat ordonnée selon les colonnes données

Il n'y pas de contraintes sur les clés, il faut donc assurer les règles du schema manuellement.

Delete

```
public boolean removeStudent(int studentId){
    SQLiteDatabase db = getWritableDatabase();

    String[] whereArgs = new String[] {String.valueOf(studentId)};
    db.delete(ClassTable.TABLE_NAME, ClassTable.STUDENT_ID + "= ?", wh

    int result = db.delete(StudentTable.TABLE_NAME, StudentTable.ID + "= ?"
    return (result > 0);
}
```

```
long sid1 = helper.addStudent("name1", 3);
long sid2 = helper.addStudent("name2", 4);
long sid3 = helper.addStudent("name3", 5);

long cid1 = helper.addCourse("Secu");
long cid2 = helper.addCourse("Android");

helper.enrollStudentClass(sid1, cid1);
helper.enrollStudentClass(sid1, cid2);
helper.enrollStudentClass(sid2, cid1);
helper.enrollStudentClass(sid3, cid1);

Cursor c = helper.getStudentsForCourse((int) cid2);
while (c.moveToNext()){
  int sid = c.getInt(c.getColumnIndex(ClassTable.STUDENT_ID));
  Log.d("DB","STUDENT " + sid + "IS ENROLLED IN COURSE" + cid2);
}
```

Accès direct à la base

- `adb shell` : connexion en mode console
- `sqlite3 /data/data/<package>/databases/<mabase>.db`

  `sqlite3 /data/data/minfo.examples.sqlitecourses/databases/courses_db.db`
- `.help` pour voir la liste des commandes, par exemple
  - voir les tables de la base
  - ecriture de requetes SQLite
  - ...

Requêtes

- méthode bas-niveau

```
Cursor rawQuery(String sql, String selectionArgs)
```

  - sql : chaîne de caractère représentant la requête
  - selectionArgs : arguments de la clause `WHERE`

- méthode haut-niveau

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

Content Providers

Content Providers

Content Provider

Interface pour la publication et la cosommation de données basée sur un adressage par URI utilisant le schema `content`.

- les *content providers* partagés peuvent être consultés, leur enregistrements peuvent être mis à jour ou supprimés et de nouvelles données peuvent y être ajoutées.
- toute application disposant de permissions appropriées peut effectuer ces applications.

Exemple : liste des contacts

- sur un système android la liste des contacts se trouve dans une base de donnée accessible par un *content provider*,
  - le programmeur n'a pas besoin de connaitre le schéma de la base, si celui-ci change il n'y a pas de conséquence sur le code tant que le content provider ne change pas.

Mettre en place un content provider

- Définir le modèle de donnée
  - en général une base de données, mais pas forcément
- Choisir un URI (Unifier Ressource Identifier)
- Déclarer le content provider dans le fichier manifest en précisant au moins
  - la classe implantant le *content provider*
  - l'URI du *content provider*
- implanter les méthodes abstraites de la classe `ContentProvider`

`query`	`insert`	`update`
`delete`	`getType`	`onCreate`

ContentProvider

Accéder à un *content provider*

L'accès à un *content provider* se fait par l'intermédiaire d'un *content resolver* accessible par le contexte de l'application.

- un *content resolver* expose les méthodes corresponsdant aux méthodes que l'on implante dans le *content provider* par lesquelles l'accés à ce dernier est réalisé.

ContentResolver

URI du *content provider*

Elle doit être propre au *content provider*, la forme conseillée est

`content://com.<CompanyName>.provider.<ApplicationName>/<DataPath>`

où `<DataPath>` représente généralement :

- une table

  `content://com.minfo.provider.reservation/salle`

- une ligne donnée d'une table

  `content://com.minfo.provider.reservation/salle/1`

Gestion des URI

Deux classes sont particulièrement utiles android.content.UriMatcher et android.content.ContentUris

Méthodes de la classe `Content Provider`

- query
- insert
- delete
- update
- getType
- onCreate

Content Providers

Exemple : inscription étudiants

on reprend la base de données précédente et on fournit les services suivants

- consulter les listes d'étudiants, de cours et les inscriptions
- ajouter/retirer un étudiant, un cours ou une inscription
- la consultation d'un élément ou d'un retrait d'un élément pourra être fait par son identifiant

Les classes

- activités : `MainActivity`, `SeeStudents`, `AddStudents`, ...
- base de données : `ClassesSQLiteHelper`
- content provider : `ClassesContentProvider`
- tables : `StudentTable`, `CourseTable`, `ClasseTable`

Le fichier manifest

```
<application ...>
...
<provider
android:name="com.minfo.classes.ClassesContentProvider"
android:authorities="com.minfo.provider.classes"/>
...
</application>
```

- `android:name` : la classe du *content provider*
- `android:authorities` : l'URI du *content provider*

Types MIME

On définit le type associé aux tables et aux entrées des tables

- table : `vnd.android.cursor.dir/<contenttype>`
- entrée : `vnd.android.cursor.item/<contenttype>`

où `<contenttype>` est de la forme `vnd.<name>.<type>`

- `<name>` : nom unique, par exemple nom du fournisseur
- `<type>` : nom unique pour l'URI, par exemple nom de la table

StudentTable

```
public class StudentTable {

    public static final String TABLE_NAME = "students";

    public static final String ID = "_id";
    public static final String FIRSTNAME = "firstname";
    public static final String LASTNAME = "lastname";
    public static final String GRADE = "grade";

    public static final Uri CONTENT_URI =
    Uri.parse("content://" +
            ClassesContentProvider.AUTHORITY + "/students");

    public static final String CONTENT_TYPE =
    "vnd.android.cursor.dir/vnd.minfo.students";

    public static final String CONTENT_ITEM_TYPE =
    "vnd.android.cursor.item/vnd.minfo.students";

}
```

Même chose pour les autres tables

ClassesContentProvider

```
public class ClassesContentProvider extends ContentProvider {

    // URI
    public static final String AUTHORITY =
    "com.minfo.provider.classes";

    // code for services : access a table or a particular row
    private static final int STUDENTS = 1;
    private static final int COURSES = 2;
    private static final int STUDENT_ID = 3;
    private static final int COURSE_ID = 4;

    private ClassesSQLiteHelper helper;
    private UriMatcher uriMatcher;

    public boolean onCreate() {...}
    public Uri insert(...){...}
    public int delete(...) {...}
    public Cursor query(...) {...}
    public String getType(...) {...}
    public int update(...) {...}
}
```

onCreate()

```
@Override
    public boolean onCreate() {

    helper = new ClassesSQLiteHelper(getContext());

    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI(AUTHORITY, "students", STUDENTS);
    uriMatcher.addURI(AUTHORITY, "courses", COURSES);
    uriMatcher.addURI(AUTHORITY, "students/#", STUDENT_ID);
    uriMatcher.addURI(AUTHORITY, "courses/#", COURSE_ID);
    return true;
}
```

Pour filtrer des adresses de la forme `content://AUTHORITY/students/<num>`

uriMatcher.addURI(AUTHORITY, "students/#", STUDENT_ID)

- voir la documentation : android.content.UriMatcher
- la méthode `match` de la classe `UriMatcher` permettra de récupérer le code (ici `STUDENT_ID`) correspondant à l'URI.

insert

```
@Override
public Uri insert(Uri uri, ContentValues values) {

    SQLiteDatabase db = helper.getWritableDatabase();

    switch(uriMatcher.match(uri)){
    case STUDENTS :
    ...
    case COURSES :
    ...
    default :
    String msg = "Invalid URI for insertion " + uri;
    throw new IllegalArgumentException(msg);
    }
}
```

Ici, on rejette les URI mentionnant un numéro de ligne

```
case STUDENTS :
long studentId = db.insert(StudentTable.TABLE_NAME, null, values);
if (studentId > 0){
    Uri studentUri = ContentUris.withAppendedId(uri, studentId);
    getContext().getContentResolver().notifyChange(studentUri, null);
    return studentUri;
}
else throw new SQLException("Failed to insert row into " + uri);
case COURSES :
// Similaire
```

- On construit l'URI correspondant à la nouvelle entrée
- Une notification de changement de l'URI est envoyée, le paramètre `null` indique qu'on utilise les observeurs par défaut (les objets `CursorAdapter`).

delete

```
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {

    int count;
    SQLiteDatabase db = helper.getWritableDatabase();

    switch (uriMatcher.match(uri)){
    case STUDENTS :
    count = db.delete(StudentTable.TABLE_NAME,
              selection, selectionArgs);
    break;
    case COURSES : // idem
    case STUDENT_ID : ...
    case COURSE_ID : ...
    default :
    String msg = "Invalid URI for deletion " + uri;
    throw new IllegalArgumentException(msg);
    }
    return count;
}
```

delete

```
case STUDENT_ID :

long studentId = ContentUris.parseId(uri);
String selectionWithStudentId = StudentTable.ID + "=" + studentId;

if (selection != null)
    selectionWithStudentId += " AND " + selection;

count = db.delete(StudentTable.TABLE_NAME,
          selectionWithStudentId, selectionArgs);
break;
```

On récupère l'identifiant de la ligne et on modifie la requête en fonction

Permissions

Permissions

Catégories

- **normal** : sans dangerosité pour l'utilisateur (e.g changé le fond d'écran), l'application doit déclarer son intention de l'utiliser mais l'utilisateur n'a pas besoin de l'authoriser (il peut en revanche consulté les permissions accordées à l'application).
- **dangerous** : potentiellement dangereux pour l'utilisateur, il doit explicitement l'authorisée
- **signature** : permission automatiquement accordée à une application si elle est signée par le même certificat que l'application qui déclare la permission.
- **signatureOrSystem** : permission accordées aux applications du système `android` ou aux applications signées avec le même certificat.

Permissions

```
<permission
    android:name="com.minfo.perm.MY_PERMISSION"
    android:label="Custom permissionl"
    android:description="text describing the permission"
    android:protectionLevel="dangerous"
    android:permissionGroup="android.permission-group.PERSONAL_INFO" />
```

- `label`, `description` : description courte et longue de la permission (important pour la comprehension par l'utilisateur de l'action en cause)
- `permissionGroup` groupe d'appartenance de la permission, utilisée par le système pour organiser l'affichage des permissions à destination de l'utilisateur. Voir la liste dans

  Manifest.permission_group

Permissions

Pour obtenir la permission depuis une autre application (fichier manifest)

```
<uses-permission
    android:name="com.minfo.perm.MY_PERMISSION" />
```

La syntaxe est la même pour une permission préféfinie.

Limiter l'accès à un composant

On ajoute un attribut `android:permission` à la balise du composant (application, activité, service, broadcast receiver, content provider) dans le fichier manifest de l'application.

Restrictions

- application : soumet tous les composants de l'application à la détention de la permission.
- activité : restriction sur qui a la possibilité de démarrer l'activité
- service : restriction sur qui peut démarrer le service ou s'y lier
- broadcast receiver : restriction sur qui peut diffuser des messages au broadcast receiver
- content provider : restriction sur qui peut accéder aux données

Vérification à petit-grain de permission

Il peut être utile d'utiliser un grain plus fin lors de la vérification de permissions, par exemple lorsqu'une méthode est appelée au travers de communication inter-process comme dans le cas de la liaison de services. Dans ce cas on utilise la méthode `Context.checkCallingPermission()` depuis la méthode concernée.