LẬP TRÌNH ANDROID CĂN BẢN

Bài 6: Storage

Ths. Trần Xuân Thanh Phúc | Trường Đại học Công Nghiệp Thực Phẩm

Nội dung

- File
- Share Preferences
- SQLite
- Realm
- ContentProvider

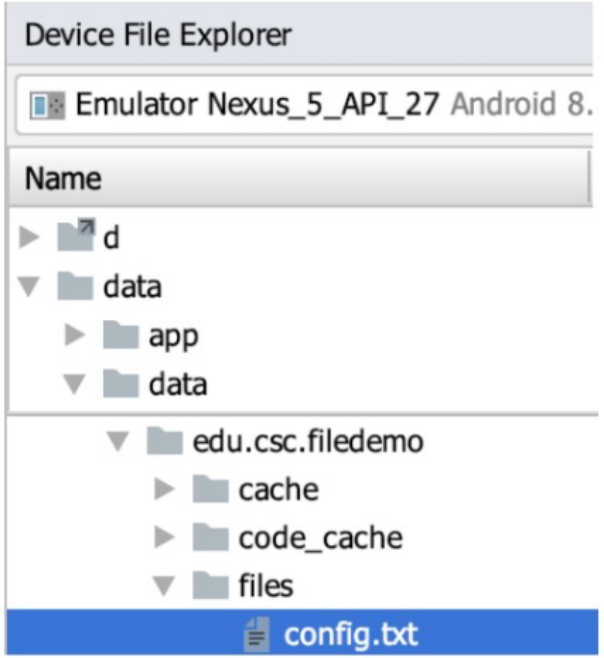
1. File

- Bộ nhớ trong được cấp phát khi ứng dụng được cài đặttrên thiết bị trên một thư mục riêng biệt.
- Thông thường ứng dụng chỉ có thể xử lý được tập tin trong nội bộ của ứng dụng đó, nếu muốn xử lý các tập tin của những ứng dụng khác thì cần phải có những quyền và điều kiện đặc biệt
- Đường dẫn các tập tin được lưu trữ: data/data/<pakage_name_app>/files

1. File

- Ghi Internal File: Việc đọc/ghi file trên Android kế thừa lại từ Java API

```
public void write(Context context, String fileName, String
message) {
    File file = new File(context.getFilesDir(), fileName);
    try {
        PrintWriter pw = new PrintWriter(
                new FileWriter(file)
        );
        pw.println(message);
        pw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



1. File

- Đọc internal file

```
public String read(Context context, String fileName) {
    File file = new File(context.getFilesDir(), fileName);
    String s = "";
    try {
        Scanner scanner = new Scanner(
                new FileReader(file)
        );
        while (scanner.hasNextLine()) {
            s += scanner.nextLine() + "\n";
        }
        scanner.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return s;
}
```

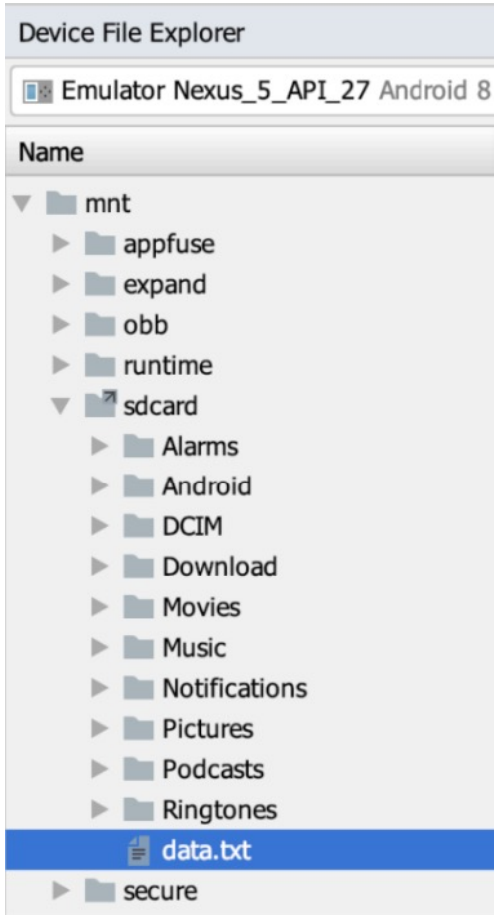
1. File

- Mỗi thiết bị Android cung cấp bộ nhớ ngoài cho phép người dùng có thể lưu trữ và truy xuất trực tiếp trên thiết bị hoặc thông qua máy tính khi kết nối USB Storage.
- Bộ lưu trữ ngoài bao gồm hai dạng:
  - Bộ nhớ thiết bị (non-removable)
  - Bộ nhớ ngoài (sd-card, usb...)
  - Cần xin cấp quyền để truy xuất bộ nhớ này
    - android.permission.WRITE_EXTERNAL_STORAGE
    - android.permission.READ_EXTERNAL_STORAGE

1. File

- Ghi external file

```
public static void writeExternalFile(String fileName, String message) {
    String filePath = Environment.getExternalStorageDirectory().getPath() + "/" + fileName;
    try {
        PrintWriter writer = new PrintWriter(new FileWriter(filePath));
        writer.println(message);
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



1. File

- Đọc external file

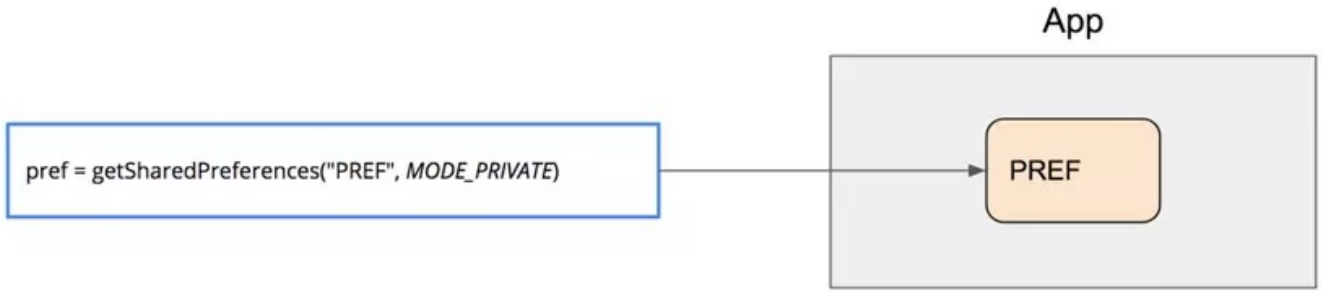
```
public static String readExternalFile(String fileName) {
    String filePath = Environment.getExternalStorageDirectory().getPath() + "/" + fileName;
    String s = "";
    try {
        Scanner scanner = new Scanner(new FileReader(filePath));
        while(scanner.hasNextLine()) {
            s += scanner.nextLine() + "\n";
        }
        scanner.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    return s;
}
```

2. Share Preference

- Shared Preferences là đối tượng Android cung cấp cho việc lưu trữ và truy xuất các dữ liệu có kiểu cơ bản như Boolean, string, float, long, và integer
- Shared Preference sẽ được lưu lại thông qua việc sử dụng chỉ định khóa cho từng giá trị dữ liệu (theo từng cặp key/value), những giá trị key.value này sẽ được tự động ghi vào một tập tin XML được chứa bên trong thư mục của ứng dụng. Những cặp giá trị này sẽ sẵn sàng cho việc sự dụng trong suốt phiên làm việc của ứng dụng và được chia sẻ bên trong các thành phần của ứng dụng (như activity, service,…)

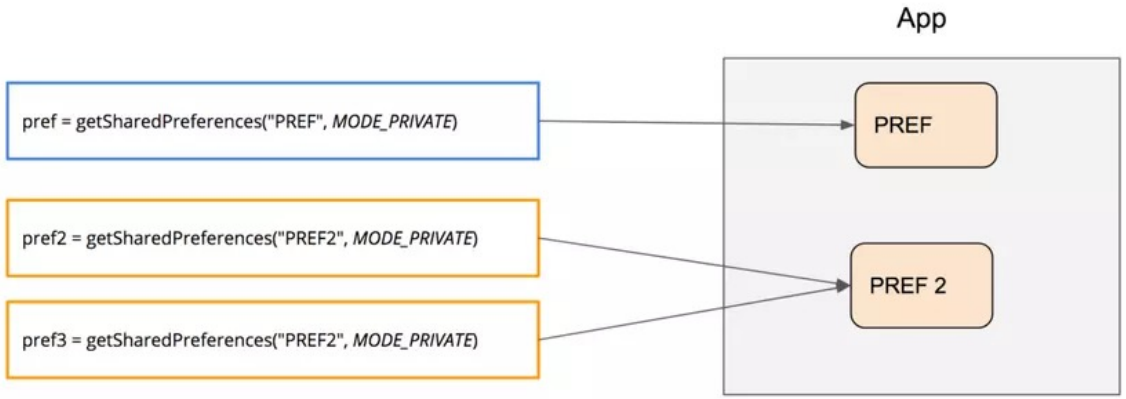
2. Share Preference

- Các giá trị shared preference này không thể truy xuất và sử dụng từ những ứng dụng khác.
- Dữ liệu của ứng dụng được lưu trữ trong thư mục data/data/Tên-package của ứng dụng.
- Trong một Context hiện tại của ứng dụng, bạn có thể truy xuất đối tượng lớp Shared Preferences thông qua hàm getSharedPreferences(String name, int mode).



2. Share Preference

- Mỗi một shared preferences sẽ có instance riêng dựa trên tên của preference.
- Không quan trọng trong context nào (Activity, Fragment, Application, ...) mà nó được gọi, nó luôn luôn trả về instance giống nhau khi được gọi method trên với tên giống nhau. Điều này đảm bảo rằng luôn luôn nhận được dữ liệu mới nhất.



2. Share Preference

- Có 3 loại Mode trong Shared Preference:
  - Context.MODE_PRIVATE – Giá trị mặc định (Không chấp nhận ứng dụng từ bên ngoài)
  - ~~Context.MODE_WORLD_READABLE~~ – Có thể đọc từ ứng dụng khác
  - ~~Context.MODE_WORLD_WRITEABLE~~ – Đọc/Ghi từ ứng dụng khác

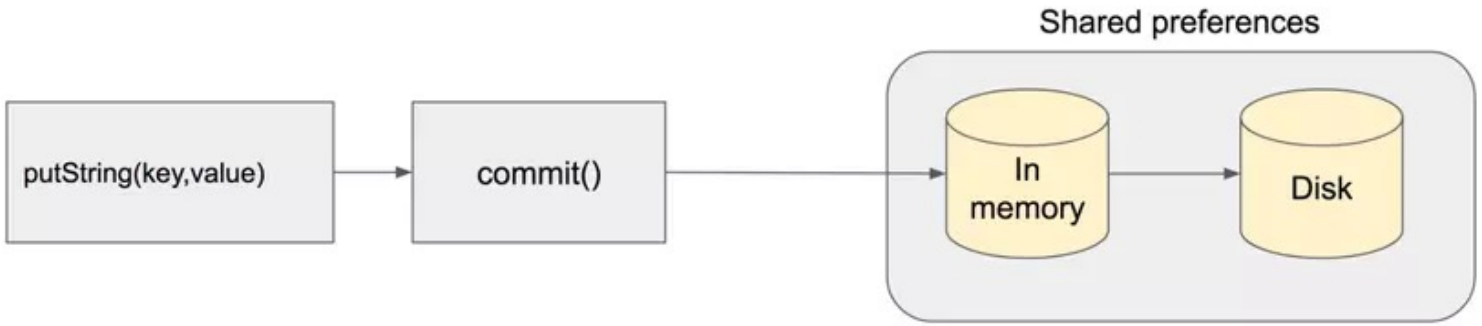
```
//======== Code to save data ====================
SharedPreferences sp = getSharedPreferences("PREF_NAME" , Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit();
editor.putInt("KEY_1", 100).commit();

//========= Code to get saved/ retrieve data ==============
SharedPreferences sp = getSharedPreferences("PREF_NAME" , Context.MODE_PRIVATE);
int data  = sp.getInt("KEY_1",0);
Log.d("Demo", "Value  is " + data);
```

Sử dụng commit() để gọi đồng bộ và trả về kết quả lưu thành công hay không hoặc apply() nếu không quan tâm đến kết quả trả về

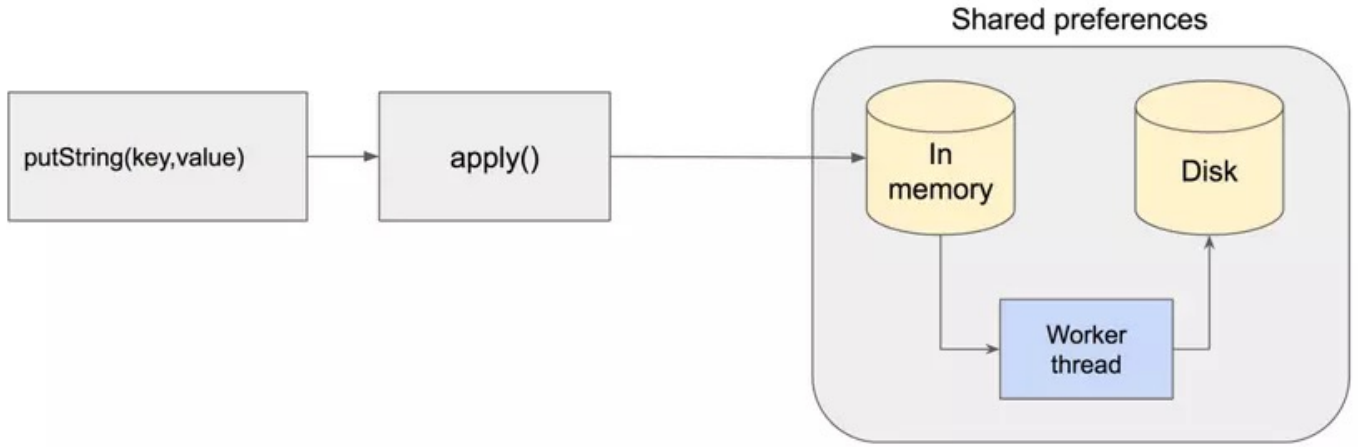
2. Share Preference

- Lưu dữ liệu thông qua commit()
  - Lưu dữ liệu vào in-memory trước sau đó thực hiện ghi vào disk một cách đồng bộ. Bởi vì có sự tham gia của thao tác I/O, main thread sẽ bị blocked cho đến khi data được ghi hết vào disk storage. Đây là một thao tác kém hiệu quả so với apply()
  - Bởi vì thao tác ghi dữ liệu lên disk storage là đồng bộ cho nên trạng thái trả về sẽ là một biến Boolean → có thể xác nhận rằng liệu thao tác mà bạn thực hiện có thành công hay không.



2. Share Preference

- Lưu dữ liệu thông qua apply()
  - Lưu giá trị vào in-memory trước sau đó ghi dữ liệu một cách bất đồng bộ vào disk-storage. Main thread sẽ không bị blocked.
  - Tuy thao tác ghi lên disk storage là asynchronous nhưng bất kỳ thao tác read ngay sau lời gọi apply() sẽ đều trả về kết quả là mới nhất bởi vì thao tác read (get) sẽ được thực thi thông qua in-memory.



3. SQLite

- SQLite là cơ sở dữ liệu mở được nhúng vào Android, hỗ trợ các đặc điểm về quan hệ chuẩn của cơ sở dữ liệu như cú pháp, transaction, các câu lệnh.
- Sử dụng SQLite không yêu cầu về thiết lập bất cứ cơ sở dữ liệu hoặc đòi hỏi quyền admin.
- Hỗ trợ các kiểu dữ liệu: TEXT, INTEGER, REAL, BLOB.

3. SQLite

- SQLite có các ưu điểm sau:
  - Tin cậy: các hoạt động transaction nội trong cơ sở dữ liệu được thực hiện trọn vẹn, không gây lỗi khi xảy ra sự cố phần cứng
  - Tuân theo chuẩn SQL92
  - Không cần cài đặt cấu hình
  - Kích thước chương trình gọn nhẹ, với cấu hình đầy đủ chỉ không đầy 300 kB
  - Thực hiện các thao tác đơn giản nhanh hơn các hệ thống cơ sở dữ liệu khách/chủ khác
  - Không cần phần mềm phụ trợ
  - Phần mềm tự do với mã nguồn mở, được chú thích rõ ràng

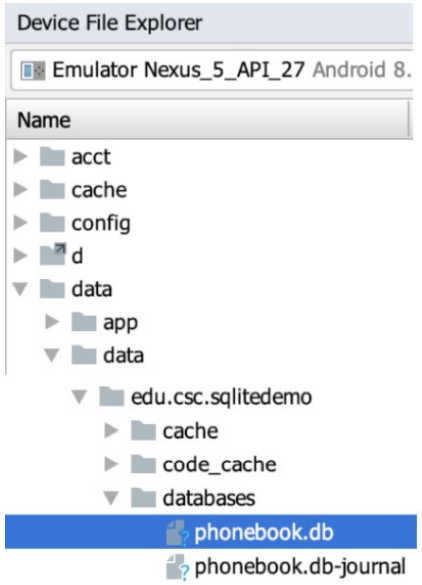
3. SQLite

- Mở internal database (hoặc tạo mới và mở nếu chưa tồn tại)

```
private String dbName = "phonebook.db";
private SQLiteDatabase openDB() {
   return openOrCreateDatabase(dbName, MODE_PRIVATE, null);
}
```

- Đóng database đã mở

```
private void closeDB(SQLiteDatabase db) {
   db.close();
}
```



3. SQLite

- Mở/tạo external database:
  - Mode:
    - CREATE_IF_NECESSARY
    - OPEN_READWRITE
    - OPEN_READONLY
- Trước khi mở/tạo external database, cần phải khai báo permissions tương ứng trong AndroidManifest.xml và request runtime permissions đối với người dùng

```
private SQLiteDatabase openExternalDB() {
    String dbName = "phonebook.edb";
    String path = Environment.getExternalStorageDirectory().getPath()
            + "/" + dbName;
    return SQLiteDatabase.openDatabase(
            path,
            null,
            SQLiteDatabase.CREATE_IF_NECESSARY
    );
}
```

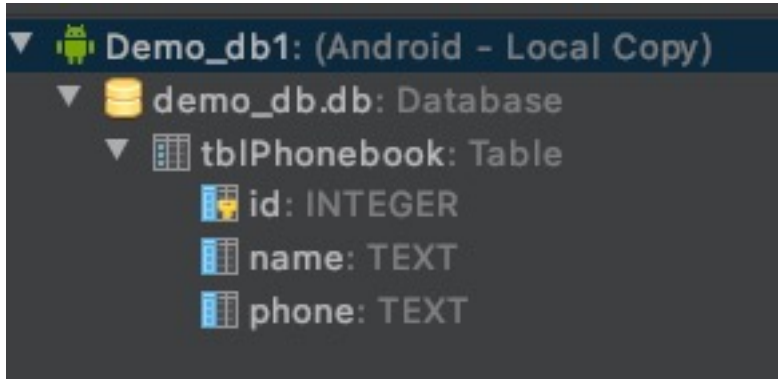
```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

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

3. SQLite

- Tạo bảng mới

```
private void createPhoneTable() {
    SQLiteDatabase db = openOrCreateDatabase("demo_db.db", MODE_PRIVATE, null);
    String sql = "create table if not exists tblPhonebook(id integer PRIMARY KEY
autoincrement, name text, phone text);";
    db.execSQL(sql);
    db.close();
}
```



3. SQLite

- Đọc dữ liệu

```
private void readContacts() {
    SQLiteDatabase db = openOrCreateDatabase("demo_db.db", MODE_PRIVATE, null);
    String sql = "select * from tblPhonebook";
    Cursor csr = db.rawQuery(sql, null);
    if (csr != null) {
        while (csr.moveToNext()) {
            System.out.println("ID: " + csr.getInt(0));
            System.out.println("Name: " + csr.getString(1));
            System.out.println("Phone: " + csr.getString(2));
        }
    }
    db.close();
}
```

3. SQLite

- Thêm dữ liệu

```
private void insert(String name, String phone) {
    SQLiteDatabase db = openOrCreateDatabase("demo_db.db", MODE_PRIVATE, null);
    ContentValues cv = new ContentValues();
    cv.put("name", name);
    cv.put("phone", phone);
    db.insert("tblPhonebook", null, cv);
    db.close();
}
```

- Cập nhật

```
private void update(int id, String name, String phone) {
    SQLiteDatabase db = openOrCreateDatabase("demo_db.db", MODE_PRIVATE, null);
    ContentValues cv = new ContentValues();
    cv.put("name", name);
    cv.put("phone", phone);
    int row = db.update("tblPhonebook", cv, "id = ?", new String[]{ String.valueOf(id) });
    Log.d(getLocalClassName(), "Affected rows: " + String.valueOf(row));
    db.close();
}
```

3. SQLite

- Xóa dữ liệu

```
private void deleteContact(int id) {
    SQLiteDatabase db = openOrCreateDatabase("demo_db.db", MODE_PRIVATE, null);
    int row = db.delete("tblPhonebook", "id = ?", new String[]{ String.valueOf(id) });
    Log.d(getLocalClassName(), "Affected rows: " + String.valueOf(row));
    db.close();
}
```

- Cách sử dụng SQLite: https://www.javatpoint.com/android-sqlite-tutorial

4. Realm

- Realm là một cơ sở dữ liệu nhẹ, không sử dụng SQLite làm engine. Thay vào đó, nó dùng core C++ nhằm mục đích cung cấp một thư viện cơ sở dữ liệu thay thế SQLite.
- Realm lưu trữ dữ liệu trong các bảng viết bằng core C++. Việc này cho phép Realm được truy cập dữ liệu từ nhiều ngôn ngữ cũng như một loạt các truy vấn khác nhau.

4. Realm

- Những ưu điểm của Realm so với SQLite:
  - Nhanh hơn so với SQLite (gấp 10 lần)
  - Dễ sử dụng
  - Hỗ trợ ORM
  - Thuận tiện cho việc tạo ra và lưu trữ dữ liệu nhanh chóng.

4. Realm

- Cách cài đặt:

// Gradle project

```
buildscript {
    repositories {
        ...
        mavenCentral()
    }
    dependencies {
        ...
        classpath "io.realm:realm-gradle-plugin:10.11.1"
    }
}
```

// Gradle app

```
apply plugin: "realm-android"
```

4. Realm – Tạo database

```
// Manifest file
<application
   android:name=".HelloApplication”
   ….
```

```
public class HelloApplication extends Application {
//    private static Realm realm;
   @Override
   public void onCreate() {
      super.onCreate();
      Realm.init(getApplicationContext());
      RealmConfiguration config = new RealmConfiguration.Builder()
            .name("demo_realm.realm")
            .schemaVersion(1)
            .deleteRealmIfMigrationNeeded()
            .build();
      Realm.setDefaultConfiguration(config);
   }

   public static Realm getRealmInstance()
   {
      return Realm.getDefaultInstance();
   }
}
```

4. Realm

- Bất kì một đối tượng JavaBean (*lớp Java đó là serializable, có một constructor mặc định, và có getter/setter cho các thuộc tính của nó*) nào cũng có thể được lưu trữ trong Realm nếu nó kế thừ từ lớp RealmObject.

```
public class SinhVienModel extends RealmObject {
    @PrimaryKey
    private long id;
    @Required
    private String name;

    public void setId(long id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public long getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

4. Realm

- Thêm dữ liệu

```
Realm r = HelloApplication.getRealmInstance();
r.beginTransaction();

int primaryKeyValue = 1;
SinhVienModel svDB = r.createObject(SinhVienModel.class, primaryKeyValue);
svDB.setName("Nguyen Van A");

r.commitTransaction();
```

- Đọc dữ liệu

```
RealmResults<SinhVienModel> svs = r.where(SinhVienModel.class).findAll();
for (SinhVienModel s : svs) {
   System.out.println(s.getName());
}
```

4. Realm

- Cập nhật dữ liệu

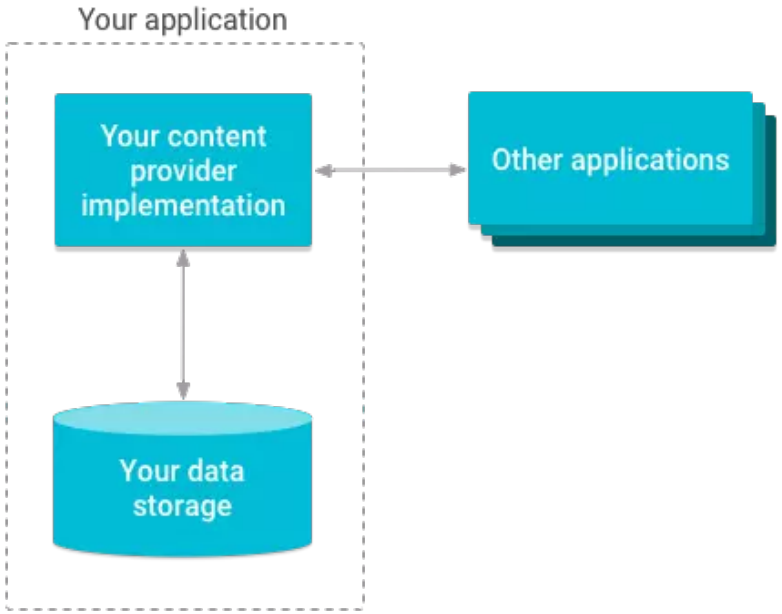
```
r.beginTransaction();
SinhVienModel sv = r.where(SinhVienModel.class).equalTo("id", 1)
        .findFirst();
sv.setName(”Le Thi B");
r.commitTransaction();
```

- Xóa dữ liệu

```
r.beginTransaction();
SinhVienModel sv = r.where(SinhVienModel.class).equalTo("id", 1)
        .findFirst();
sv.deleteFromRealm();
r.commitTransaction();
```

5. Content Provider

- Content provider là một thành phần để quản lý truy cập dữ liệu, nó cung cấp các phương thức khác nhau để các ứng dụng có thể truy cập dữ liệu từ một ứng dụng khác bằng cách sử dụng ContentResolver.

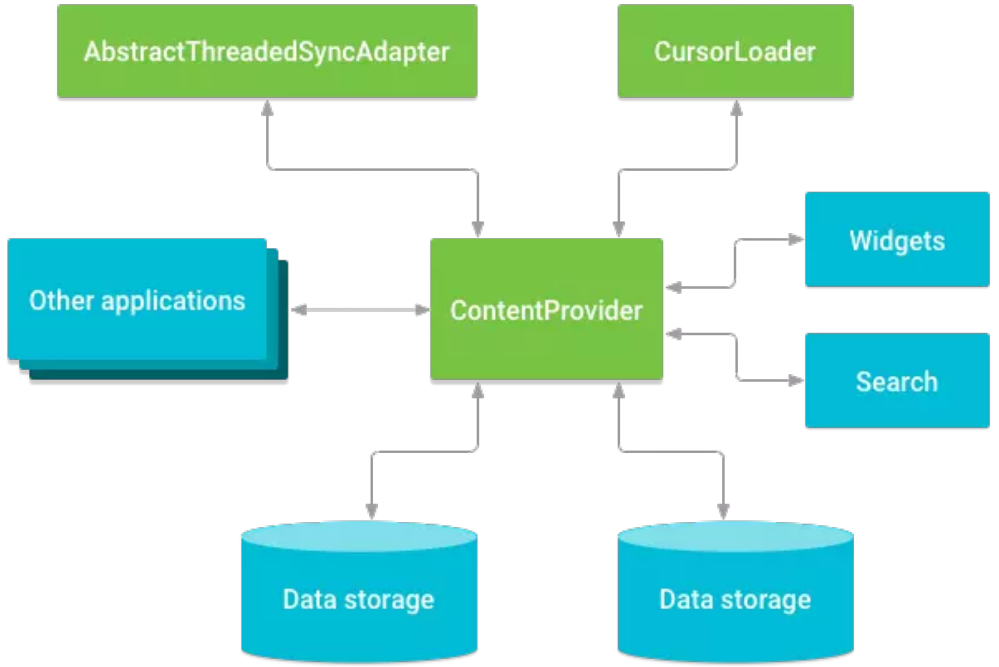


5. Content Provider

- Content Provider điều phối việc truy cập tới bỗ lưu trữ dữ liệu thông qua các API và các component, nó bao gồm:
  - Chia sẻ dữ liệu từ ứng dụng của bán tới các ứng dụng khác
  - Gửi dữ liệu sang widget
  - Trả về một kết quả gợi ý khi search cho ứng dụng của bạn thông qua Seach Framework sử dụng SearchRecentSuggestionsProvider
  - Đồng bộ dữ liệu của ứng dụng với server bằng cách sử dụng AbstractThreadedSyncAdapter
  - Tải dữ liệu lên UI sử dụng CursorLoader (deprecated in API level 28)

5. Content Provider

- Content Provider hoạt động rất giống với một cơ sở dữ liệu, có thể truy vấn, chỉnh sửa nội dung, cũng như là thêm xóa các nội dung sử dụng các phương thức: insert(), update(), delete(), query().



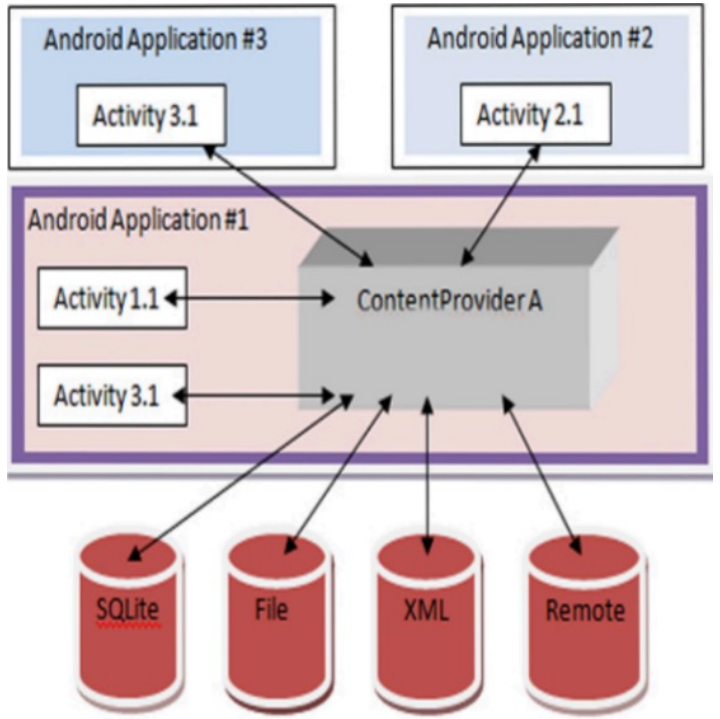
5. Content Provider

- Content Provider không được truy cập trực tiếp, mà gián tiếp thông qua Content Resolver
- Content Provider không được tạo ra cho đến khi 1 Content Resolver truy cập đến nó
- Content Resolver quản lý và hỗ trợ Content Provider, cho phép sử dụng Content Provider từ các ứng dụng khác nhau.

➔ Khi ứng dụng cần truy vấn dữ liệu, tạo Content Resolver, đối tượng này sẽ gọi đến Content Provider tương ứng để truy xuất dữ liệu

5. Content Provider

- Bất kỳ một URI nào cũng đều bắt đầu với content:// biểu diễn một tài nguyên cung cấp bởi Content Provider.
- Cú pháp: content://<authority>/<path>[/<id>]
  - **content**: dữ liệu này được quản lý bởi Content Provider
  - **authority**: id của Content Provider
  - **path**: loại dữ liệu truy cập
  - **id**: record cụ thể được truy cập
- Ví dụ:
  - content://contacts/people/5
  - content://com.example.app.provider/table3/1



5. Content Provider

- Tạo mới content provider

```
// Manifest, application tag
<provider
android:authorities="com.example.hellostorage.
DemoContentProvider"
   android:name=".DemoContentProvider"
   android:enabled="true"
   android:exported="true" />
```

```
public class DemoContentProvider extends ContentProvider {
    1 usage
    private static final String AUTHORITY
            = DemoContentProvider.class.getCanonicalName();
    2 usages
    private static final UriMatcher sURIMatcher =
            new UriMatcher(UriMatcher.NO_MATCH);
    static {
        sURIMatcher.addURI(AUTHORITY, path: "demo", code: 345);
    }

    @Override
    public boolean onCreate() { return false; }

    @Nullable
    @Override
    public Cursor query(@NonNull Uri uri, @Nullable String[] strings, @Nullable String s, @Nullable String[] strings1, @Nullable String s1) {
        int uriType = sURIMatcher.match(uri);
        switch (uriType)
        {
            case 345:
                Realm r = HelloApplication.getRealmInstance();
                RealmResults<SinhVienModel> svs = r.where(SinhVienModel.class).findAll();
                MatrixCursor cs = new MatrixCursor(new String[]{"id", "name"}, initialCapacity: 1);
                for (SinhVienModel sv : svs) {
                    cs.addRow(new Object[]{sv.getId(), sv.getName()});
                }
                return cs;
        }
        return new MatrixCursor(new String[]{"id", "name"}, initialCapacity: 0);
    }

    @Nullable
    @Override
    public String getType(@NonNull Uri uri) { return null; }

    @Nullable
    @Override
    public Uri insert(@NonNull Uri uri, @Nullable ContentValues contentValues) { return null; }

    @Override
    public int delete(@NonNull Uri uri, @Nullable String s, @Nullable String[] strings) {...}

    @Override
    public int update(@NonNull Uri uri, @Nullable ContentValues contentValues, @Nullable String s, @Nullable String[] strings) {...}
}
```

5. Content Provider

- Gọi dữ liệu với Content Resolver

```
Cursor cursor = getContentResolver().query(

Uri.parse("content://com.example.hellostorage.DemoContentProvider/demo"),
      null, null, null, null
);
while (cursor.moveToNext()) {
   Log.d(getClass().getCanonicalName(), "Name: " + cursor.getString(1));
}
```

- Nếu API level >= 30

```
// Manifest file
<uses-permission
android:name="android.permission.QUERY_ALL_PACKAGES"
   tools:ignore="QueryAllPackagesPermission"/>
```

LẬP TRÌNH ANDROID CĂN BẢN

Kết thúc ☺️

Ths. Trần Xuân Thanh Phúc | Trường Đại học Công Nghiệp Thực Phẩm