

目录

目录

一、存储空间概览

- 1.1 存储空间的区分
- 1.2 存储目录
- 1.3 Internal目录的获取
- 1.4 External目录的获取
 - 1.4.1 获取授权
 - 1.4.2 Environment APIs
 - 1.4.3 检查外置存储器的可用性
 - 1.4.4 External目录的获取
- 1.5 注意事项

二、键值对

三、SharedPreferences

- 3.1 介绍
- 3.2 获取SharedPreferences
- 3.3 读取SharedPreferences
- 3.4 写SharedPreferences
- 3.5 SharedPreferences的原理
- 3.6 注意事项

四、文件File

- 4.1 流
- 4.2 API
- 4.3 文件操作
- 4.4 文件IO读写操作示例
- 4.7 ? 拓展: OkIO

五、数据库

- 5.1 使用场景
- 5.2 数据库的设计
- 5.3 SQL
- 5.4 使用示例: Todo List App
 - 5.4.1 定义Contract类
 - 5.4.2 继承SQLiteOpenHelper
 - 5.4.3 Insert
 - 5.4.4 Query
 - 5.4.5 Delete
 - 5.4.6 Update
 - 5.4.7 ? Debug
 - 5.4.8 注意事项
- 5.5 ? Room Library

六、Content Provider

- 6.1 定义
- 6.2 Content Provider架构
- 6.3 优点
- 6.4 URI
- 6.5 URI使用示例
 - 6.5.1 查询: 获取联系人数据
 - 6.5.2 查询: 获取系统相册中的视频文件

一、存储空间概览

1.1 存储空间的区分

1. **Internal Storage**: 是系统分配给应用的专属内部存储空间

1. APP专有的
2. 用户不可以直接读取(root用户除外)
3. 应用卸载时自动清空
4. 有且仅有一个

2. **External Storage**: 是系统外部存储空间, 如 SD卡

1. 所有用户均可访问
2. 不保证可用性(可挂载/物理移除)
3. 可以卸载后仍保留
4. 可以有多个

1.2 存储目录

1. **Internal Storage**: /

1. APP专用:

1. data/data/{your.package.name}/ files、cache、db...

2. **External Storage**:

1. APP专用:

1. /storage/emulated/0/Android/data/{your.package.name}/ files、cache

2. 公共文件夹: ./

1. \ --- Standard: DCIM、Download、Movies
2. \ --- Others

	Internal Private	External Private	External Public
本应用可访问	Yes	Yes (4.3 以前需要授权)	有授权时 Yes
其他应用可访问	No	有授权时 Yes 7.0后只用FileProvider访问	有授权时 Yes
用户可访问	No (除非root)	Yes	Yes
可用性保证	Yes	No	No
卸载后自动清除	Yes	Yes	No

1.3 Internal目录的获取

1. file目录: `context.getFilesDir()`
2. cache目录: `context.getCacheDir()`
3. 自定义目录: `context.getDir(name, mode_private)`

1.4 External目录的获取

1.4.1 获取授权

AndroidManifest.xml中声明权限

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

请求授权 (Android 6.0及以上)

```
private final static int CODE_REQUEST_PERMISSION = 1;
if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_CALENDAR)
    != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this,
        new String[] {Manifest.permission.READ_EXTERNAL_STORAGE},
        CODE_REQUEST_PERMISSION);
}
```

在 Activity 的 onRequestPermissionsResult 方法中获取授权结果

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (requestCode == CODE_REQUEST_PERMISSION) {
        if (grantResults.length > 0 &&
            grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // 用户已授权
        } else {
            // 用户拒绝了授权
        }
    }
}
```

1.4.2 Environment APIs

提供了访问环境变量的方法

```
public class Environment extends Object{};
android.os.Environment;
```

1.4.3 检查外置存储器的可用性

通过Environment.getExternalStorageState();调用获取当前外部存储的状态，并以此判断外部存储是否可用

```

/* Checks if external storage is available for read
and write */
public boolean isExternalStorageWritable() {
    String state =
Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least
read */
public boolean isExternalStorageReadable() {
    String state =
Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||

Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}

```

1.4.4 External目录的获取

1. 应用私有目录：

1. **file**目录： `context.getExternalFilesDir(String type)`
2. **cache**目录： `context.getExternalCacheDir()`

2. 公共目录：

1. 标准目录： `Environment.getExternalStoragePublicDirectory(String type)`
2. 根目录： `Environment.getExternalStorageDirectory()`

3. 标准目录：

1. **DIRECTORY_ALARMS**
2. **DIRECTORY_DCIM**
3. **DIRECTORY_DOCUMENTS**
4. **DIRECTORY_DOWNLOADS**
5. **DIRECTORY_MOVIES**

1.5 注意事项

1. 如果用户卸载应用，系统会移除保存在应用专属存储空间中的文件
2. 由于这一行为，不应使用此存储空间保存用户希望独例于应用而保留的任何内容
 1. 例如，如果应用允许用户拍摄照片，用户会希望即使卸载应用后仍可访问这些照片
 2. 因此，应改为使用共享存储空间将此类文件保存到适当的媒体集合中。

更多信息可参考： <https://developer.android.com/guide/topics/data?hl=zh-cn>

二、键值对

在发校服时，如何统计班级同学的身高？

name : zhangsan
height : 170
weight : 120

在发校服时，如何统计班级同学的身高？



三、SharedPreferences

3.1 介绍

1. SharedPreferences 就是 Android 提供的数据库持久化的一个方式，适合单进程，小批量的数据存储和访问。基于 XML 进行实现，本质上还是文件的读写，API 相较 File 更简单。
2. 以“键-值”对的方式保存数据的xml文件，其文件保存在 `/data/data/[packageName]/shared_prefs` 目录下

3.2 获取SharedPreferences

- ❑ `context.getSharedPreferences(name, Context.MODE_PRIVATE);`
- ❑ `getActivity().getPreferences(Context.MODE_PRIVATE);`
- ❑ Mode 只能填 `MODE_PRIVATE`，以下都废弃
 - ❑ `MODE_WORLD_READABLE`
 - ❑ `MODE_WORLD_WRITEABLE`
 - ❑ `MODE_MULTI_PROCESS`

3.3 读取SharedPreferences

```
1. String getString(String key, String defValue);
2. Set<String> getStringSet(String key, Set<String>
    defValues);
3. int getInt(String key, int defValue);
4. long getLong(String key, long defValue);
5. float getFloat(String key, float defValue);
6. boolean getBoolean(String key, boolean defValue);
```

3.4 写SharedPreferences

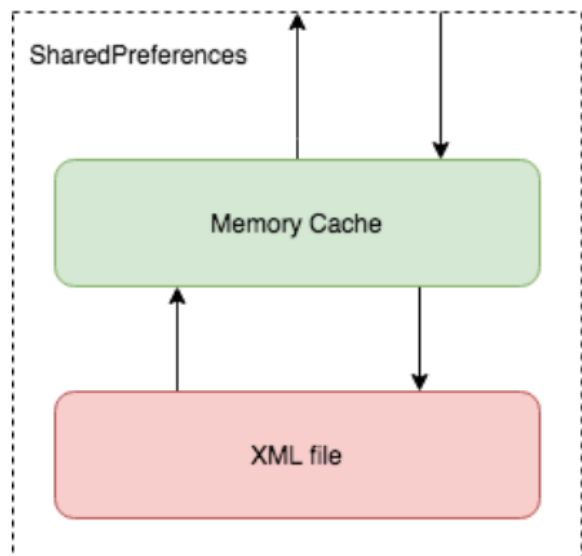
通过Editor类来提交修改

```
SharedPreferences sp = SpDemoActivity.this.getSharedPreferences(SP_DEMO,
    MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit();
editor.putString(key, value);
editor.commit();
// 或者调用 apply 方法
// editor.apply();
```

3.5 SharedPreferences的原理

SharedPreferences 的原理

- ❑ 一次性读取到内存
- ❑ 提供同步和异步两种写回文件的方式



写 SharedPreferences

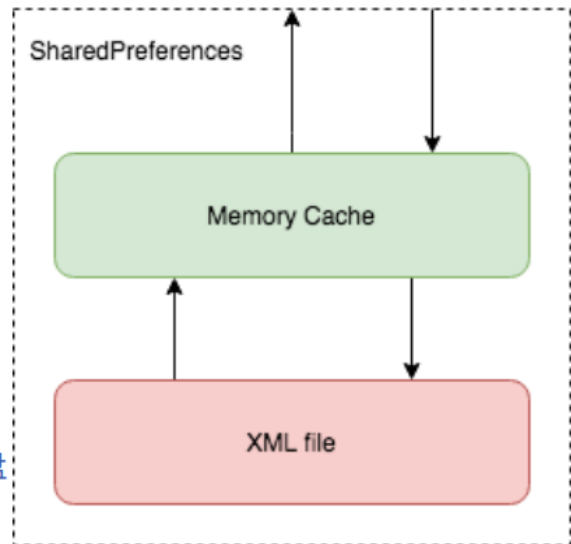
❑ commit 和 apply 的区别

❑ commit()

- ❑ 同步写入内存和磁盘
- ❑ 有返回值
- ❑ 同时调用时，取最后一次调用

❑ apply()

- ❑ 同步写入内存，异步保存至磁盘
- ❑ 无返回值
- ❑ 同时调用时，取最后一次调用



ByteDance 字节跳动

3.6 注意事项

1. SharedPreferences 适合场景：小数据
2. SharedPreferences 每次写入均为全量写入
3. 禁止大数据存储在 SharedPreferences 中，导致 ANR

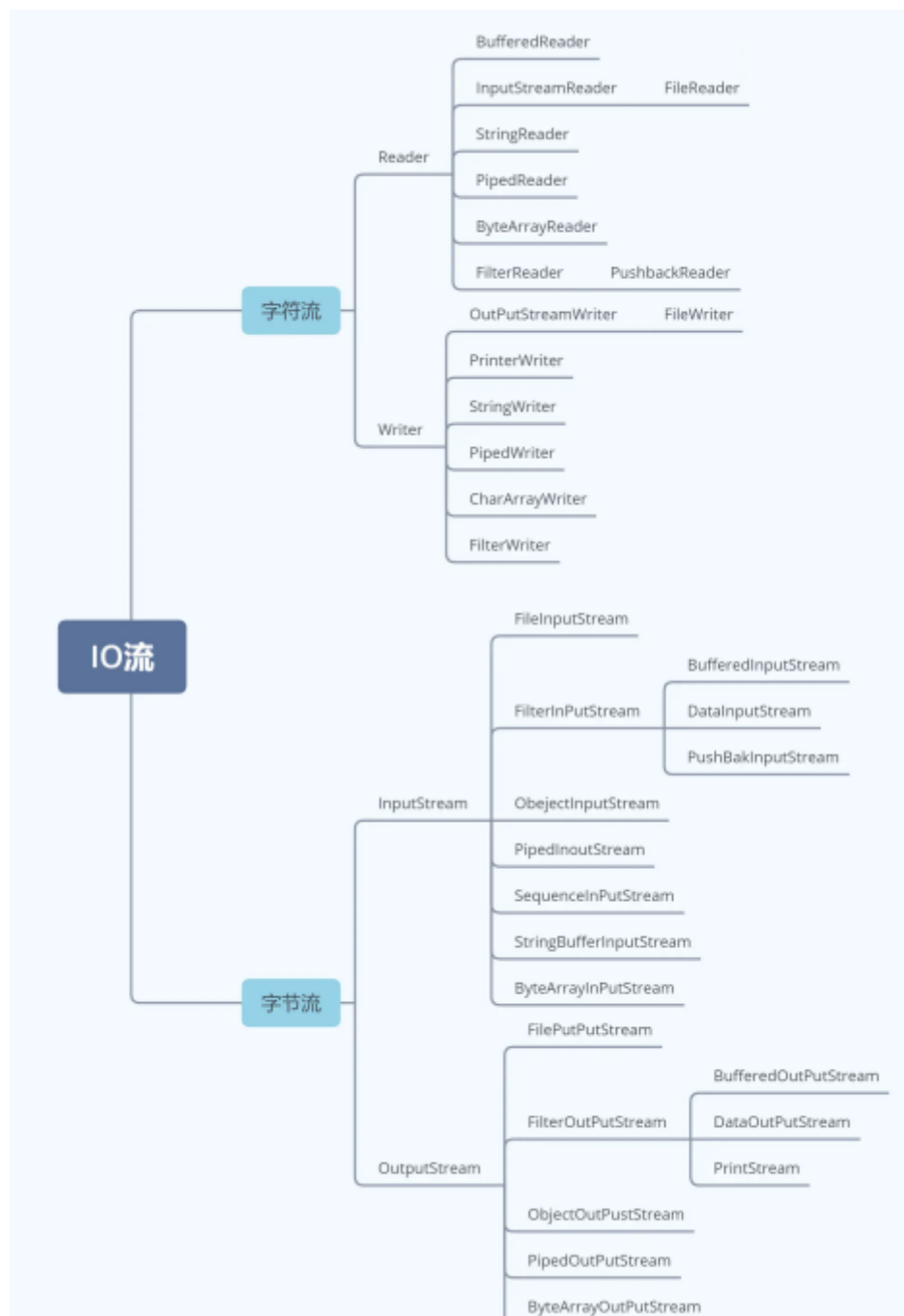
官方推荐的DataStore: <https://developer.android.com/topic/libraries/architecture/datastore>

四、文件File

4.1 流

都是相对调用者而言的

1. 按流向分为：
 1. 输入流
 2. 输出流
2. 按传输单位分为：
 1. 字节流: **InputStream** 和 **OutputStream** 基类
 2. 字符流: **Reader** 和 **Writer** 基类



4.2 API



File APIs

File

added in API level 1

```
public class File
    extends Object implements Serializable, Comparable<File>
    java.lang.Object
        ↳ java.io.File
```

Public constructors

`File(String pathname)`

Creates a new `File` instance by converting the given pathname string into an abstract pathname.

`File(String parent, String child)`

Creates a new `File` instance from a parent pathname string and a child pathname string.

`File(File parent, String child)`

Creates a new `File` instance from a parent abstract pathname and a child pathname string.

`File(URI uri)`

Creates a new `File` instance by converting the given `file: URI` into an abstract pathname.

4.3 文件操作

- ❑ `exists()`
- ❑ `createNewFile()`
- ❑ `mkdir()` vs `mkdirs()` 创建单个目录/多级目录
- ❑ `list()` vs `listFiles()` 打印当前文件夹
- ❑ `getFreeSpace()` & `getTotalSpace()`
- ❑ ...

4.4 文件IO读写操作示例

1. 创建 `File` 对象，通过构造函数：

1. `new File()`

2. 创建输入输出流对象：

1. `new FileReader()`

2. `new FileWriter()`

3. 读取 or 写入

1. `read` 方法

2. `write` 方法

4. 关闭资源

1. 有借有还，再借不难

```

@Test
public void FileReaderFileWriteTest(){
    FileReader fr = null;
    FileWriter fw = null;
    try {
        //1. 创建File对象
        File srcFile = new File("源文件地址");
        File destFile = new File("目标文件地址");
        //2. 创建输入流输出流的对象
        fr = new FileReader(srcFile);
        fw = new FileWriter(destFile);
        //3. 数据的读入和写出操作
        char[] cbuf = new char[5];
        int len; //记录每次读入到cbuf数组中字符的个数
        while((len=fr.read(cbuf))!=-1){
            fw.write(cbuf,0,len);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        //4. 关闭资源
        try {
            if(fr!=null){
                fr.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                if(fw!=null){
                    fw.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

4.7 ? 拓展：OkIO

1. 是在JavaIO基础上再次进行封装的IO框架
2. <https://square.github.io/okio/>

五、数据库

5.1 使用场景

1. 重复的数据
2. 结构化的数据
3. 关系型数据

5.2 数据库的设计



数据库的设计

□ 基本概念

- 表、主键、外键、索引
- SQL语法: <https://www.w3schools.com/sql/default.asp>

	主键	外键	索引
定义:	唯一标识一条记录,不能有重复的,不允许为空	表的外键是另一表的主键,外键可以有重复的,可以是空值	该字段没有重复值,但可以有一个空值
作用:	用来保证数据完整性	用来和其他表建立联系用的	是提高查询排序的速度
个数:	主键只能有一个	一个表可以有多个外键	一个表可以有多个唯一索引

5.3 SQL

使用 CREATE TABLE - 创建新表

```
CREATE TABLE database_name.table_name(  
    column1 datatype PRIMARY KEY(one or more columns),  
    column2 datatype,  
    column3 datatype,  
    .....  
    columnN datatype,  
);  
  
# Example  
CREATE TABLE todo.todolist(  
    id INTEGER PRIMARY KEY NOT NULL,  
    priority INTEGER,  
    content TEXT  
);
```

使用 INSERT INTO - 创建新表

```
INSERT INTO table_name (column1,column2,column3,...)  
VALUES (value1,value2,value3,...);  
  
# Example  
INSERT INTO todolist (id,priority,content)  
VALUES (1,0,"吃饭");
```

使用 **DELETE** - 删除表中已有的记录

```
DELETE FROM table_name
WHERE [condition];

# Example
DELETE FROM todolist
WHERE id = 8;
```

使用 **UPDATE** - 修改表中已有的记录

```
UPDATE table_name
SET column1 = value1, column2 = value2..., columnN = valueN
WHERE [condition];

# Example
UPDATE todolist
SET content = "吃面"
WHERE id = 1;
```

使用 **SELECT** - 查询表中已有的记录

```
SELECT column1, column2, columnN FROM table_name;

# Example
# 选取所有字段
SELECT * FROM todolist;
# 仅选取 id、content 字段的内容
SELECT id, content FROM todolist;
# 结合 WHERE 使用
SELECT content FROM todolist WHERE id = 1;
```

5.4 使用示例：Todo List App

5.4.1 定义Contract类

定义表结构、SQL语句

```

public class TodoDbHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "todo.db";
    private static final int DB_VERSION = 1;

    public TodoDbHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // 创建
        db.execSQL(TodoContract.SQL_CREATE_TODO_TABLE);
    }

    @Override

```

5.4.2 继承SQLiteOpenHelper

执行Create 和 Delete 操作

```

public class TodoDbHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "todo.db";
    private static final int DB_VERSION = 1;

    public TodoDbHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // 创建
        db.execSQL(TodoContract.SQL_CREATE_TODO_TABLE);
    }

    @Override

```

5.4.3 Insert

通过ContentValues进行插入操作

```

private boolean saveNote2Database(String content) {
    // 插入数据库
    ContentValues contentValues = new ContentValues();
    contentValues.put(TodoNote.COLUMN_TODO_CONTENT, content);
    database.insert(TodoNote.TABLE_NAME, null, contentValues);
    return true;
}

```

5.4.4 Query

调用query 方法，返回 Cursor ，对应查询结果集合

```
List<Note> result = new LinkedList<>();
// 查询数据库
String[] projection = {
    TodoContract.TODO_NOTE._ID,
    TodoContract.TODO_NOTE.COLUMN_TODO_CONTENT,
    TodoContract.TODO_NOTE.COLUMN_TODO_STATE
};
Cursor cursor =
    database.query(TodoContract.TODO_NOTE.TABLE_NAME, projection, null, null, null, null,
        null);
if (cursor != null) {
    while (cursor.moveToNext()) {
        int id = cursor.getInt(cursor.getColumnIndex(TodoContract.TODO_NOTE._ID));
        String content = cursor.getString(
            cursor.getColumnIndexOrThrow(TodoContract.TODO_NOTE.COLUMN_TODO_CONTENT));
        int state =
            cursor.getInt(cursor.getColumnIndex(TodoContract.TODO_NOTE.COLUMN_TODO_STATE));
    }
}
```

5.4.5 Delete

删除数据库中对应该 id 的数据

```
// 删除数据库中的对应数据
database.delete(TodoContract.TODO_NOTE.TABLE_NAME, TodoContract.TODO_NOTE._ID + "= ?",
    new String[] { String.valueOf(note.id) });
```

5.4.6 Update

```
// 更新数据库中的对应数据
ContentValues contentValues = new ContentValues();
int state = (note.getState() == State.DONE ? State.DONE.intValue : State.TODO.intValue);
contentValues.put(TodoContract.TODO_NOTE.COLUMN_TODO_STATE, state);
database.update(TodoContract.TODO_NOTE.TABLE_NAME, contentValues,
    TodoContract.TODO_NOTE._ID + "= ?", new String[] { String.valueOf(note.id) });
```

5.4.7 ? Debug

adb + sqlite3: <http://www.sqlite.org/cli.html>

5.4.8 注意事项

在合适的时机 close 数据库连接：

- 过早：getWritableDatabase() 和 getReadableDatabase() 是耗时操作，close后再创建连接成本大；
- 过晚：没有及时释放，内存泄漏；

- ❑ IO 操作不能放在 UI 绘制线程
- ❑ SQLiteDatabase 建议设置成单例。
- ❑ getWritableDatabase 和 getReadableDatabase 获取的对象没有本质区别，唯一的区别：后者在 DB 不可写的时候不会抛异常。
- ❑ 多次频繁操作，可以通过事务完成，减少 IO 次数。

5.5 ? Room Library

<https://developer.android.com/jetpack/androidx/releases/room>

- ❑ SQLite APIs 的痛点：
 - SQL 语句无编译时校验，容易出错，调试成本大。
 - 表结构变化后无需手动更新，并处理升级逻辑。
 - 使用大量模板代码从 SQL 查询向 JavaBeans 转换。
- ❑ Room: <https://developer.android.com/jetpack/androidx/releases/room>
 - JetPack 中的库
 - 对数据库的使用做了一层抽象
 - 通过 APT 减少模版代码

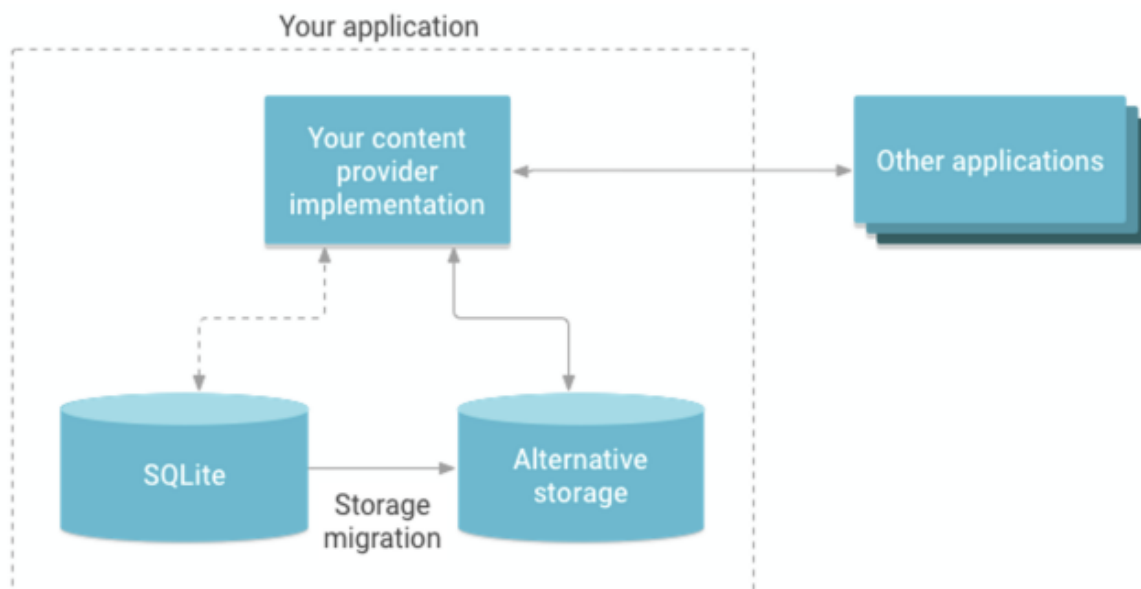
六、Content Provider

6.1 定义

1. 当我们需要在应用间共享数据时，**ContentProvider** 就是一个非常值得使用的组件
2. 四大组件之一，**ContentProvider** 是一种 **Android** 数据共享机制，无论其内部数据以什么样的方式组织，对外都是提供统一的接口
3. 通过 **ContentProvider** 可以获取系统的媒体、联系人、日程等数据

<https://developer.android.com/reference/android/content/ContentProvider>

6.2 Content Provider 架构



6.3 优点

1. 跨应用分享数据
 1. 系统的 **providers** 有联系人等
 2. 是对数据层的良好抽象
 3. 支持精细的权限控制

6.4 URI

URI: Uniform Resource Identifier, 唯一标识ContentProvider的数据

格式为:

[schema://authority/path/id](#)

schema : 固定为 content

authority : 标识 ContentProvider 的唯一字符串, 对应于注册时指定的 android:authority 属性

path : 标识 authority 数据的某些子集

id : 标识 path 子集中的某个记录 (不指定是标识全部记录)

系统预置了一些 ContentProvider, 例如通讯录、媒体资源等, 一些常用的系统 ContentProvider 的 Authority:

Authority	描述
com.android.contacts	通讯录
media	媒体
com.android.calendar	日历
user_dictionary	用户词典

增

删

改

查

- `query(Uri, String[], Bundle, CancellationSignal)` which returns data to the caller
- `insert(Uri, ContentValues)` which inserts new data into the content provider
- `update(Uri, ContentValues, Bundle)` which updates existing data in the content provider
- `delete(Uri, Bundle)` which deletes data from the content provider

6.5 URI使用示例

6.5.1 查询：获取联系人数据

1、AndroidManifest 权限声明

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

2、动态请求权限

```
if (ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_CONTACTS)
    != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new String[] {
        Manifest.permission.READ_CONTACTS
    }, CODE_REQUEST_PERMISSION);
}
```

4、遍历 cursor，获取对应字段的值

```
if (cursor == null) {
    return;
}
while (cursor.moveToNext()) {
    // 取得联系人名字（显示出来的名字），实际内容在 ContactsContract.Contacts 中
    int nameIndex = cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME);
    String name = cursor.getString(nameIndex);

    // 取得联系人 ID
    String contactId =
        cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID));

    // 根据联系人 ID 查询对应的电话号码
    Cursor phoneNumbers = cr.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
        ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " + contactId, null, null);

    // 取得电话号码（可能存在多个号码）
    while (phoneNumbers != null && phoneNumbers.moveToNext()) {
        String strPhoneNumber = phoneNumbers.getString(
```

6.5.2 查询：获取系统相册中的视频文件

1、AndroidManifest 中声明权限

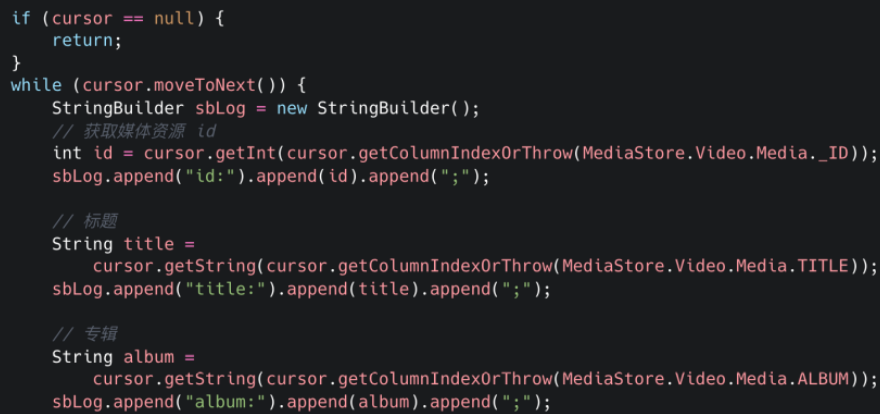
```
<uses-permission  
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

2、动态请求权限（略）

3、通过 ContentResolver 查询

```
// 获取 ContentResolver 对象  
ContentResolver contentResolver = this.getContentResolver();  
Cursor cursor =  
    contentResolver.query(MediaStore.Video.Media.EXTERNAL_CONTENT_URI,  
        null, null, null,  
        null);
```

4、遍历 cursor，获取对应字段的值



```
if (cursor == null) {  
    return;  
}  
while (cursor.moveToNext()) {  
    StringBuilder sbLog = new StringBuilder();  
    // 获取媒体资源 id  
    int id = cursor.getInt(cursor.getColumnIndexOrThrow(MediaStore.Video.Media._ID));  
    sbLog.append("id:").append(id).append(";");  
  
    // 标题  
    String title =  
        cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Video.Media.TITLE));  
    sbLog.append("title:").append(title).append(";");  
  
    // 专辑  
    String album =  
        cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Video.Media.ALBUM));  
    sbLog.append("album:").append(album).append(";");  
}
```