Android 存储

徐颖逸 - 字节跳动 Android 工程师

ByteDance字节跳动

目录

- 1. Storage overview
- 2. Files
- 3. SharedPreferences
- 4. Database
- 5. Content providers
- 6. Project: To-do list app

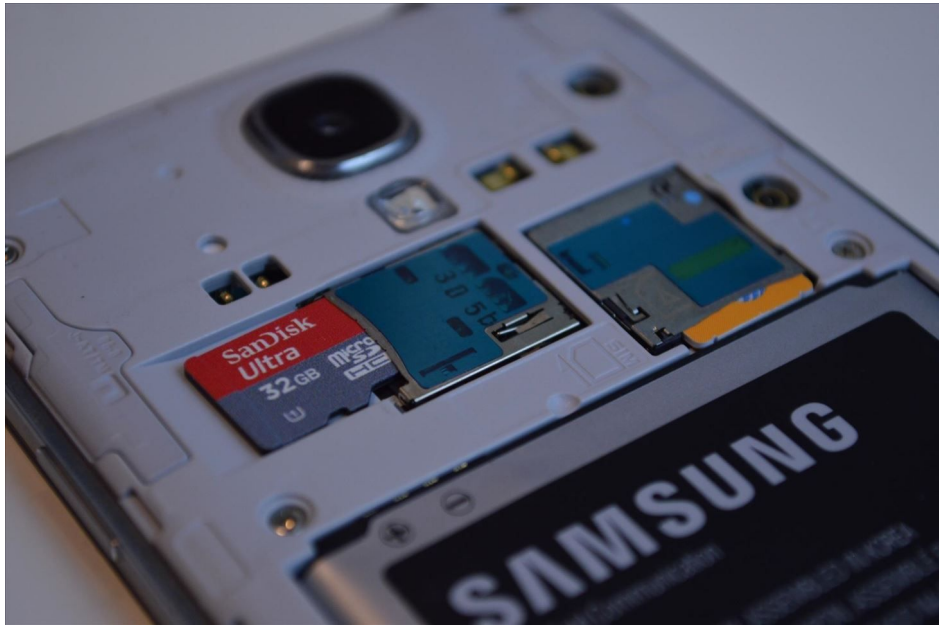
#Storage Overview

课程目标

- 系统理解 Android 设备的存储设计
- 学会在应用内保存文件、键值对、数据库等数据
- 学会在应用间分享文件和数据

存储介质

- Internal storage
  - App-private
  - 用户不可直接读取（root用户除外）
  - 应用卸载时自动清空
  - 有且仅有一个
- External storage
  - World-accessible
  - 不保证可用性（可挂载或物理移除）
  - 可以卸载后仍保留
  - 可以有多个



存储目录

Internal Storage /

\--- App-specific directory data/data/{your.package.name}/

\--- files、cache、db...

External Storage /storage/emulated/0/

\--- App-specific directory Android/data/{your.package.name}/

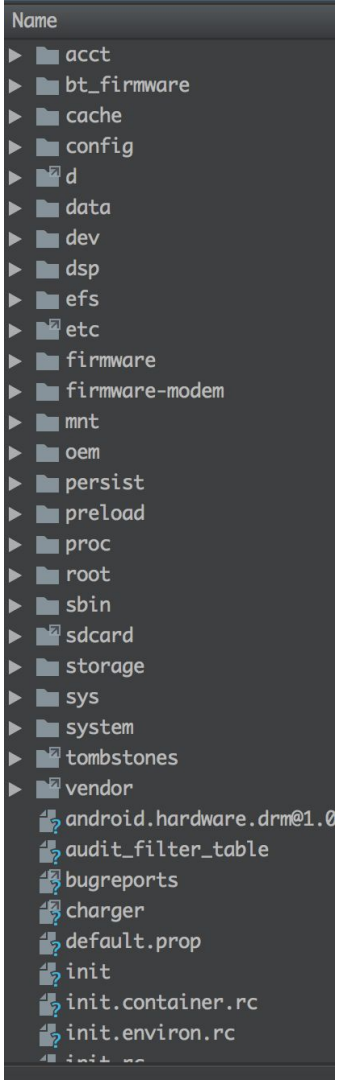
\--- files、cache、db...

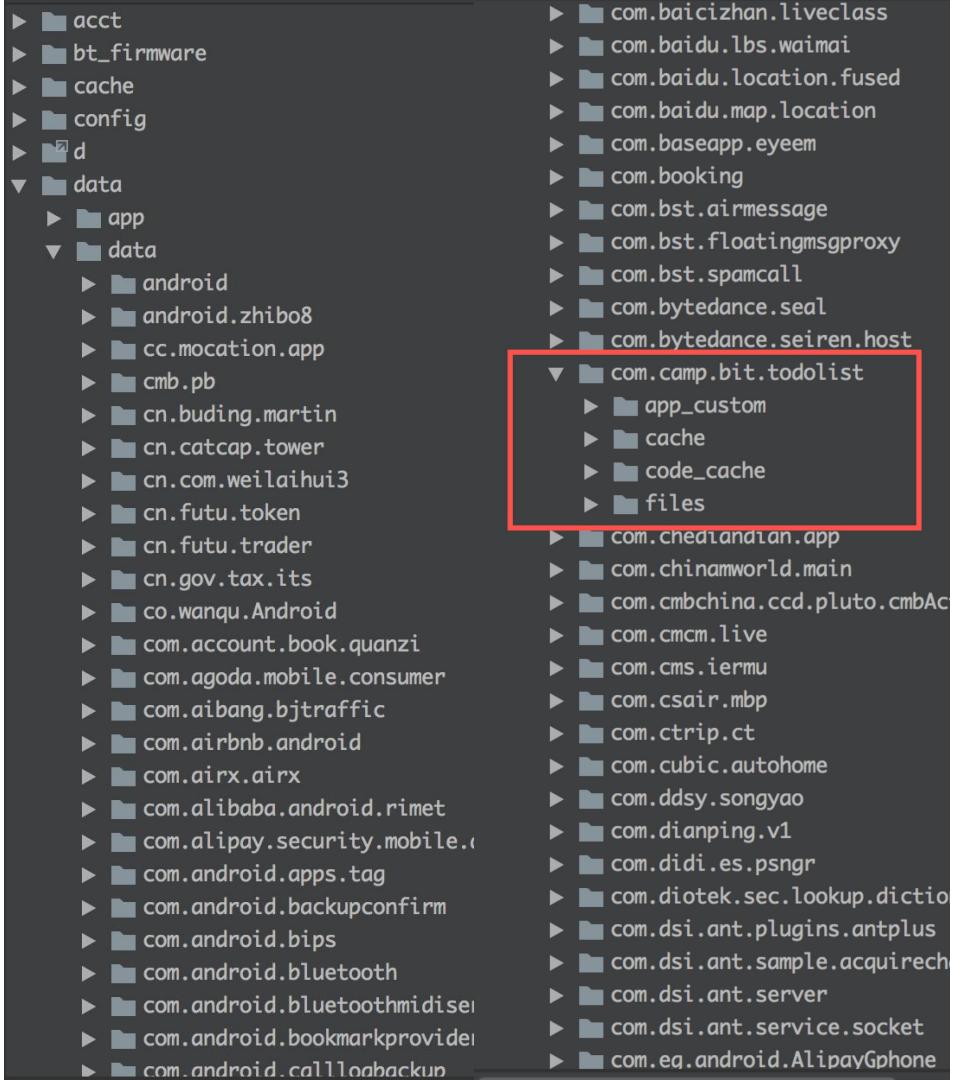
\--- Public directory ./

\--- Standard: DCIM、Download、Movies

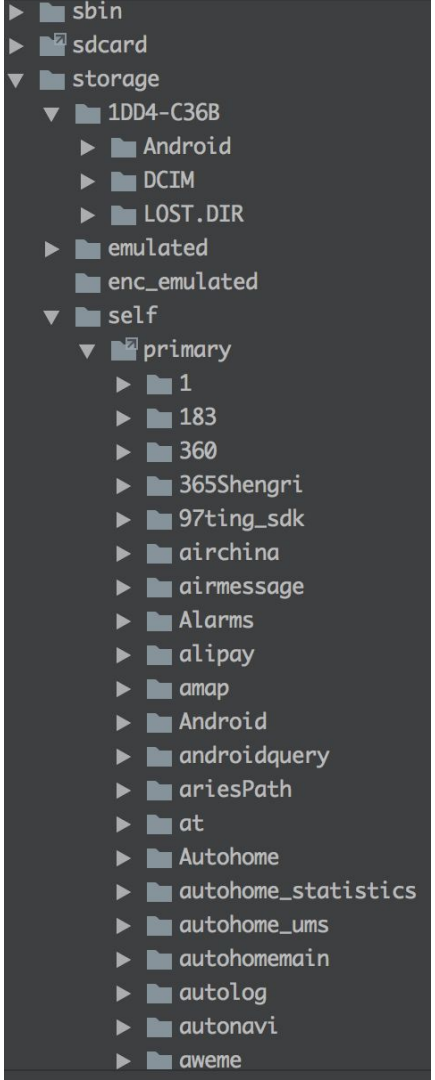
\--- Others

存储目录


Name
acct
bt_firmware
cache
config
d
data
dev
dsp
efs
etc
firmware
firmware-modem
mnt
oem
persist
preload
proc
root
sbin
sdcard
storage
sys
system
tombstones
vendor
android.hardware.drm@1.0
audit_filter_table
bugreports
charger
default.prop
init
init.container.rc
init.environ.rc


acct
bt_firmware
cache
config
d
data
app
data
android
android.zhibo8
cc.mocation.app
cmb.pb
cn.buding.martin
cn.catcap.tower
cn.com.weilaihui3
cn.futu.token
cn.futu.trader
cn.gov.tax.its
co.wanqu.Android
com.account.book.quanzi
com.agoda.mobile.consumer
com.aibang.bjtraffic
com.airbnb.android
com.airx.airx
com.alibaba.android.rimet
com.alipay.security.mobile.
com.android.apps.tag
com.android.backupconfirm
com.android.bips
com.android.bluetooth
com.android.bluetoothmidise
com.android.bookmarkprovide
com.android.calllogbackup
com.baicizhan.liveclass
com.baidu.lbs.waimai
com.baidu.location.fused
com.baidu.map.location
com.baseapp.eyeem
com.booking
com.bst.airmessage
com.bst.floatingmsgproxy
com.bst.spamcall
com.bytedance.seal
com.bytedance.seiren.host
com.camp.bit.todolist
app_custom
cache
code_cache
files
com.chinamworld.main
com.cmbchina.ccd.pluto.cmbAc
com.cmcm.live
com.cms.iermu
com.csair.mbp
com.ctrip.ct
com.cubic.autohome
com.ddsy.songyao
com.dianping.v1
com.didi.es.psngr
com.diotek.sec.lookup.dictio
com.dsi.ant.plugins.antplus
com.dsi.ant.sample.acquirech
com.dsi.ant.server
com.dsi.ant.service.socket
com.eg.android.AlipayGphone

存储目录


sbin
sdcard
storage
1DD4-C36B
Android
DCIM
LOST.DIR
emulated
enc_emulated
self
primary
1
183
360
365Shengri
97ting_sdk
airchina
airmessage
Alarms
alipay
amap
Android
androidquery
ariesPath
at
Autohome
autohome_statistics
autohome_ums
autohomemain
autolog
autonavi
aweme


sdcard
storage
1DD4-C36B
Android
data
android.zhibo8
cmb.pb
cn.buding.martin
cn.catcap.tower
cn.com.weilaihui3
cn.futu.token
cn.futu.trader
cn.gov.tax.its
co.wanqu.Android
com.agoda.mobile.consumer
com.airbnb.android
com.airx.airx
com.alibaba.android.rimet
com.android.chrome
com.android.phone
com.android.systemui
com.android.vending
com.antfortune.wealth
com.asiainno.uplive
com.autonavi.minimap
com.baidu.lbs.waimai
com.bytedance.seal
com.bytedance.seiren.host
com.camp.bit.todolist
cache
files
com.chediandian.app

存储目录的区别

	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	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

应用的安装位置

- 从 API Level 8 开始支持应用指定安装位置

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:installLocation="preferExternal">
```

Value	Description
"**internalOnly**"	The app must be installed on the internal device storage only. If this is set, the app will never be installed on the external storage. If the internal storage is full, then the system will not install the app. This is also the default behavior if you do not define **android:installLocation**.
"**auto**"	The app may be installed on the external storage, but the system will install the app on the internal storage by default. If the internal storage is full, then the system will install it on the external storage. Once installed, the user can move the app to either internal or external storage through the system settings.
"**preferExternal**"	The app prefers to be installed on the external storage (SD card). There is no guarantee that the system will honor this request. The app might be installed on internal storage if the external media is unavailable or full. Once installed, the user can move the app to either internal or external storage through the system settings.

#Files

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.

Internal 目录的获取

- file目录：context.getFilesDir()
- cache目录：context.getCacheDir()
- 自定义目录：context.getDir(name, mode)

Internal 目录的获取

- 自定义目录：context.getDir(name, mode)

Parameters	
name	**String**: Name of the directory to retrieve. This is a directory that is created as part of your application data.
mode	**int**: Operating mode. Value is either **0** or combination of MODE_PRIVATE, MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE or MODE_APPEND.

Environment APIs

Environment

added in API level 1

```
public class Environment
extends Object
```

java.lang.Object
  ↳ android.os.Environment

Provides access to environment variables.

External 目录的获取

- 应用私有目录：
  - file目录: context.getExternalFilesDir(String type)
  - cache目录：context.getExternalCacheDir()
- 公共目录：
  - 标准目录：Environment.getExternalStoragePublicDirectory(String type)
  - 根目录：Environment.getExternalStorageDirectory()

External 目录的获取

- 标准目录
  - DIRECTORY_ALARMS
  - DIRECTORY_DCIM
  - DIRECTORY_DOCUMENTS
  - DIRECTORY_DOWNLOADS
  - DIRECTORY_MOVIES
  - ...

External 目录的前置检查

- 1. 获取授权
- 2. 检查外置存储器的可用性

External 目录的授权

1. 声明权限

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

2. 动态申请权限

```
ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, REQUEST_CODE);
```

External 目录的可用性检查

```
/* 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;
}
```

文件操作

- exists()
- createNewFile()
- mkdir() vs mkdirs()
- listFiles()
- getFreeSpace() & getTotalSpace()
- ...

文件IO

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

文件IO - 字节流

InputStream

added in API level 1 ☆☆☆☆☆

```
public abstract class InputStream
extends Object implements Closeable
```

java.lang.Object
↳ java.io.InputStream

Known direct subclasses
AssetManager.AssetInputStream, BackupDataInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

Known indirect subclasses
AssetFileDescriptor.AutoCloseInputStream, Base64InputStream, BufferedInputStream, CheckedInputStream, CipherInputStream, DataInputStream, DeflaterInputStream, DigestInputStream, GZIPInputStream, InflaterInputStream, JarInputStream, LineNumberInputStream, ParcelFileDescriptor.AutoCloseInputStream, PushbackInputStream, ZipInputStream

文件IO - 字符流

Reader

added in API level 1

```
public abstract class Reader
extends Object implements Readable, Closeable
```

java.lang.Object
↳ java.io.Reader

Known direct subclasses
BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

Known indirect subclasses
FileReader, LineNumberReader, PushbackReader

文件IO - 串联

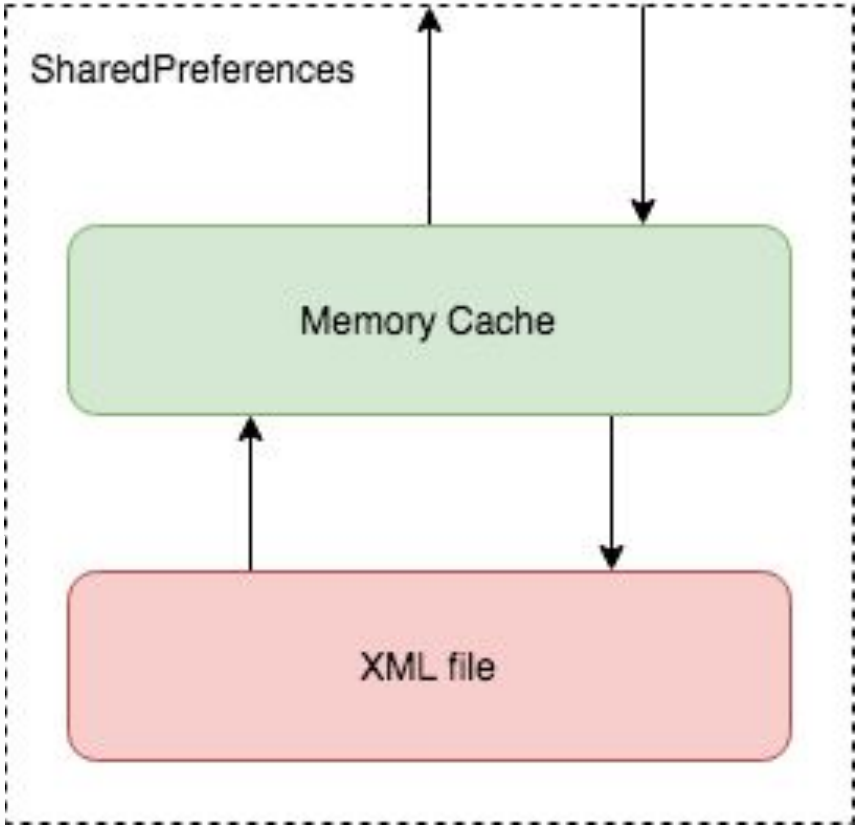
```java
public boolean copy(File from, File to) throws IOException {
    if (from == null || !from.exists() || from.isDirectory()
            || to == null || !to.exists() || to.isDirectory()) {
        return false;
    }
    FileReader fileReader = null;
    FileWriter fileWriter = null;

    BufferedReader bufferedReader = null;
    BufferedWriter bufferedWriter = null;
    try {
        fileReader = new FileReader(from);
        fileWriter = new FileWriter(to);

        bufferedReader = new BufferedReader(fileReader);
        bufferedWriter = new BufferedWriter(fileWriter);

        String line;
        while ((line = bufferedReader.readLine()) != null) {
            bufferedWriter.write( str: line + "\n");
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return false;
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    } finally {
        if (bufferedWriter != null) {
            bufferedWriter.close();
        }
        // TODO close other streams
    }
    return true;
}
```

#SharedPreferences

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

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="Name">Ider</string>
    <boolean name="Android" value="true" />
    <set name="Subsites">
        <string>code.iderzheng.com</string>
        <string>blog.iderzheng.com</string>
        <string>manual.iderzheng.com</string>
    </set>
    <int name="VersionCode" value="21" />
    <long name="VersionNumber" value="1355" />
    <float name="Version" value="5.0" />
    <null name="Null" />
</map>
```



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

读 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);
```

写 SharedPreferences

- 通过 Editor 类来提交修改事务

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);

SharedPreferences.Editor editor = sharedPref.edit();

editor.putInt(getString(R.string.saved_high_score_key), newHighScore);

editor.commit();
```

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，最后一次调用获胜
  - apply()
    - 同步写入内存，异步保存磁盘
    - 无返回值
    - 同时调用时，最后一次调用覆盖

和 Preference APIs 的区别

- Preference Library 是一个 UI 框架，用于实现一个设置页面；
- SharedPreferences 是一个存储实现，可以用于任何适用的场景；
- 通常 Preference Library 会使用 SharedPreferences 来存储页面的设置选项；

#Database

适用场景

- 重复的数据
- 结构化的数据
- 关系型数据

数据库的设计

- 基本概念
  - 表、主键、外键、索引、唯一索引
  - 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	一个表可以有多个惟一索引

数据库的设计

- 数据库冗余？
  - 数据库范式：1NF、2NF、3NF、BCNF…
  - JOIN：inner join、left join、right join...

数据库的设计

- 查询效率
  - 索引
  - Full-text-search

查询关键字	FTS DB	普通 DB
a	5645ms	4396ms
b	2678ms	3931ms
g	2749ms	3874ms
cons（consumer前半部分）	2003ms	3938ms
consumer（完整单词）	165ms	3807ms
prof（profile前半部分）	998ms	3880ms
profile（完整单词）	163ms	3804ms

数据库的设计

- 写入效率:
  - 事务处理

1、关闭事务模式（默认）

操作类型 Counts TotalTime(us) AverageTime(us)

插入 1000 31307617 31307.000000

查询 1000 944442 944.000000

更新 1000 32264043 32264.000000

删除 1000 30638604 30638.000000

2、开启事务模式

操作类型 Counts TotalTime(us) AverageTime(us)

插入 1000 23326 23.000000

查询 1000 935739 935.000000

更新 1000 39197 39.000000

删除 1000 24394 24.000000

数据库的设计

- 定义 Contract 静态类
  - 定义表名、列名等常量
  - 定义表结构和关系
  - 定义 SQL 指令

数据库的设计

```
public final class FeedReaderContract {
    // To prevent someone from accidentally instantiating the contract class,
    // make the constructor private.
    private FeedReaderContract() {}

    /* Inner class that defines the table contents */
    public static class FeedEntry implements BaseColumns {
        public static final String TABLE_NAME = "entry";
        public static final String COLUMN_NAME_TITLE = "title";
        public static final String COLUMN_NAME_SUBTITLE = "subtitle";
    }
}
```

Create

1. 在 Contract 类中定义建表和删表的 SQL 代码

```
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
    FeedEntry._ID + " INTEGER PRIMARY KEY," +
    FeedEntry.COLUMN_NAME_TITLE + " TEXT," +
    FeedEntry.COLUMN_NAME_SUBTITLE + " TEXT)";

private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE_NAME;
```

Create

2. 实现一个 SQLiteOpenHelper

```
public class FeedReaderDbHelper extends SQLiteOpenHelper {
    // If you change the database schema, you must increment the database version.
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "FeedReader.db";

    public FeedReaderDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

Create

3. 注意数据库升级逻辑

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    for (int i = oldVersion; i < newVersion; i++) {
        switch (i) {
            case 1:
                try {
                    db.execSQL("ALTER TABLE " + TABLE_NAME + " ADD " + EXTRA + " text");
                } catch (Exception e) {
                    e.printStackTrace();
                }
                break;
            default:
                break;
        }
    }
}
```

Insert

1. new 一个 DbHelper 对象

```
FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());
```

2. 获取数据库引用

```
SQLiteDatabase db = mDbHelper.getWritableDatabase();
```

Insert

3. 通过 ContentValues 写入 DB

```
// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle);

// Insert the new row, returning the primary key value of the new row
long newRowId = db.insert(FeedEntry.TABLE_NAME, null, values);
```

Insert

4. 在合适的时机 close 数据库连接：

- 过早：getWritableDatabase() 和 getReadableDatabase() 是耗时操作（需要异步执行），close 后再建连成本巨大；
- 过晚：没有及时释放，内存泄露；

```
@Override
protected void onDestroy() {
    mDbHelper.close();
    super.onDestroy();
}
```

Query

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    BaseColumns._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_SUBTITLE
    };

// Filter results WHERE "title" = 'My Title'
String selection = FeedEntry.COLUMN_NAME_TITLE + " = ?";
String[] selectionArgs = { "My Title" };

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_SUBTITLE + " DESC";

Cursor cursor = db.query(
    FeedEntry.TABLE_NAME,   // The table to query
    projection,             // The array of columns to return (pass null to get all)
    selection,              // The columns for the WHERE clause
    selectionArgs,          // The values for the WHERE clause
    null,                   // don't group the rows
    null,                   // don't filter by row groups
    sortOrder               // The sort order
    );
```

Query

2. 遍历游标 Cursor

- 默认位置为 -1

```
List itemIds = new ArrayList<>();
while(cursor.moveToNext()) {
  long itemId = cursor.getLong(
      cursor.getColumnIndexOrThrow(FeedEntry._ID));
  itemIds.add(itemId);
}
cursor.close();
```

Query

3. 从 cursor 到 JavaBeans

```
private List<Note> loadNotesFromDatabase() {
    if (database == null) {
        return Collections.emptyList();
    }
    List<Note> result = new LinkedList<>();
    Cursor cursor = null;
    try {
        cursor = database.query(TodoNote.TABLE_NAME,
                new String[]{TodoNote.COLUMN_CONTENT, TodoNote.COLUMN_DATE,
                        TodoNote.COLUMN_STATE},
                selection: null, selectionArgs: null,
                groupBy: null, having: null,
                orderBy: TodoNote.COLUMN_DATE + " DESC");

        while (cursor.moveToNext()) {
            String content = cursor.getString(cursor.getColumnIndex(TodoNote.COLUMN_CONTENT));
            long dateMs = cursor.getLong(cursor.getColumnIndex(TodoNote.COLUMN_DATE));
            int intState = cursor.getInt(cursor.getColumnIndex(TodoNote.COLUMN_STATE));

            Note note = new Note();
            note.setContent(content);
            note.setDate(new Date(dateMs));
            note.setState(State.from(intState));

            result.add(note);
        }
    } finally {
        if (cursor != null) {
            cursor.close();
        }
    }
    return result;
}
```

Delete

```
// Define 'where' part of query.
String selection = FeedEntry.COLUMN_NAME_TITLE + " LIKE ?";
// Specify arguments in placeholder order.
String[] selectionArgs = { "MyTitle" };
// Issue SQL statement.
int deletedRows = db.delete(FeedEntry.TABLE_NAME, selection, selectionArgs);
```

Update

```
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// New value for one column
String title = "MyNewTitle";
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);

// Which row to update, based on the title
String selection = FeedEntry.COLUMN_NAME_TITLE + " LIKE ?";
String[] selectionArgs = { "MyOldTitle" };

int count = db.update(
    FeedReaderDbHelper.FeedEntry.TABLE_NAME,
    values,
    selection,
    selectionArgs);
```

Debug

- adb + sqlite3: https://www.sqlite.org/cli.html

```
$ adb shell
generic_x86:/ $ su
generic_x86:/ # sqlite3 /data/data/com.camp.bit.todolist/databases/todo.db
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> .table
android_metadata  note
sqlite> .head on
sqlite> .mode column
sqlite> .schema note
CREATE TABLE note(_id INTEGER PRIMARY KEY AUTOINCREMENT, date INTEGER, state INTEGER, content TEXT);
sqlite> select * from note;
_id         date        state       content
----------  ----------  ----------  -------------
1           1234        0           FIRST CONTENT
sqlite>
```

Debug

```
sqlite> insert into note (date, state, content) values(5678, 1, "SECOND NOTE");
sqlite> select * from note;
_id         date        state       content
----------  ----------  ----------  -------------
1           1234        0           FIRST CONTENT
2           5678        1           SECOND NOTE
sqlite> delete from note where data=1234;
Error: no such column: data
sqlite> delete from note where date=1234;
sqlite> select * from note;
_id         date        state       content
----------  ----------  ----------  -----------
2           5678        1           SECOND NOTE
sqlite>
```

Room Library

- Room：
  - JetPack 中的库
  - 对数据库的使用做了一层抽象
  - 通过 APT 减少模板代码
- SQLite APIs 的痛点：
  - SQL 语句无编译时校验，容易出错，调试成本大；
  - 表结构变化后需要手动更新，并处理升级逻辑；
  - 使用大量模板代码从 SQL 查询向 JavaBeans 转换；

Room Library

```
// Use `@Fts3` only if your app has strict disk space requirements or if you
// require compatibility with an older SQLite version.
@Fts4
@Entity(tableName = "users")
public class User {
    // Specifying a primary key for an FTS-table-backed entity is optional, but
    // if you include one, it must use this type and column name.
    @PrimaryKey
    @ColumnInfo(name = "rowid")
    public int id;

    @ColumnInfo(name = "first_name")
    public String firstName;
}
```

Room Library

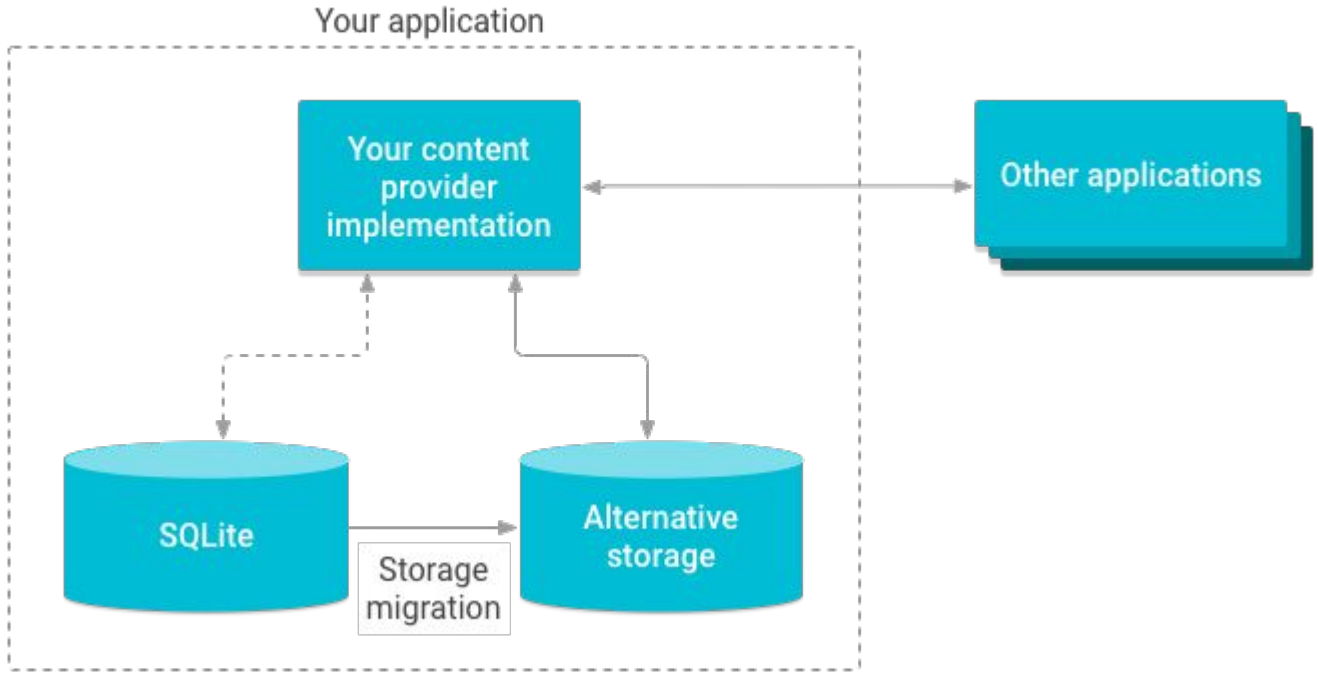
```
@Dao
public interface MyDao {
    @Query("SELECT * FROM user WHERE age BETWEEN :minAge AND :maxAge")
    public User[] loadAllUsersBetweenAges(int minAge, int maxAge);

    @Query("SELECT * FROM user WHERE first_name LIKE :search " +
           "OR last_name LIKE :search")
    public List<User> findUserWithName(String search);
}
```

#Content Providers

Why

- 跨应用分享数据
  - 系统的 providers 有 联系人、图库 等；
- 是对数据层的良好抽象
- 支持精细的权限控制

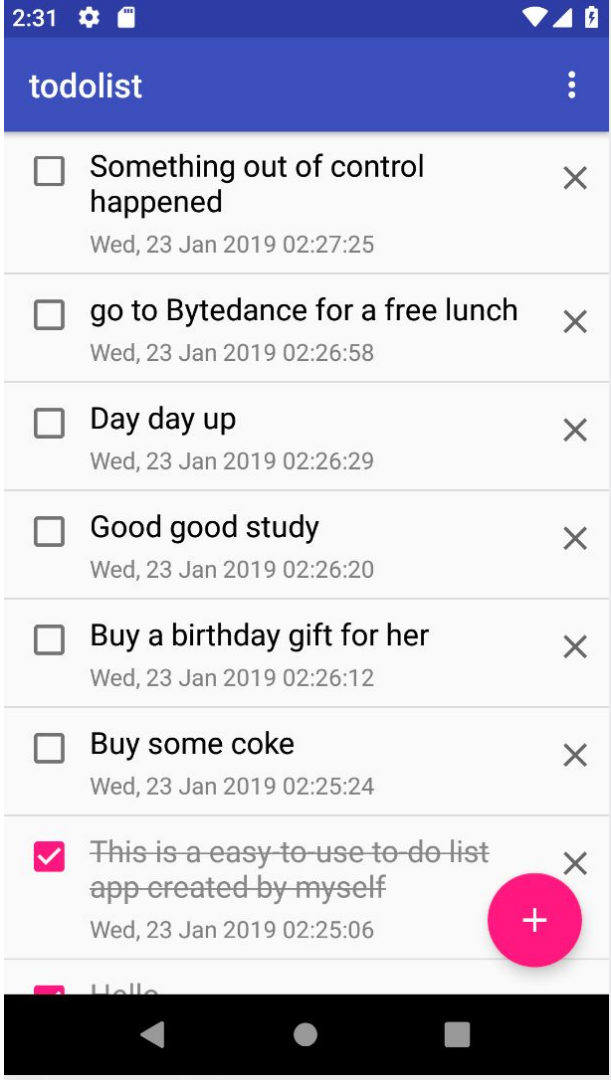
Content provider 架构



#Project

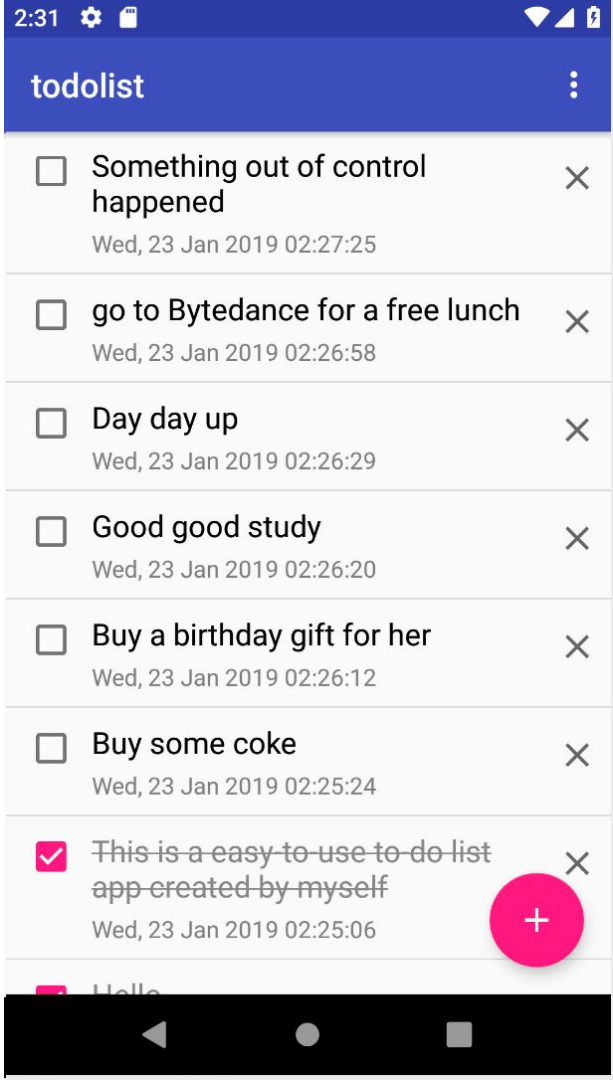
一个简单的 To-do List App

- demo



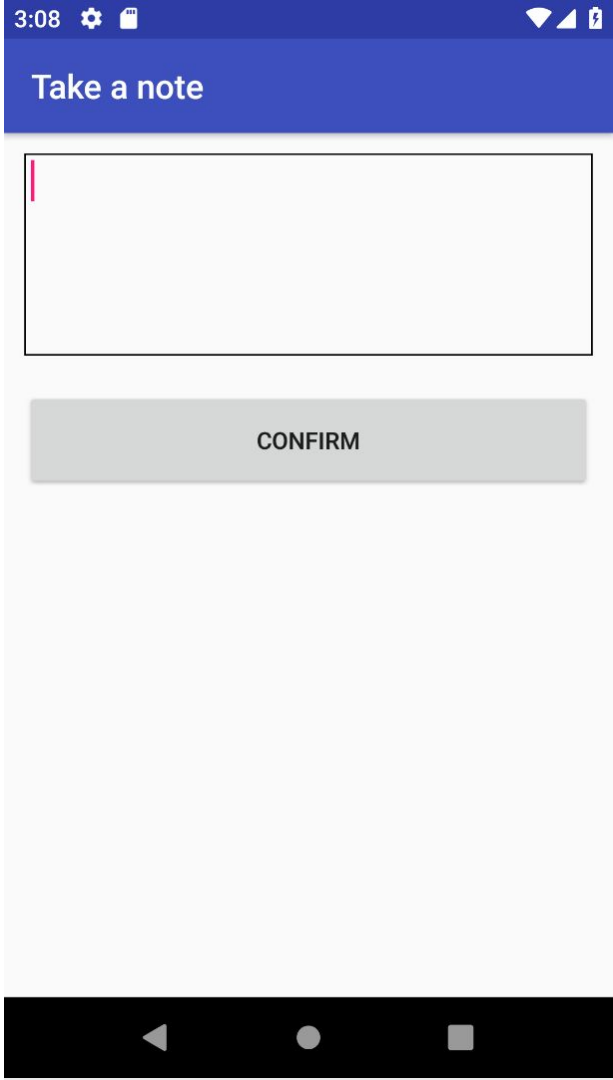
基础版要求

- 为 To-do List 的场景建立一个数据库，完成数据库表的设计和创建；
- 进入主页后，从数据库中查询出所有的数据，并以列表形式呈现出来；



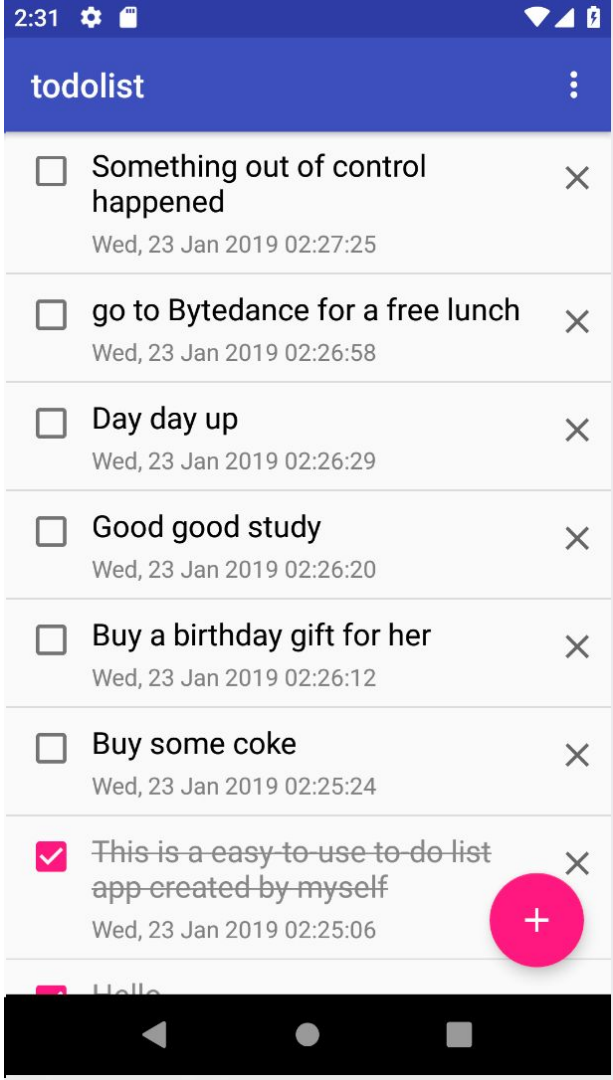
基础版要求

- 点击加号后跳转到一个新页面，输入任意内容，点击 CONFIRM 后把内容插入数据库中，返回主页并更新主页数据；



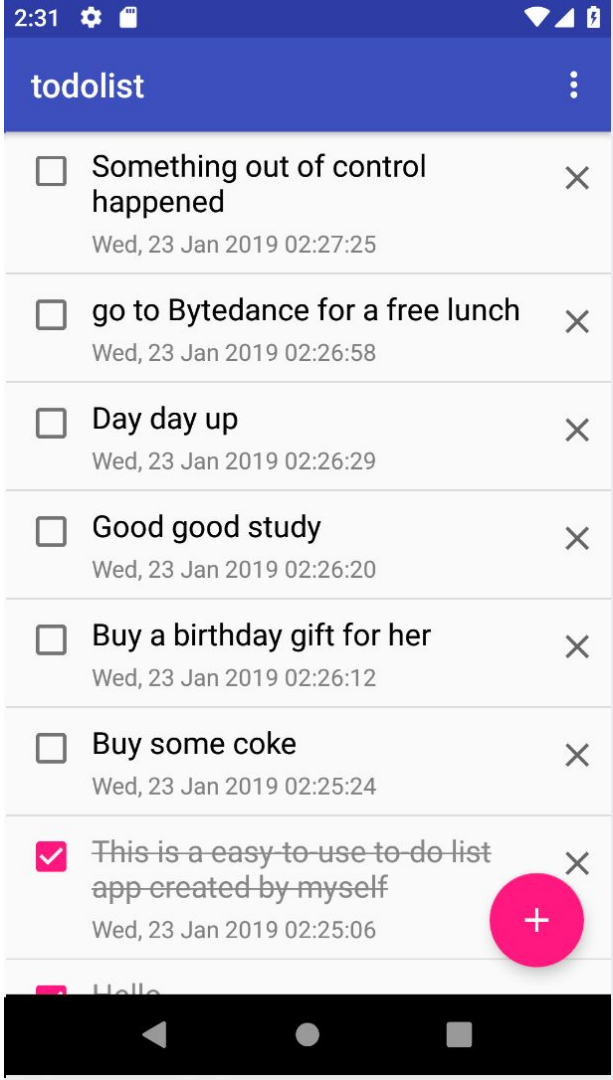
基础版要求

- 点击每个 note 前边的 checkbox 能把该条 note 置为“已完成”，并更新数据库和 UI；
- 点击每个 note 后边的 x 能把该条 note 删除，并更新数据库和 UI；



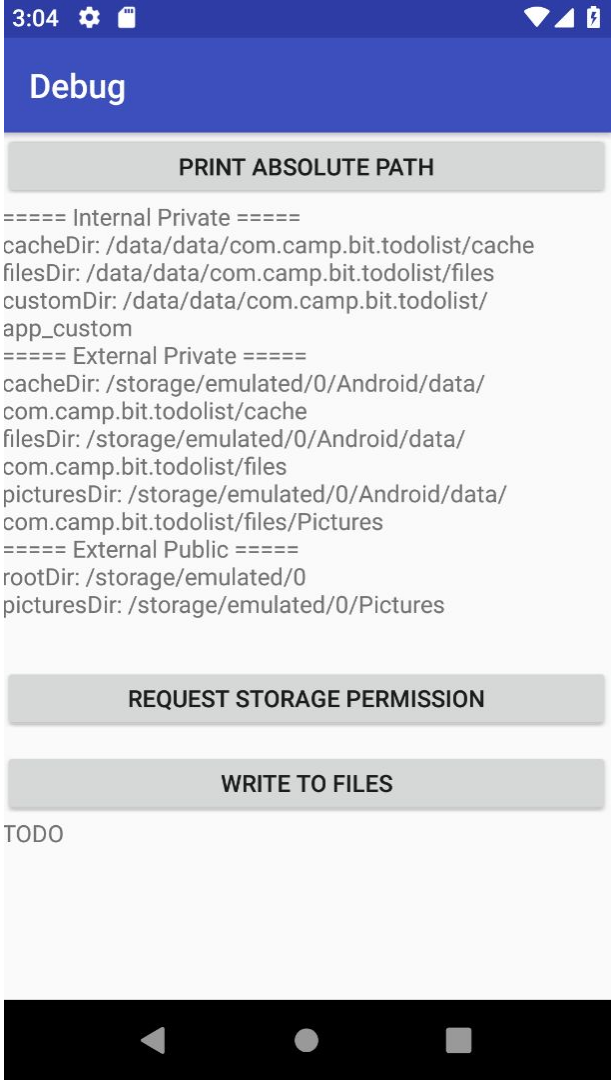
进阶版要求

- 在基础版的基础上增加“优先级”功能：
  - 升级原数据库，增加“优先级”字段；
  - 创建 note 时可以选择优先级；
  - 显示 note 时，不同的优先级背景色不一样，支持优先级越高的显示在越顶部；



文件操作

- 点击“WRITE TO FILES”时把一段文本写入某个存储目录下的文件里；
- 再把这个文件的内容读出来，并显示在下方的 TextView 里；



作业上交

- 使用 github 托管你的项目
- 发邮件
  - 发给：xuyingyi@bytedance.com
  - 标题：北理Android课设-存储
  - 内容：你的姓名、学号和项目地址

THANKS

.