

# Android移动开发

Android Mobile Application Development

## 第9讲 数据存储

吴以凡

计算机学院 一教505

yfwu@hdu.edu.cn

# 数据存储



# Android数据存储

## ■ 文件存储

- 通过文件的方式存储数据

## ■ SharedPreferences

- 以XML文件方式存放在程序的私有空间

## ■ SQLite

- 小型数据库，适合结构化数据存储

## ■ 网络存储

- 数据存储在服务器，使用的时候通过网络获取

## ■ ContentProvider内容提供器

- 实现文件在程序间的共享，可使用SQLite或文件作为存储方式

# 文件存储

## ■ Android使用Linux文件系统，主要包括

- 内部存储（内置闪存）
  - 尽量只存放少量数据
  - 程序能访问的内容在/data/data/包名/目录下
- 外部存储（SD卡、设备内置存储卡等）
  - 大文件最好放在外部文件系统，即/sdcard/目录下（/storage/emulated/0/）

## ■ Android系统文件存储有两种模式

- 存储在指定程序下手机内存中的私有数据，可在文件创建时设置访问权限
- 存储在手机SD卡中，所有程序都可以访问

## ■ 访问文件系统的方法

- 可使用java.io包中的常规Java文件I/O操作
- Context类也提供了一些辅助方法进行文件操作

# 访问内部存储的文件

- Android系统的Context类提供了能够简化读写流式文件过程的函数：

- openFileOutput

```
FileOutputStream openFileOutput (String filename, int mode)
```

- mode :

- Context.MODE\_PRIVATE: 默认的操作权限，只能被当前应用程序所读写
- Context.MODE\_APPEND: 可以添加文件的内容
- Context.MODE\_WORLD\_READABLE: 可以被其他程序所读取
- Context.MODE\_WORLD\_WRITEABLE: 可以被其他的程序所写入

- 文件保存在/data/data/<package name>/files/目录下

- openFileInput

```
FileInputStream openFileInput (String filename)
```

\* 该方法只适合存放小文件

# 示例

## ■ 访问内部存储的文件

- 通过openFileOutput方法对文件进行写入
- 通过openFileInput方法对文件进行读取
- 文件查看
  - View > Tool Windows > Device File Explorer
  - /data/data/<package\_name>/files/text.txt



# 示例

## 内部文件读取

```
String file_name="test.txt";
String str="";
FileInputStream fi_in;
try{
    fi_in=openFileInput(file_name);
    // fi_in.available()返回实际可读字节数
    byte[] buffer=new byte[fi_in.available()];
    fi_in.read(buffer);
    str=new String(buffer);
    fi_in.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

## 内部文件写入

```
//文件名称
String file_name="test.txt";
String str="Android";
FileOutputStream fi_out
try {
    fi_out=openFileOutput (file_name, MODE_PRIVATE);
    fi_out.write(str.getBytes());
    fi_out.close();
} catch(Exception e) {
    e.printStackTrace();
}
```

# 访问外部存储的文件

- SD卡使用FAT文件系统，不支持访问模式和权限控制
- 由于外部存储方式一般存放在外部设备里，所以在使用之前要先检查外围设备是否存在
  - 通过Environment.getExternalStorageState()获取SD卡状态
    - Environment.MEDIA\_MOUNTED (手机装有SD卡，且可以进行读写)
  - 通过Environment.getExternalStorageDirectory()获取SD卡路径
  - 使用FileInputStream、FileOutputStream、FileReader、FileWriter对象来读写外部设备中的文件

```
File SDCard = Environment.getExternalStorageDirectory();  
File file = new File(SDCard, "text.txt");  
FileOutputStream outStream = new FileOutputStream(file);  
outStream.write(...);  
outStream.close();
```



# 访问外部存储的文件

- 需在AndroidManifest.xml文件中加入访问SD卡的权限，否则无法正确读写文件

```
<!--在SD卡中写入数据的权限-->
```

```
<uses-permission
```

```
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

```
<!--在SD卡中读取数据的权限-->
```

```
<uses-permission
```

```
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

# 示例

## ■ 如何创建文件到SDCard

- ❑ 先判断SDCard是否处于挂载并可写入状态
- ❑ 获取SDCard路径，创建文件
- ❑ 通过FileOutputStream方法写入到SDCard
- ❑ 模拟器中开启app的存储权限
  - 长按app > App info > Permissions > Storage
- ❑ 文件查看
  - /sdcard/text.txt



# 示例

## 外部文件写入

```
String environment = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(environment)) {
    //外部设备可以进行读写操作
    File sd_path = Environment.getExternalStorageDirectory();
    File file = new File(sd_path,"test.txt");
    String str = "Android";
    FileOutputStream fos;
    try {
        //写入数据
        fos = new FileOutputStream(file);
        fos.write(str.getBytes());
        fos.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

# 示例

## 外部文件读取

```
String environment = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(environment)) {
    //外部设备可以进行读写操作
    File sd_path = Environment.getExternalStorageDirectory();
    File file = new File(sd_path,"test.txt");
    FileInputStream fis;
    try {
        //读取文件
        fis = new FileInputStream(file);
        BufferedReader buff_reader = new BufferedReader(new InputStreamReader(fis));
        str = buff_reader.readLine();
        fis.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

# SharedPreferences

- SharedPreferences是Android平台上一个轻量级的存储类
  - 通过key-value对存储数据，以XML方式保存在程序私有目录下
    - /data/data/<package name>/shared\_prefs/
  - 主要保存应用的配置信息、记录程序运行状态
  - 提供了Android平台常规的Long（长整形）、Int（整形）、String（字符串型）等数据的保存
  - 一次性读取到内存

# SharedPreferences保存数据的步骤

## ■ SharedPreferences保存数据的步骤

- ❑ 根据Context获取SharedPreferences对象
- ❑ 利用edit()方法获取Editor对象
- ❑ 通过Editor对象存储key-value键值对数据
- ❑ 通过commit()方法提交数据

# SharedPreferences保存数据的步骤

//保存数据

```
SharedPreferences myPreferences  
    = getSharedPreferences("myTable", MODE_PRIVATE);  
SharedPreferences.Editor editor = myPreferences.edit();  
editor.putString("name", tbName.getText().toString());  
editor.putBoolean("work", cbWork.isChecked());  
editor.commit();
```

```
<map>  
  <boolean name="work" value="true" />  
  <string name="name">张三</string>  
</map>
```

# SharedPreferences保存数据的步骤

//读取数据

```
SharedPreferences myPreferences = getSharedPreferences();  
String name = myPreferences.getString("name", "");  
String work = myPreferences.getBoolean("work", false);
```

//删除数据

```
SharedPreferences myPreferences = getSharedPreferences();  
Editor editor = myPreferences.edit();  
editor.remove("name"); //删除一条数据  
editor.clear(); //删除所有数据  
editor.commit(); //提交修改
```



# 访问SharedPreferences的API

## ■ SharedPreferences的API :

- getSharedPreferences : 获取SharedPreferences的对象

```
public SharedPreferences getSharedPreferences (String name, int mode)
```

- putXxxx 方法 : 通过key-value对 , 将数据保存到XML文件中

```
public int putXxxx (String key, xxxx value)
```



xxxx可为 : boolean、float、int、long、String等基本类型

- commit 方法 : 数据提交
- getXxxx 方法 : 获取保存的key-value对的值

```
public int getXxxx (String key, xxxx defaultValue)
```

# 获取SharedPreferences对象的方法

## ■ PreferenceManager

```
public static SharedPreferences getDefaultSharedPreferences (Context context)
```

- 指向一个默认的文件

## ■ Context

```
public SharedPreferences getSharedPreferences (String name, int mode)
```

文件名



## ■ Activity

```
public SharedPreferences getPreferences (int mode)
```

- 将Activity的类名作为文件名传给Context.getSharedPreferences
- 仅该Activity使用

# 示例

- 使用SharedPreferences保存用户输入信息
  - 在Activity的onCreate方法中读取上次保存的数据, 恢复各个UI控件的输入值
  - 在程序退出时执行的onStop方法中保存当前各UI控件的用户输入值



# SharedPreferences

## ■ 注意事项

- ❑ 适合场景：小数据
- ❑ 每次写入均为全量写入
- ❑ 大数据存取时导致ANR ( Application Not Responding )
- ❑ MMKV
  - I/O->内存映射、XML->Protobuf、增量更新

# SQLite

- SQLite是一个开源的嵌入式关系数据库，它在2000年由D. Richard Hipp发布，是用C语言编写的
- SQLite是一款轻型数据库，主要为嵌入式设备开发，资源占用非常低，仅需要几百K内存
- 适用于应用程序存放大量结构化数据
- 特点
  - 免费
  - 小型嵌入式引擎
  - 单一文件
  - 无需安装或管理
  - 流行：iPhone, Firefox, Skype等



# SQLite

## ■ SQLite支持的数据类型

- NULL ( 空值 )
- INTEGER ( 整数 )
- REAL ( 浮点数 )
- TEXT ( 字符串文本 )
- BOLB ( 二进制对象 )

## ■ 对varchar(n)、char(n)、decimal(p,s) , 会转换成上述5种类型

## ■ SQLite可以把各种类型的数据保存到任何字段 , 不用关心声明的数据类型 ( 列类型并不是强制的 )

# 创建和打开数据库

- SQLite中用于创建或打开数据库的方法是 `openOrCreateDatabase()`
  - 是Context的方法，调用此方法会自动检测是否存在此数据库。
  - 如果数据库已经存在执行打开动作，否则创建数据库。
  - 创建成功会返回一个SQLiteDatabase对象，否则抛出异常 `FileNotFoundException`。使用方法如下：

```
mSQLiteDatabase = this.openOrCreateDatabase("test.db",MODE_PRIVATE,null);
```

# 创建和打开数据库

## ■ 或使用Android提供的SQLiteOpenHelper类

- SQLiteOpenHelper是SQLiteDatabse的一个帮助类，用来管理数据的创建和版本更新。一般的用法是定义一个类继承SQLiteOpenHelper，并实现两个回调方法
  - onCreate(SQLiteDatabase db)：在初次生成数据库时才会被调用
  - onUpgrade(SQLiteDatabase, int oldVersion, int newVersion)在数据库的版本发生变化时会被调用，一般在软件升级时才需改变版本号，而数据库的版本是由程序员控制的
- 通过getWritableDatabase()和getReadableDatabase()方法获得SQLiteDatabase对象



# 创建表和插入记录

- 通过SQLiteDatabase对象的execSQL方法执行一条SQL语句来创建表，使用方法如下：

```
String Create_Table = "Create table table1 (_id INTEGER PRIMARY KEY,  
                                num INTEGER,data TEXT)";  
mSQLiteDatabase.execSQL(Create_Table);
```

- 两种方式实现记录的插入

- 通过SQLiteDatabase的insert方法插入
- 通过execSQL方法并指定SQL语句插入记录。具体实现如下：

```
ContentValues cv = new ContentValues();  
cv.put("num", 2);  
cv.put("data", "测试");  
mSQLiteDatabase.insert(TABLE_NAME, null, cv);
```

# 删除和更新记录

## ■ 通过SQLiteDatabase对象的delete方法删除记录

```
String delete_data = "DELETE FROM table1 WHERE _id=1";  
mSQLiteDatabase.delete(delete_data);
```

## ■ 通过SQLiteDatabase对象的update方法更新记录

```
ContentValues cv = new ContentValues();  
cv.put("num", 3);  
mSQLiteDatabase.update("table1", cv, "num"+"=" + Integer.toString(0), null);
```

# 查询记录

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

- distinct: 每一行记录是否为唯一
- table: 表名
- columns: 返回的字段名
- selection: SQL WHERE
  - "column1 = ? OR column1 = ?"
- selectionArgs: SQL WHERE参数
  - new String[] { "value1", "value2" }
- groupBy: SQL GROUP BY
- having: SQL HAVING
- orderBy: SQL ORDER BY
- limit: SQL LIMIT

# Cursor

■ Cursor是一个游标接口，在数据库中作为返回值，相当于结果集ResultSet，其常用方法如下：

- ❑ moveToFirst()：移动光标到第一行。
- ❑ moveToLast()：移动光标到最后一行。
- ❑ moveToNext()：移动光标到下一行。
- ❑ moveToPosition(int position)：移动光标到一个绝对的位置。
- ❑ moveToPrevious()：移动光标到上一行。
- ❑ getColumnCount()：返回所有列的总数。
- ❑ getColumnIndex(String columnName)：根据列名返回索引值，如果不存在返回-1。
- ❑ getColumnName(int columnIndex)：从给定的索引值返回列名。
- ❑ getColumnNames()：返回一个字符串数组的列名。
- ❑ getCount()：返回Cursor 中的行数。