Android移动应用开发基础教程

讲授：葛新

第5章 数据存储

本章主要内容：

- 文件存储
- 共享存储
- SQLite数据库存储

5.1 文件存储

文件是一种基本的数据存储方式，适合于存储简单的文本或二进制数据。在使用文件时，可将其存放在内部存储器或外部存储器（SD卡等）中。

本节主要内容：

1. 读写内部存储文件
2. 读写外部存储文件
3. 应用的私有文件
4. 访问公共目录

5.1.1 读写内部存储文件

- Android运行应用程序直接在内部存储器中存放访问。默认情况下，保存到内部存储器中的文件是当前应用的私有文件，其他应用或用户不能访问。在卸载应用时，文件也会随之删除。
- Context类的openFileOutput()方法用于打开一个内部存储文件，向文件写入数据，其基本格式如下：

```
FileOutputStream fos =
        openFileOutput(FILENAME, Context.MODE_PRIVATE);
```

- openFileOutput()方法第一个参数为文件名，需注意的是文件名中不能包含路径。
- 第二个参数为访问模式，MODE_PRIVATE为默认模式，表示在指定文件存在时，原来的文件会被覆盖。MODE_APPEND表示在指定文件存在时，写入的数据会添加到文件末尾。
- 较早版本的Android还提供另外两种文件访问模式：MODE_WORLD_READABLE和MODE_WORLD_WRITEABLE，因为这两种模式容易容易引起安全漏洞，已在Android 4.2版本中被废弃。
- openFileOutput()方法返回一个FileOutputStream对象，使用该对象可将数据写入文件。例如，下面的代码将一个字符串写入内部存储文件。（实例项目：源代码\05\UseInternalStorage）

```
String FILENAME = "myfile";
String data = "在内部文件中的数据";
try {
    FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
    OutputStreamWriter osw=new OutputStreamWriter(fos);
    osw.write(data);
    osw.flush();
    fos.flush();
    osw.close();
    fos.close();
} catch (Exception e) {    e.printStackTrace();}
```

例如，下面的代码读出文件中的字符串。（实例项目：源代码\05\UseInternalStorage）

```
try {
    FileInputStream fis = openFileInput(FILENAME);
    InputStreamReader isr=new InputStreamReader(fis,"UTF-8");
    char[] data2=new char[fis.available()];
    isr.read(data2);
    isr.close();
    fis.close();
    TextView textView=(TextView)findViewById(R.id.textView);
    textView.setText(new String(data2));
} catch (Exception e) {    e.printStackTrace();}
```

5.1.2 读写外部存储文件

- 内部存储是设备自带的内部存储空间，外部存储空间是设备出厂时不存在，用户使用时添加的外部存储介质，例如TF卡、SD卡等。
- 要访问外部存储中的文件，首先应用必须具有READ_EXTERNAL_STORAGE（读）或WRITE_EXTERNAL_STORAGE（写）权限（写权限包含了读权限）。 可在应用的清单文件AndroidManifest.xml中为应用申请权限。例如：

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

即使为应用申请了权限，在应用安装到设备中之后，还需要在设备的“设置/应用管理”中找到该应用，为其启用存储访问权限，否则仍然无法访问外部存储卡。

- 内置的外部存储卡路径通常是/storage/emulated/0或者/mnt/sdcard，不同设备中可能有所区别。可用下面的方法来获得外部存储卡路径：

```
File sdcard=Environment.getExternalStorageDirectory();
```

在使用外部存储卡之前，应监测其状态

```
private boolean isReadable(){//检测存储卡是否可读
    String state = Environment.getExternalStorageState();
    if (state.equals(Environment.MEDIA_MOUNTED) ||
            state.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {        return true;    }
    return false;
}
private boolean isWritable(){//检测存储卡是否可写
    String state = Environment.getExternalStorageState();
    if (state.equals(Environment.MEDIA_MOUNTED)) {        return true;    }
    return false;
}
```

5.1.3 应用的私有文件

- Environment.getExternalStorageDirectory()返回的是第一个外部存储卡根目录。
- 对目前的绝大多数设备而言，第一个外部存储卡已经内置到设备中。建议不要直接访问外部存储卡的根目录。
- 如果存储数据的文件仅仅在当前应用中使用，可以使用应用程序私有的外部存储路径。
- Context.getExternalFilesDir()方法可获得应用程序的私有外部存储路径。存储在私有外部存储路径中的文件可称为应用的私有文件，这些文件会随着应用程序的卸载被删除。
- 例如，下面的代码说明如何创建私有文件：

```
File privatepath=Context.getExternalFilesDir();
File mf=new File(privatepath,"myfile.txt");
```

5.1.4 访问公共目录

- Android允许应用将文件存放到“公共”目录中，例如documents、download、music等等，以便与其他应用分享数据。
- 要获得相应的公共目录的 File，可调用Environment的getExternalStoragePublicDirectory()方法，其参数为目录类型，例如DIRECTORY_MUSIC、DIRECTORY_PICTURES或其他类型。
- 例如，下面的代码在公共目录documents中创建一个TXT文件。（实例项目：源代码\05\UsePublicPath）

```
If(!isWritable()){
  Toast.makeText(this,"SD卡不可用",Toast.LENGTH_SHORT).show();
  return;
}
```

```
File sdcard=
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
File mf=new File(sdcard,"myfile.txt");
try {
    mf.createNewFile();
    Toast.makeText(this,"成功创建文件",Toast.LENGTH_SHORT).show();
} catch (Exception e) {
    Toast.makeText(this,e.getMessage(),Toast.LENGTH_SHORT).show();
}
```

5.2 共享存储

共享存储是采用SharedPreferences来保存数据。SharedPreferences虽然也用文件来保存数据，但存取数据的方式有所区别。

SharedPreferences使用键值对的方式存储数据。在保存数据时，需要为数据提供一个相对唯一的键。

在读取数据时，通过键把相应的值取出。SharedPreferences支持多种不同类型的数据存储，包括boolean、int、long、float、String以及Set<String>等。

本节主要内容：

1. 将数据存入SharedPreferences文件
2. 读取SharedPreferences文件数据
3. 实现记住密码功能

5.2.1 将数据存入SharedPreferences文件

- 要将数据存入SharedPreferences文件需要下列几个步骤。
  - 获得SharedPreferences对象。
  - 获得SharedPreferences对象的Editor对象。
  - 调用Editor对象的方法向文件添加数据。
  - 提交数据，完成数据存储操作。

1、获得SharedPreferences对象

- 第一种方法：调用Context类的getSharedPreferences()方法。例如：

```
SharedPreferences pref=
        getSharedPreferences("myPreferences",MODE_PRIVATE);
```

  - getSharedPreferences()方法第一个参数是SharedPreferences文件的名称，第2个参数是操作模式。
  - MODE_PRIVATE是默认操作模式，等价于0。
  - MODE_PRIVATE表示文件属于当前应用的私有文件，其他应用不能访问。
  - 另一种模式是MODE_APPEND，表示在指定文件存在时，向文件中添加数据。

1、获得SharedPreferences对象

- 第二种方法：调用Activity类的getPreferences()方法。例如：

  SharedPreferences pref=getPreferences(MODE_PRIVATE);

  - getPreferences()方法的参数指定文件操作模式，它默认以当前活动的类名作为SharedPreferences文件的名称。

1、获得SharedPreferences对象

- 第三种获得SharedPreferences对象的方法是调用PreferenceManager类的getDefaultSharedPreferences()方法。例如：
  - SharedPreferences pref= PreferenceManager.getDefaultSharedPreferences(this);
  - getDefaultSharedPreferences()方法参数为当前应用上下文，它默认以当前应用的包名作为SharedPreferences文件的名称。

2、获得SharedPreferences对象的Editor对象

- 调用SharedPreferences对象的edit()方法可创建一个Editor对象。
- 例如：
  - SharedPreferences.Editor editor=pref.edit();

3、调用Editor对象的方法向文件添加数据

- 调用Editor对象的各种putXXX()方法可向SharedPreferences文件添加数据。例如：

```
editor.putString("username","admin");
editor.putString("password","12345");
editor.putBoolean("remembered",true);
```

- putXXX()方法第1个参数为键，第2个参数为通过键保存的数据（值）。

4、提交数据，完成数据存储操作

- 在调用putXXX()方法添加了数据后，必须调用Editor对象的apply()方法提交数据，才能将数据存入文件，完成数据存储操作。
- 例如：

```
editor.apply();
```

5.2.2 读取SharedPreferences文件数据

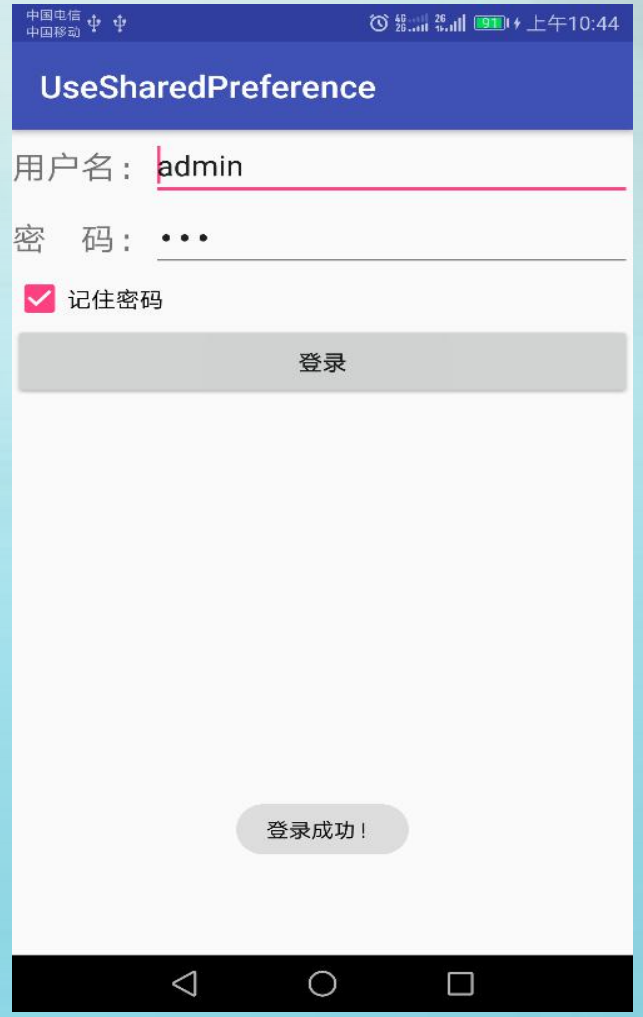
- 获得SharedPreferences对象后，调用相应的getXXX()方法读取存储在文件中的数据。例如：

```
boolean isRemembered=pref.getBoolean("remembered",false);
etName.setText(pref.getString("username",""));
etPwd.setText(pref.getString("password",""));
```

- getXXX()方法第1个参数为键，第2个参数为默认值。如果SharedPreferences文件中无指定的键，则getXXX()方法返回第2个参数指定的默认值。

5.2.3 实现记住密码功能

- 通常，应用的登录界面中往往会提供一个记住密码功能，避免用户下一次登录时再次输入登录信息。
- 下面的实例使用SharedPreferences文件来存储登录界面中输入的登录信息，具体操作步骤如下。（实例项目：源代码\05\UseSharedPreferences）



5.3 SQLite数据库存储

SQLite是一款轻量级的关系数据库，它运算速度快，运行内存少，只需几百KB的内存即可，因而适用于移动设备。SQLite不仅支持标准的SQL语法，还支持ACID失误。Android系统内置了SQLite数据库，使得在Android应用中可以轻松适用数据库来完成数据存储。

本节主要内容：

1. 创建数据库
2. 升级数据库
3. 添加数据
4. 更新数据
5. 删除数据
6. 查询数据
7. 执行SQL命令操作数据库

5.3.1 创建数据库

- Android提供了一个抽象类SQLiteOpenHelper来帮助我们使用SQLite数据库，借助该类，可以很方便地实现数据库的创建、升级以及数据的管理。
- SQLiteOpenHelper是一个抽象类，所以需要创建一个类来继承它，并实现需要的方法。
- SQLiteOpenHelper的子类必须实现两个方法：onCreate()和OnUpgrade()方法。
- onCreate()方法在创建数据库时被调用，完成数据库的初始化操作，例如创建数据表、添加初始数据等。
- OnUpgrade()方法在升级数据库时调用。

- SQLiteOpenHelper类提供了getWritableDatabase()和getReadableDatabase()两个方法用于打开或创建数据库。
- 如果指定的数据库存在，则直接打开，否则创建一个新的数据库。getWritableDatabase()和getReadableDatabase()都返回一个SQLiteDatabase实例对象，通过该对象完成对数据库的各种操作。
- 如果数据库无法写入数据（如磁盘空间已满）时，getReadableDatabase()返回一个只读数据库对象，此时使用getWritableDatabase()方法则会出错。

- SQLiteOpenHelper类提供了两个构造方法：

```
SQLiteOpenHelper(Context context, String name,
                 SQLiteDatabase.CursorFactory factory,
                int version)
SQLiteOpenHelper(Context context,
                 String name,
                 SQLiteDatabase.CursorFactory factory,
                 int version,
                 DatabaseErrorHandler errorHandler)
```

- 参数context为上下文对象，name为数据库名称，factory是用于创建保存查询结果的自定义cursor对象（一般使用null表示使用默认cursor对象），version为数据库版本号（从1开始）。

MySQLiteHelper类

```
package com.example.xbg.usesqlite;

……

SQLiteOpenHelper {
    private static String CREATE_TABLE_USER="create table users("+
            "id integer primary key autoincrement,"+
            "userid text,password text)";
    private Context sContext;
    public MySQLiteHelper(Context context, String name,
                          SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
        sContext=context;
    }
```

```
@Override
    public void onCreate(SQLiteDatabase db) {
        //执行数据库初始化操作
        db.execSQL(CREATE_TABLE_USER);
        Toast.makeText(sContext," 成功创建数据表",Toast.LENGTH_LONG).show();
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //执行数据库升级操作
    }
```

创建MySQLiteHelper类对象，用于创建数据库

```
public class MainActivity extends AppCompatActivity {
    private MySQLiteHelper sqLiteHelper;
    private SQLiteDatabase myDb;
    TextView tvPath;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
……
        btCreateDb.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sqLiteHelper=new MySQLiteHelper(MainActivity.this,"usersdb.db",null,1);
                myDb=sqLiteHelper.getWritableDatabase();//完成创建数据库
                String path=myDb.getPath();
                tvPath.setText("数据库："+path);//显示数据库文件及其路径
```

调用SQLiteOpenHelper删除数据库

```
Button btDeleteDb=(Button)findViewById(R.id.btDeleteDb);
btDeleteDb.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    if(myDb.isOpen()){
        myDb.close();                                   //若数据库以打开，则先将其关闭
    }
    String path=myDb.getPath();                         //获得数据库文件名（含路径）
    File db=new File(path);
    SQLiteDatabase.deleteDatabase(db);                  //删除数据库
    tvPath.setText("数据库已删除");                      //显示数据库文件及其路径
```

5.3.2 升级数据库

- SQLiteOpenHelper类的构造方法中的数据库版本号用于升级或降级数据库。
- 若提供的版本号比当前版本号大，则调用onUpgrade()方法当前数据库升级。
- 如果提供的版本号比当前版本号小，则调用onDowngrade()方法对数据库进行降级。
- 在上一节中，我们创建了一个数据库usersdb.db，并为其创建一个users表。users表保存用户ID和登录密码。现在如果需要增加一个数据表保存用户类型，不同类型具有不同的权限。同时，实现升级数据库功能，在升级时重建数据库中的表。

修改MySQLiteHelper类

```
package com.example.xbg.usesqlite;
import android.content.Context;
……
public class MySQLiteHelper extends SQLiteOpenHelper {
    private static String CREATE_TABLE_USER="create table users("+
            "id integer primary key autoincrement,"+
            "userid text,password text)";
    private static String CREATE_TABLE_TYPE="create table types("+
            "id integer primary key autoincrement,"+
            "type_code,describe text)";
……
```

```
public void onCreate(SQLiteDatabase db) {
        //执行数据库初始化操作
        db.execSQL(CREATE_TABLE_USER);
        db.execSQL(CREATE_TABLE_TYPE);
        Toast.makeText(sContext,"成功创建数据表",Toast.LENGTH_LONG).show();
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //执行数据库升级操作
        db.execSQL("drop table if exists users");
        db.execSQL("drop table if exists types");
        onCreate(db);
    }
```

修改MainActivity，添加一个按钮来执行数据库升级操作

```
Button btUpgradeDb=(Button)findViewById(R.id.btUpgradeDb);
btUpgradeDb.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    sqLiteHelper=
            new MySQLiteHelper(MainActivity.this,"usersdb.db",null,2);
    myDb=sqLiteHelper.getWritableDatabase();//完成数据库升级
    }
});
```

5.3.3 添加数据

- QLiteDatabase对象的insert()方法用于为表添加记录。insert()方法基本格式如下：

  insert(String table, String nullColumnHack, ContentValues values)

- 参数table指定要添加记录的表的名称。参数nullColumnHack指定记录中需要赋值为Null的列名，可用Null作参数表示没有列需要赋值为Null。参数values 包含要添加的记录数据。

单击按钮时，将用户输入的记录数据添加到表

```
Button btAdd=(Button)findViewById(R.id.btAdd);
btAdd.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(myDb==null){            return;}                    //在没有创建数据库时，不执行添加数据操作
        ContentValues cv=new ContentValues();
        EditText etID= (EditText) findViewById(R.id.etName);
        EditText etPwd= (EditText) findViewById(R.id.etPassword);
        String name=etID.getText().toString();
        String password=etPwd.getText().toString();
        cv.put("userid",name);
        cv.put("password",password);
        myDb.insert("users",null,cv);                                    //将数据添加到数据表
        Toast.makeText(MainActivity.this,"成功添加记录",Toast.LENGTH_LONG).show();
        refreshList();
```

5.3.4 更新数据

- SQLiteDatabase对象的update()方法用于更新数据。update ()方法基本格式如下：

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

- 参数table指定要更新的数据所在的表的名称。
- 参数values 包含要更新的列的值。
- 参数whereClause指定记录筛选条件，只有符合条件的记录才修改指定列，其中用问号指定需要填充的参数。
- whereArgs指定要填充到whereClause中的参数值。
- update()方法返回被更新的记录数。

用输入的数据更新表中的记录

```
Button btUpdate=(Button)findViewById(R.id.btUpdate);
btUpdate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(myDb==null){ return;}//在没有创建数据库时，不执行后继操作
        EditText etID= (EditText) findViewById(R.id.etName);
        EditText etPwd= (EditText) findViewById(R.id.etPassword);
        String name=etID.getText().toString();
        String password=etPwd.getText().toString();
        ContentValues cv=new ContentValues();
        cv.put("password",password);
        myDb.update("users",cv,"userid=?",new String[]{name});
        Toast.makeText(MainActivity.this,"成功修改记录",Toast.LENGTH_LONG).show();
        refreshList();
```

5.3.5 删除数据

- SQLiteDatabase对象的delete()方法用于删除数据。delete()方法基本格式如下：
  - delete(String table, String whereClause, String[] whereArgs)
  - 各参数意义与update()方法类似，只有符合条件的记录才被删除。
  - delete()方法返回被删除的记录数。

将用户输入的数据作为条件删除表中的记录

```
Button btDelete=(Button)findViewById(R.id.btDelete);
btDelete.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(myDb==null){
            return;//在没有创建数据库时，不执行后继操作
        }
        EditText etID= (EditText) findViewById(R.id.etName);
        String name=etID.getText().toString();
        myDb.delete("users","userid=?",new String[]{name});
        Toast.makeText(MainActivity.this,"成功删除记录",Toast.LENGTH_LONG).show();
        refreshList();
```

5.3.6 查询数据

- SQLiteDatabase对象的query()方法用于查询数据。常用query()方法有下列3种基本格式如下：
  - query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
  - query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
  - query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)

各个参数的含义

- distinct：为true时，表示返回结果中不包含重复值。
- table：指定查询的表名称，对应SQL SELECT命令的“from 表名称”部分。
- columns：指定查询结果中包含的列名称，对应SQL SELECT命令的“select 列名称1,列名称2,……”部分。
- selection：指定记录筛选条件，对应SQL SELECT命令的“where 条件”部分。
- selectionArgs：指定填充筛选条件占位符的参数。
- groupBy：指定查询分组列名称，对应SQL SELECT命令的“group by 列名称1,列名称2,……”部分。
- having：指定查询分组的条件，对应SQL SELECT命令的“having 条件”部分。
- orderBy：指定查询结果排序列名称，对应SQL SELECT命令的“order by 列名称1,列名称2,……”部分。
- limit：指定返回的查询结果的最大记录数。

查询users表中的数据，并使用Toast显示

```
Button btGetAll=(Button)findViewById(R.id.btGetAll);
btGetAll.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(myDb==null){ return;}//在没有创建数据库时，不执行后继操作
        Cursor c=myDb.query("users",null,null,null,null,null,null);
        String msg="当前记录如下：\n";
        if(c.moveToFirst()){
            do{
                msg=msg+"userid:"+c.getString(c.getColumnIndex("userid"))+
                "  password="+c.getString(c.getColumnIndex("password"))+"\n";
            }while(c.moveToNext());
        }
        Toast.makeText(MainActivity.this,msg,Toast.LENGTH_LONG).show();
```

使用Cursor对象创建适配器填充ListView控件

- 自定义列表项布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:text="用户ID："
        android:layout_width="wrap_content" android:layout_height="wrap_content" />
    <TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:id="@+id/textID"  android:layout_weight="1" />
    <TextView  android:text="密码："
        android:layout_width="wrap_content" android:layout_height="wrap_content"/>
    <TextView  android:layout_width="wrap_content"  android:layout_height="wrap_content"
        android:id="@+id/textPwd"  android:layout_weight="1" />
</LinearLayout>
```

填充ListView控件

```
private void refreshList(){
    Cursor c=myDb.query("users",new String[]{"id as _id","userid","password"},
            null,null,null,null,null);
    SimpleCursorAdapter adapter=new SimpleCursorAdapter(MainActivity.this,
            R.layout.recordlist, c,new String[]{"userid","password"},
            new int[]{R.id.textID,R.id.textPwd});
    ListView lv= (ListView) findViewById(R.id.lvRecords);
    lv.setAdapter(adapter);
}
```

5.3.7 执行SQL命令操作数据库

- SQLiteDatabase类也支持SQL命令来完成各种数据操作，简单介绍如下。
  - 添加记录

    myDb.execSQL("insert into users(userid,password) values(?,?)", new String[]{name,password});
  - 修改记录

    myDb.execSQL("update users set password=? where userid=?",new String[]{password,name});
  - 删除记录

    myDb.execSQL("delete from users where userid=?",new String[]{name});
  - 查询

    Cursor c=myDb.rawQuery("select id as _id,userid,password from users",null);