数据存储

2018年4月26日 8:13

数据持久化指将那些内存中的瞬时数据保存到存储设备中，保证即使在手机或电脑关机的情况下，这些数据仍然不会丢失。

Android系统中提供3种方式实现数据持久化功能：**文件存储**、**SharesPreference存储**以及**数据库存储**

- **文件存储**

Android中最基本的一种数据存储方式，它不对存储的内容进行任何的格式化处理，所有数据都是原封不动的保存到文件当中。 ——**适合存储一些简单的文本数据或二进制数据**

（文件默认存储到/data/data/<packagename>/files/目录下）

> Context类中提供openFileOutput（）方法，可以用于将数据存储到指定的文件中。（参数一：文件名、参数二：文件的操作模式） return 一个FileOutputStream对象，得到对象之后，再使用java流的方式将数据写入到文件中

```
public void save(String inputText) {
        FileOutputStream out = null;
        BufferedWriter writer = null;
        try {
            out = openFileOutput("data", Context.MODE_PRIVATE); //文件命名为data
            writer = new BufferedWriter(new OutputStreamWriter(out));
            writer.write(inputText);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (writer != null) {
                    writer.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

> Context类中提供openFileInput（）方法，可以从文件中读取数据。（只有一个参数，即要读取的文件名，系统会自动到默认目录下去加载这个文件，并返回一个FileInputStream对象，再通过java流的方式将数据读取出来）

注意！！对字符串进行非空判断的时候使用TextUtils.isEmpty（）方法，可以一次性进行两种

空值的判断

```
public String load() {
        FileInputStream in = null;
        BufferedReader reader = null;
        StringBuilder content = new StringBuilder();
        try {
            in = openFileInput("data");  //获取到一个FileInputStream对象
            reader = new BufferedReader(new InputStreamReader(in));
            String line = " ";
            while ((line = reader.readLine()) != null) {
                content.append(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return content.toString();
    }
```

- **SharedPreferences存储**

SharedPreferences 是使用键值对的方式来存储数据的。（当保存一条数据的时候，需要给这条数据提供一个对应的键，这样再读取数据的时候就可以通过这个键把相应的值取出来）
(SharedPreferences文件默认目录：/data/data/<packagename>/shared_prefs/)

✧ 获取到SharedPreferences对象：

1) Context类中getSharedPreferences（）方法
   接收两个参数：一用于指定SharedPreferences文件的名称；二用于指定操作模式
2) Activity类中的getPreferences（）方法
   只接收一个参数，类同于Context类中getSharedPreferences（）方法，自动将当前活动的类名作为SharedPreferences的文件名
3) PreferenceManager类中的getDafaultSharesPreferences（）方法
   静态方法，接收一个context参数，并自动使用当前应用程序的包名作为前缀来命名SharedPreferences文件。得到SharedPreferences对象之后，**开始向文件中存储数据**：
   ①调用SharedPreferences对象的edit（）方法来获取一个SharedPreferences.Editor

对象
②向SharedPreferences.Editor对象中添加数据（putString（）方法）
③调用apply（）方法将添加的数据提交，从而完成数据存储操作

```
public void onClick(View view) {
        SharedPreferences.Editor editor =
getSharedPreferences("data",MODE_PRIVATE).edit();    //指定文件名为data，得到SharedPreferences.Editor对象
        editor.putString("name","Tom");
        editor.putInt("age",28);
        editor.putBoolean("married",false);
        editor.apply();  //提交，完成数据存储的操作
         }
```

从SharedPreferences中读取数据：

SharedPreferences.Editor提供一系列get方法(两个参数，第一个是键，传入存储数据时使用的键就可以得到相应的值；第二个参数是默认值，即表示当传入的键找不到对应的值时会以怎样的默认值进行返回)

```
 public void onClick(View view) {
        SharedPreferences pref = getSharedPreferences("data",MODE_PRIVATE); //得到 SharedPreferences对象
        String name = pref.getString("name","");  //如果没有找到对应的值，就会使用方法中传入的默认值来代替
        int age = pref.getInt("age",0);
        boolean married = pref.getBoolean("married",false);
         }
     });
```

记住密码：

>>>

```
if(account.equals("admin") && password.equals("123456")){
                editor = pref.edit();
                if(rememberPass.isChecked()) {  //检查复选框是否被选中
                    editor.putBoolean("remember_password",true);
                    editor.putString("account",account);
                    editor.putString("password",password);
                }else{
                    editor.clear();
                }
                editor.apply();
```

>>

>>>>>>>>>>

```
 rememberPass = (CheckBox) findViewById(R.id.remember_pass);
        pref = PreferenceManager.getDefaultSharedPreferences(this);
        boolean isRemember = pref.getBoolean("remember_password",false);
        if(isRemember){
            //将账号和密码都设置到文本框中
            String account = pref.getString("account","");
            String password = pref.getString("password","");
            accoutEdit.setText(account);
            passwordEdit.setText(password);
            rememberPass.setChecked(true);
        }
```

>>>

- **SQLite数据库存储**

SQL 是用于访问和处理数据库的标准的计算机语言

SQLite是一种轻量级的关系型数据库（存储大量复杂的关系型数据）

SQLiteOpenHelper是一个抽象类：

抽象方法：onCreate（）

onUpgrade（） //重写这两个方法，然后在这两个方法中去创建、升级数据库的逻辑

实例方法：getReadableDatabase（）

getWritableDatabase（）

//这两个方法都能创建或打开一个现有的数据库，并返回一个可对数据库进行读写操作的对象

构造方法参数：第一个参数是Context

第二个参数是数据库名，创建数据库时使用的就是这里指定的名称

第三个参数允许我们在查询数据的时候返回一个自定义的Cursor，一般null

第四个参数是当前数据库的版本库，可用于对数据库进行升级操作

1) 构建出SQLiteOpenHelper实例
2) 调用getReadableDatabase（）或者getWritableDatabase（）方法创建数据库
3) 重写onCreate（）方法（处理一些创建表的逻辑）

integer表示整型		real表示浮点型		text表示文本类型		blob表示二进制类型	

```
public static final String CREATE_BOOK = "create table Book ("
            +"id integer primary key autoincrement,"
            +"author text,"
            +"price real,"
```

```
        +"pages integer,"
        +"name text)";
//添加一张表
    public static final String CREATE_CATEGORY = "create table Category("
        +"id integer primary key autoincrement,"
        +"category_name text,"
        +"category_code integer)";
//在onUpgrade（）方法中进行升级
@Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
        sqLiteDatabase.execSQL("drop table if exists Book");
        sqLiteDatabase.execSQL("drop table if exists Category");
        onCreate(sqLiteDatabase);
    }
dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);// 只要version大于1即可
```

数据操作：C(Create添加...insert方法）R（Retrieve 查询...select方法）U（Update 更新 ... update方法）D （Delete 删除 ... delete方法） <<<对SQLiteDatabase对象进行操作>>>

- ✓ insert()方法：专门用于添加数据

  （参数一：表名 参数二：null 参数三： **ContentValues对象**，提供一些列put（）方法重载，用于向ContentValues中添加数据，只需要将表中的每个列名以及相应的待添加数据传入即可）

```
addData.setOnClickListener(new View.OnClickListener(){
@Override
public void onClick(Viewview){
SQLiteDatabase sqLiteDatabase = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
//开始组装第一条数据
values.put("name","TheDaVinciCode");
values.put("author","DanBrown");
values.put("pages",454);
values.put("price",16.96);
sqLiteDatabase.insert("Book",null,values);//插入第一条数据
values.clear();
```

- ✓ update（）方法：向表中添加数据

  （参数一：表名 参数二：**ContentValues对象**，要把更新数据在这里组装进去 参数三、四：用于约束更新某一行或某几行中的数据，不指定的话默认就是更新所有行）

```
Button updateData = (Button) findViewById(R.id.update_data);
updateData.setOnClickListener(new View.OnClickListener(){
@Override
public void onClick(View view){
SQLiteDatabase sqLiteDatabase = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("price",10.99);
sqLiteDatabase.update("Book",values,"name=?",new String[]{"The Da Vinci Code"});
```

```
}
});
```

✓ delete（）方法：从表中删除数据

（参数一：表名　　参数二、三：用于约束删除某一行或某几行的数据，不指定的话默认就是删除所有行）

```
Button deleteButton = (Button) findViewById(R.id.delete_data);
deleteButton.setOnClickListener(new View.OnClickListener(){
@Override
public void onClick(View view){
SQLiteDatabase SQLiteDatabase = dbHelper.getWritableDatabase();
SQLiteDatabase.delete("Book","pages>?",new String[]{"500"});
}
});
```

✓ query（）方法：对数据进行查询

（参数一：表名 参数二：指定去查询哪几列，不指定则默认查询所有行的数据 参数三、四：约束查询的列，不指定则默认查询所有行的数据 参数五：指定需要去group by的列，不指定则表示不对查询结果进行group by操作 参数六: 对group by数据进行过滤，不指定则表示不进行过滤 参数七： 指定查询结果的排序方式，不指定则表示使用默认的排序方式）

调用query（）方法后会返回一个cursor对象，查询到的所有数据都将从这个对象中取出

```
Button queryButton = (Button) findViewById(R.id.query_data);
queryButton.setOnClickListener(new View.OnClickListener(){
@Override
public void onClick(View view){
SQLiteDatabase sqLiteDatabase = dbHelper.getWritableDatabase();
//查询Book表中所有的数据
Cursor cursor = sqLiteDatabase.query("Book",null,null,null,null,null,null);
if(cursor.moveToFirst()){//将数据的指针移动到第一行的位置
do{
//遍历Cursor对象，取出数据并打印
String name = cursor.getString(cursor.getColumnIndex("name"));
String author = cursor.getString(cursor.getColumnIndex("author"));
int pages = cursor.getInt(cursor.getColumnIndex("pages"));
double price = cursor.getDouble(cursor.getColumnIndex("price"));
Log.d("MainActivity","booknameis"+ name);
Log.d("MainActivity","bookauthoris"+ author);
Log.d("MainActivity","bookpagesis"+ pages);
Log.d("MainActivity","bookpriceis"+ price);
}while(cursor.moveToNext());
}
cursor.close();
}
});
```

直接使用SQL来完成CRUD操作：

1) 添加数据的方法如下：

```
db.execSQL("insert into Book (name, author, pages, price) values(?, ?, ?, ?)",
        new String[] { "The Da Vinci Code", "Dan Brown", "454", "16.96" });
db.execSQL("insert into Book (name, author, pages, price) values(?, ?, ?, ?)",
        new String[] { "The Lost Symbol", "Dan Brown", "510", "19.95" });
```

2) 更新数据的方法如下：

```
db.execSQL("update Book set price = ? where name = ?",
                new String[] { "10.99","The Da Vinci Code" });
```

3) 删除数据的方法如下：

```
db.execSQL("delete from Book where pages > ?", new String[] { "500" });
```

4) 查询数据的方法如下：

```
db.rawQuery("select * from Book", null);
```