**移动终应用系统开发**

**实**

**验**

**指**

**导**

**书**

**章崇文**

**202202296**

**计算机222**

**计算机技术实验中心**

**2024年12月14日**

# **实验二 数据存储技术**

一、实验目的

1.掌握SharePreference存储方法

2.掌握SQLite数据库操作方法

3.掌握ContentProvider共享数据操作方法

二、实验要求

1.熟练使用SharePreference存取数据

2.熟练掌握SQLite数据库增、删、查、改方法

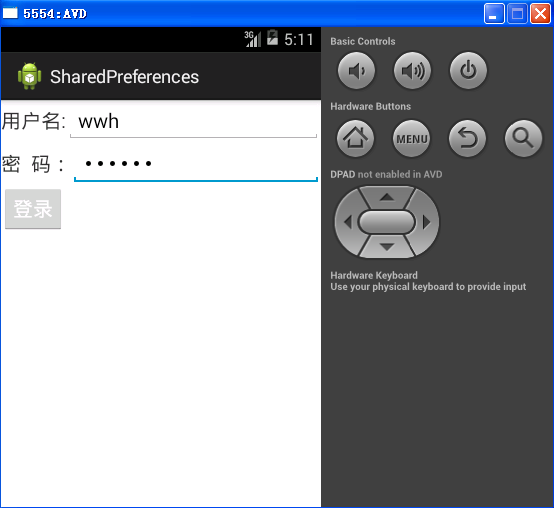
3.熟练掌握ContentProvider类封装数据，并提供方法给外部使用

三、实验内容及步骤

程序1

要求：

**1.将一个Activity里面所输入的数据存入到SharePreference中，然后在另外个Activity中读取SharePreference里面的数据，如图所示。**

**指导：**

**具体步骤如下：**

**1）主登录Activity的布局文件代码如下：**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"*

android:orientation=*"vertical"* >

<LinearLayout

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"* >

<TextView

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"用户名:"*

android:textSize=*"20dp"* />

<EditText

android:id=*"@+id/username"*

android:layout\_width=*"0dip"*

android:layout\_height=*"wrap\_content"*

android:layout\_weight=*"1"*

android:inputType=*"text"*

android:textSize=*"20dp"* >

<requestFocus />

</EditText>

</LinearLayout>

<LinearLayout

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"* >

<TextView

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"密 码："*

android:textSize=*"20dp"* />

<EditText

android:id=*"@+id/password"*

android:layout\_width=*"0dip"*

android:layout\_height=*"wrap\_content"*

android:layout\_weight=*"1"*

android:inputType=*"textPassword"*

android:textSize=*"20dp"* />

</LinearLayout>

<Button

android:id=*"@+id/login"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"登录"*

android:textSize=*"20dp"* />

</LinearLayout>

**2）第二个Activity的布局文件代码如下：**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"*

android:orientation=*"vertical"* >

<TextView

android:id=*"@+id/username"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:textSize=*"20dp"* />

<TextView

android:id=*"@+id/password"*

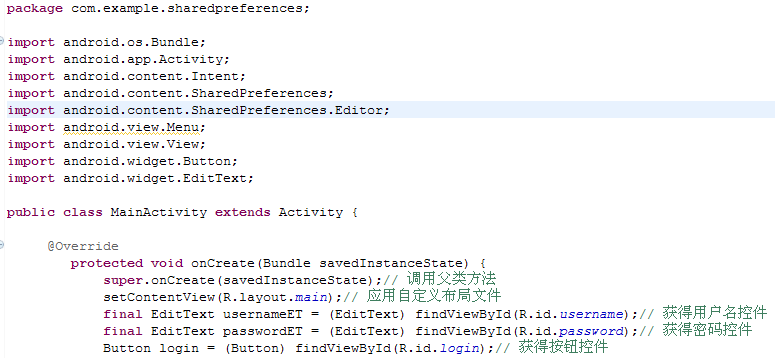
android:layout\_width=*"wrap\_content"*

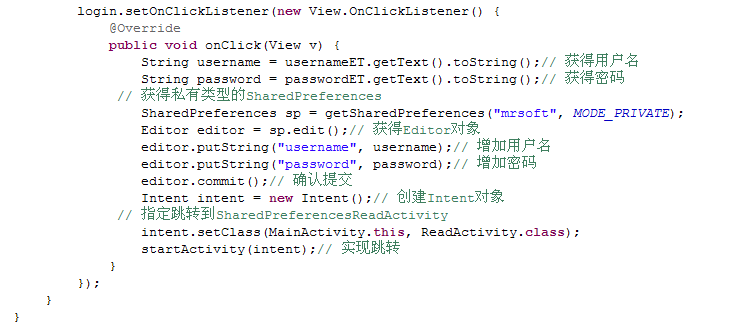
android:layout\_height=*"wrap\_content"*

android:textSize=*"20dp"* />

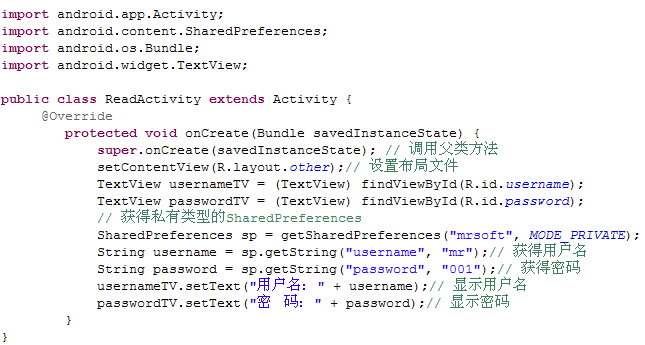
</LinearLayout>

**3）主Activity的后台Java代码中，将EditText所填入的数据存储到SahrePreferences中，代码如下：**

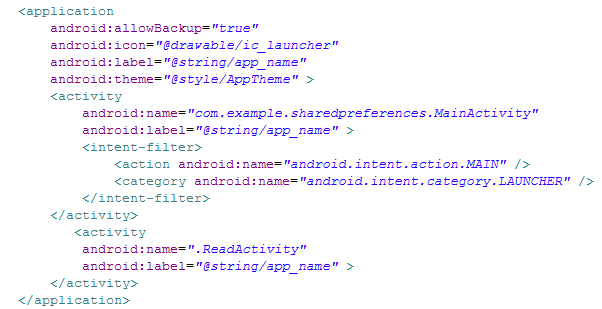
****



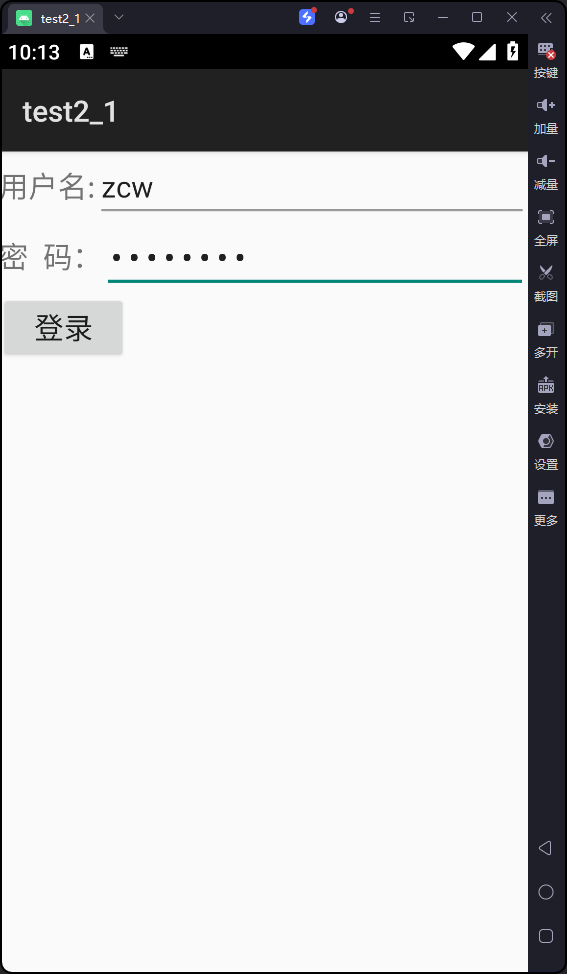
**4）在第二个Activity的后台Java代码中，将SharePreferences里面的数据读取出来，并显示之，具体代码如下：**



**5） 在AdroidManifest.xml中，对两个Activity进行注册，代码如下：**



程序演示

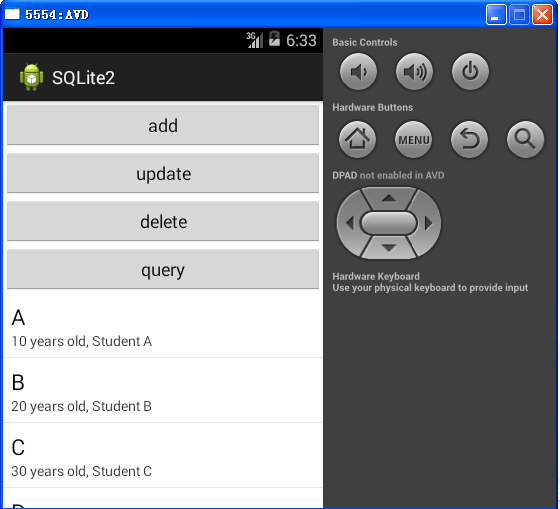
图形用户界面, 应用程序

描述已自动生成

程序2

要求：

**1.创建一个数据库，添加一个数据表，完成数据的增、删、查、改操作，如图所示。**



指导

具体步骤如下：

**1）主Activity的布局文件代码如下：**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:orientation=*"vertical"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"fill\_parent"*>

<Button

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:text=*"add"*

android:onClick=*"add"*/>

<Button

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:text=*"update"*

android:onClick=*"update"*/>

<Button

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:text=*"delete"*

**android:onClick=*"delete"*/>**

<Button

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:text=*"query"*

android:onClick=*"query"*/>

<ListView

android:id=*"@+id/listView"*

android:layout\_width=*"fill\_parent"*

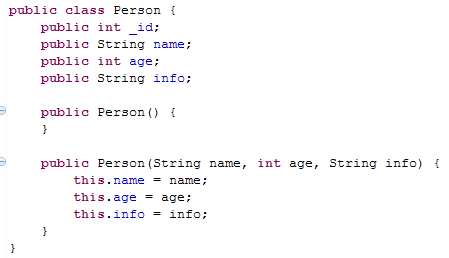
android:layout\_height=*"wrap\_content"*/>

</LinearLayout>

**2）在包下面，创建一个继承于SQLiteHelper的类，代码如下：**

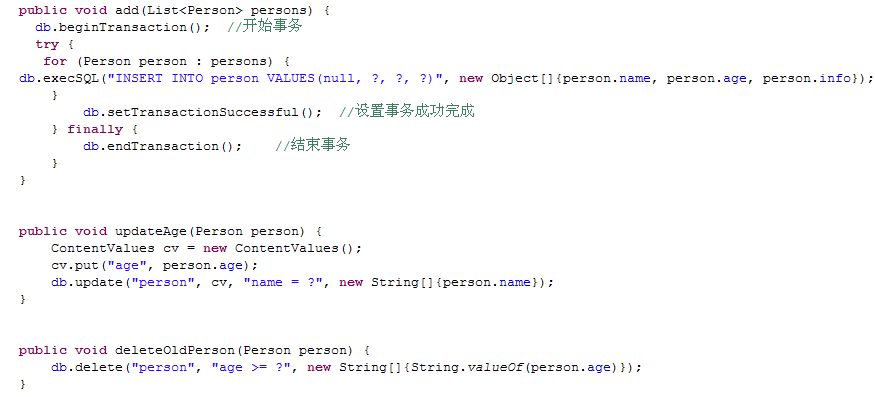


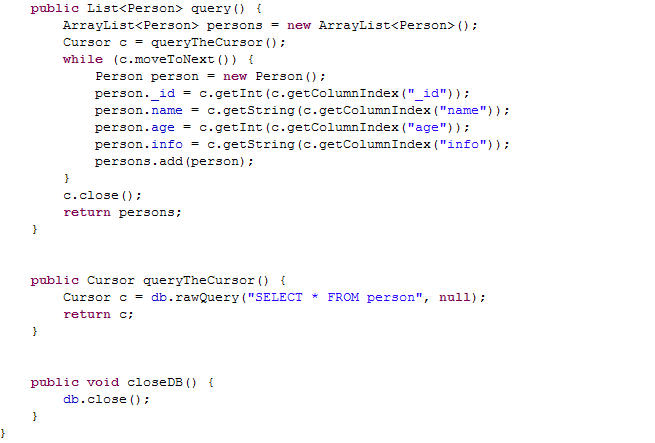
**3）在包下面创建一个person类，对应数据表的4个字段，代码如下：**



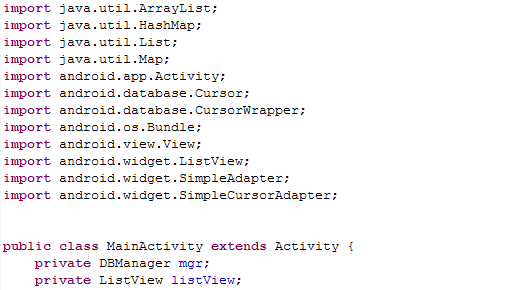
**4）在包下面创建一个DBManager类，相当于业务逻辑层，封装了对数据库增、删、查、改操作，代码如下：**

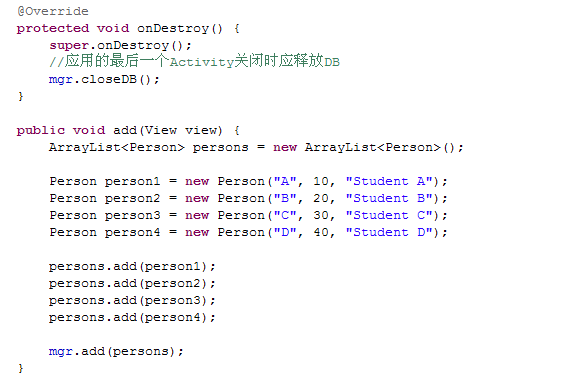


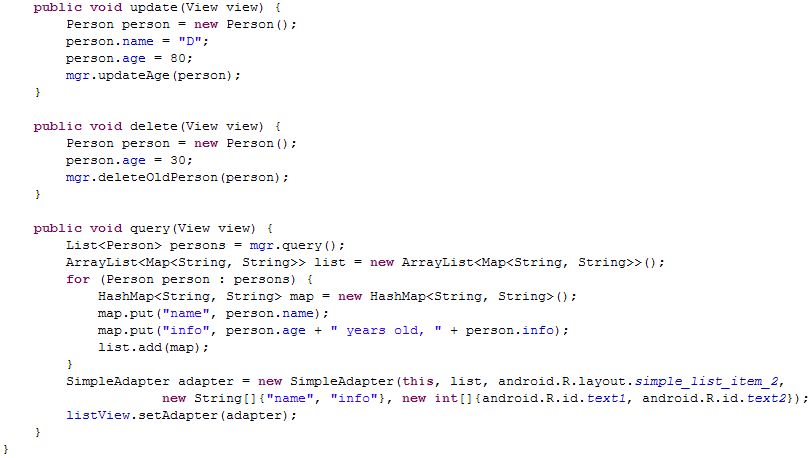




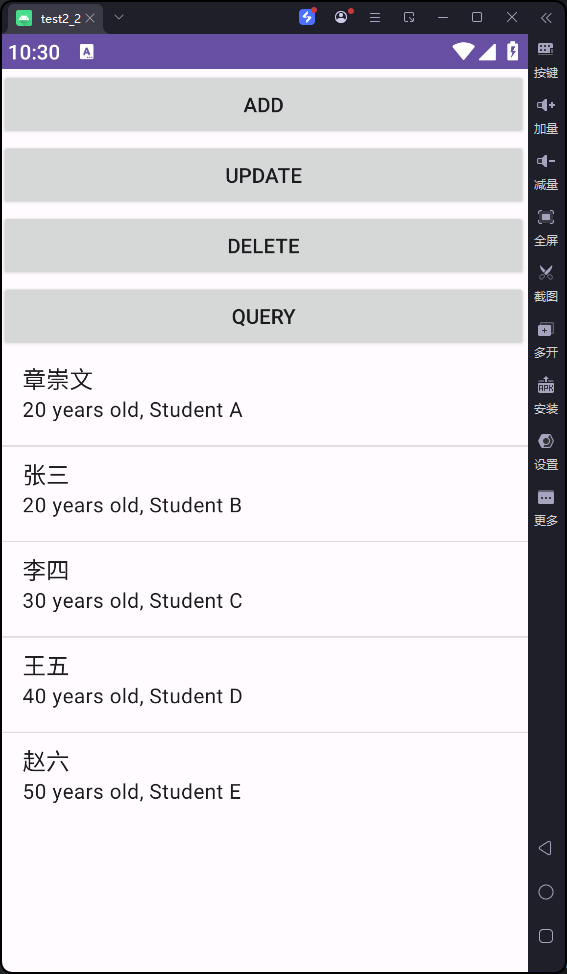
**5）主Activity类的Java代码如下：**







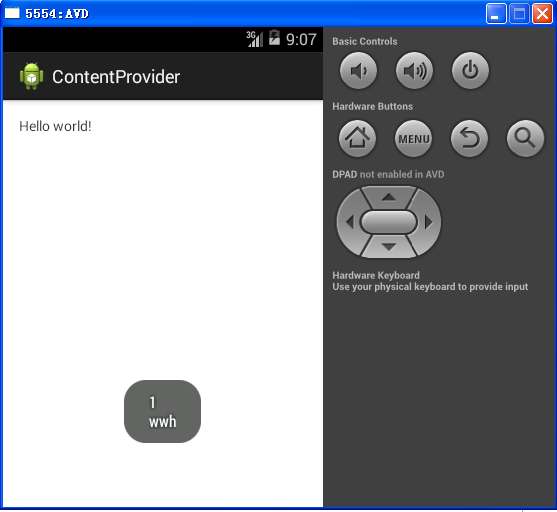
程序演示



程序3

要求：

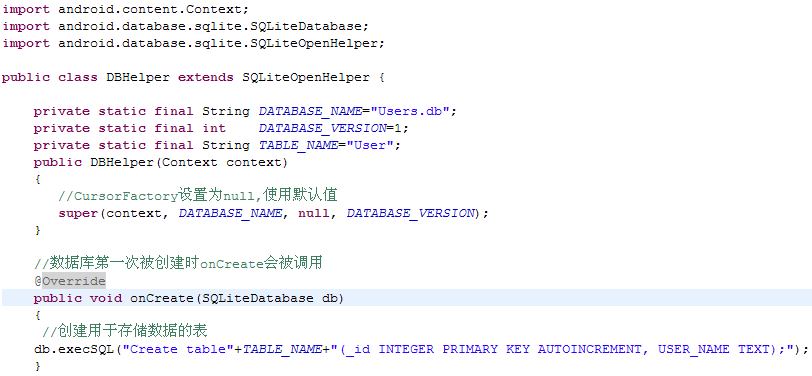
**1.采用ContentProvider封装数据操作，并通过URI曝露给外部应用程序使用。下图是将数据存储到ContentProvider中，然后读取其中数据并显示出来。**

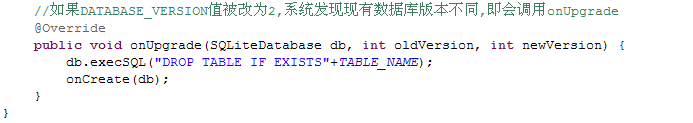


指导：

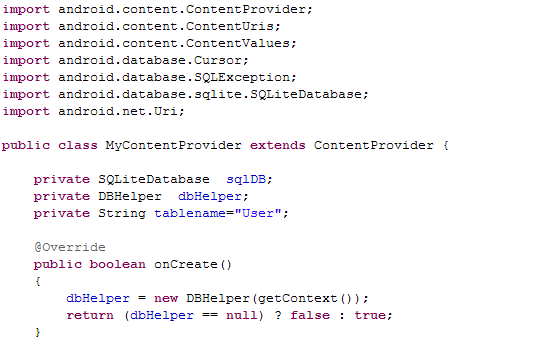
具体步骤如下：

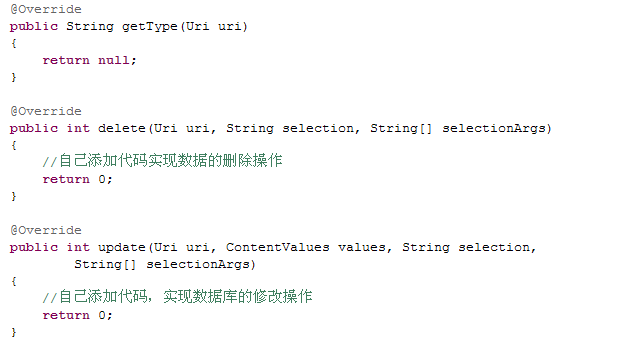
**1）由于底层数据采用SQLite数据库来存储，所以先创建一个SQLiteHelper子类，代码如下：**

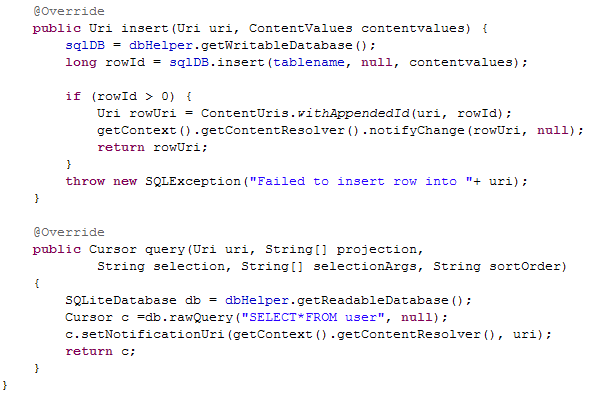




**2）编写ContentProvider的子类，需要重载onCreate()、getType()、delete()、update()、insert()、query()这几个方法，具体代码如下：**







**3）在AndroidManifest.xml中，注册ContentProvider类，代码如下：**

<application

android:allowBackup=*"true"*

android:icon=*"@drawable/ic\_launcher"*

android:label=*"@string/app\_name"*

android:theme=*"@style/AppTheme"* >

<activity

android:name=*"com.example.contentprovider.MainActivity"*

android:label=*"@string/app\_name"* >

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

</activity>

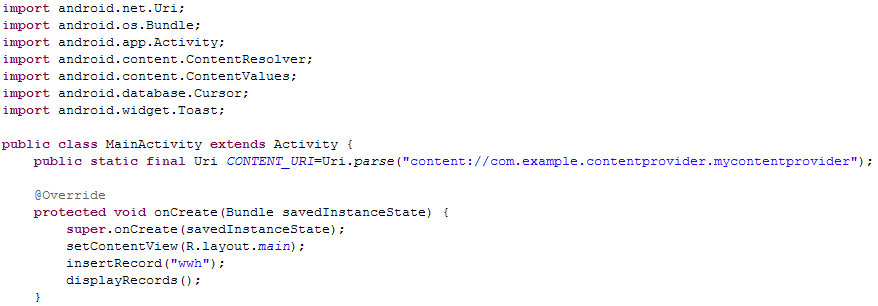
**<provider android:name=*".MyContentProvider"***

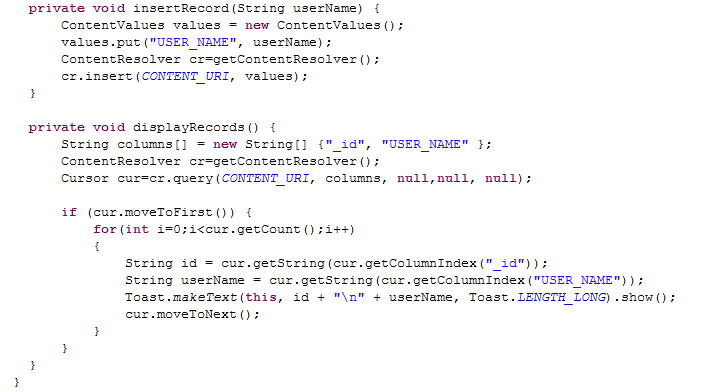
**android:authorities=*"com.example.contentprovider.mycontentprovider"*>**

**</provider>**

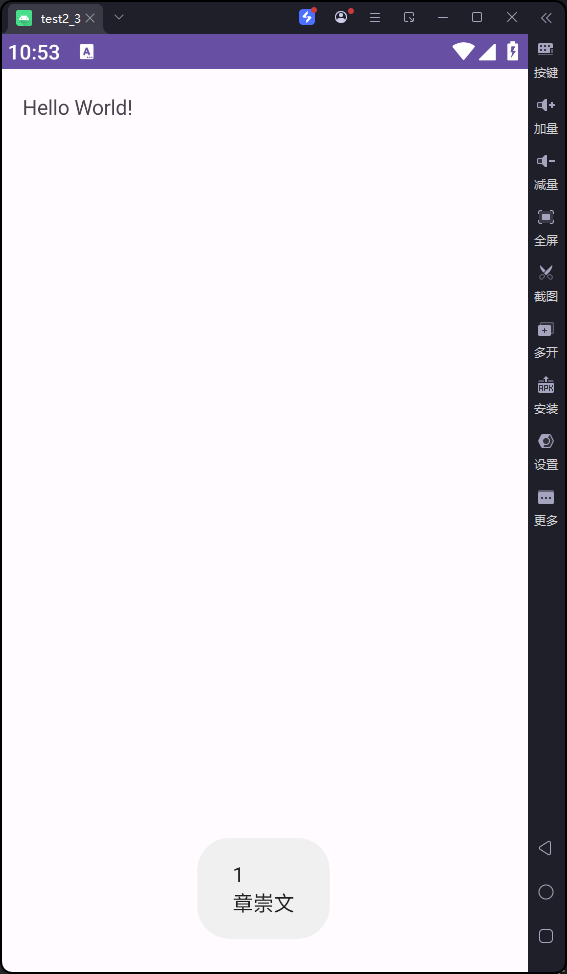
</application>

**4）在主Activity中，通过ContentResolver对象的insert(Uri uri, ContentValues values)方法来向与参数uri对应的ContentProvider对象添加数据，本质是ContentResolver对象的insert(Uri uri, ContentValues values)方法调用ContentProvider对象的insert()方法.其他的查询操作、修改操作、删除操作类似。具体代码如下：**





代码演示



四、习题

1.写入数据到SharePreferences的步骤和读取SharePreferences中的数据步骤分别是什么？

**写入数据到 SharedPreferences 的步骤**

1. **获取 SharedPreferences 对象：**
   * 使用 getSharedPreferences() 方法（适用于多个配置文件）。
   * 或者使用 getPreferences() 方法（适用于当前 Activity 的单一配置文件）。

SharedPreferences sharedPreferences = getSharedPreferences("my\_prefs", MODE\_PRIVATE);

* + "my\_prefs" 是文件名，可以自定义。
  + MODE\_PRIVATE 表示文件的访问权限，仅限应用自身访问。

1. **获取 SharedPreferences.Editor 对象：** 使用 SharedPreferences 对象调用 edit() 方法，获取 Editor 对象用于写入数据。

SharedPreferences.Editor editor = sharedPreferences.edit();

1. **写入键值对：** 通过 putXXX() 方法写入数据，例如：
   * putString(key, value)：写入字符串。
   * putInt(key, value)：写入整数。
   * putBoolean(key, value)：写入布尔值。

editor.putString("username", "JohnDoe");

editor.putInt("age", 25);

editor.putBoolean("isLoggedIn", true);

1. **提交更改：** 调用 apply() 或 commit() 方法提交更改。
   * **apply()**：异步提交，不会返回提交结果，效率较高。
   * **commit()**：同步提交，返回 boolean 表示是否提交成功。

editor.apply(); // 推荐方式

**读取 SharedPreferences 中的数据步骤**

1. **获取 SharedPreferences 对象：** 和写入数据的第一步相同，使用 getSharedPreferences() 方法获取 SharedPreferences 对象。

SharedPreferences sharedPreferences = getSharedPreferences("my\_prefs", MODE\_PRIVATE);

1. **通过键获取值：** 使用对应的 getXXX() 方法读取数据，并设置默认值：
   * getString(key, defaultValue)：读取字符串。
   * getInt(key, defaultValue)：读取整数。
   * getBoolean(key, defaultValue)：读取布尔值。

String username = sharedPreferences.getString("username", "DefaultUser");

int age = sharedPreferences.getInt("age", 0);

boolean isLoggedIn = sharedPreferences.getBoolean("isLoggedIn", false);

**完整代码示例**

**写入数据示例：**

SharedPreferences sharedPreferences = getSharedPreferences("my\_prefs", MODE\_PRIVATE);

SharedPreferences.Editor editor = sharedPreferences.edit();

editor.putString("username", "JohnDoe");

editor.putInt("age", 25);

editor.putBoolean("isLoggedIn", true);

// 提交更改

editor.apply();

**读取数据示例：**

SharedPreferences sharedPreferences = getSharedPreferences("my\_prefs", MODE\_PRIVATE);

String username = sharedPreferences.getString("username", "DefaultUser");

int age = sharedPreferences.getInt("age", 0);

boolean isLoggedIn = sharedPreferences.getBoolean("isLoggedIn", false);

// 打印数据

Log.d("SharedPreferences", "Username: " + username);

Log.d("SharedPreferences", "Age: " + age);

Log.d("SharedPreferences", "Is Logged In: " + isLoggedIn);

**注意事项：**

1. **文件名一致**：写入和读取数据时，getSharedPreferences() 的文件名必须一致。
2. **数据类型匹配**：读取数据时，getXXX() 方法的数据类型必须与写入时的 putXXX() 方法一致，否则会报错。
3. **安全性考虑**：SharedPreferences 不适合存储敏感数据（如密码），可以结合 Android Keystore 或其他加密方案。

2.添加一条记录到SQLite数据表有哪2种方法？

在 Android 的 SQLite 数据库中，添加一条记录到数据表主要有两种方法：使用 execSQL() 执行 SQL 插入语句 和 使用 insert() 方法。以下是两种方法的详细说明和代码示例：

方法 1：使用 execSQL() 执行 SQL 插入语句

步骤：

1. 获取 SQLiteDatabase 对象： 使用 SQLiteOpenHelper 提供的 getWritableDatabase() 方法获取可写的数据库对象。
2. 编写 SQL 插入语句： 使用标准的 SQL 插入语句 INSERT INTO table\_name (columns) VALUES (values)。
3. 执行插入： 使用 execSQL() 方法直接执行插入语句。

示例代码：

// 获取可写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase();

// 编写 SQL 插入语句

String sql = "INSERT INTO students (name, age, grade) VALUES ('John Doe', 18, 'A')";

// 执行插入操作

db.execSQL(sql);

// 关闭数据库（可选，视情况而定）

db.close();

优缺点：

* 优点： 操作直接，适合简单的插入。
* 缺点： 不支持动态参数绑定，容易导致 SQL 注入问题。

方法 2：使用 insert() 方法

步骤：

1. 获取 SQLiteDatabase 对象： 使用 SQLiteOpenHelper 提供的 getWritableDatabase() 方法获取可写的数据库对象。
2. 创建 ContentValues 对象： 用来存储列名和对应的值（键值对形式），代替直接编写 SQL。
3. 调用 insert() 方法：
   * insert(tableName, nullColumnHack, ContentValues)：
     + tableName：表名。
     + nullColumnHack：防止插入空值的字段名，一般传 null。
     + ContentValues：要插入的数据。

示例代码：

// 获取可写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase();

// 创建 ContentValues 对象，设置要插入的数据

ContentValues values = new ContentValues();

values.put("name", "John Doe");

values.put("age", 18);

values.put("grade", "A");

// 调用 insert() 方法插入数据

long rowId = db.insert("students", null, values);

// 检查插入是否成功

if (rowId != -1) {

Log.d("SQLite", "Record inserted successfully, Row ID: " + rowId);

} else {

Log.d("SQLite", "Failed to insert record");

}

// 关闭数据库（可选，视情况而定）

db.close();

优缺点：

* 优点：
  + 自动防止 SQL 注入，安全性更高。
  + 更灵活，尤其是在动态数据插入的情况下。
* 缺点： 相较于 execSQL() 方法，稍显繁琐。

两种方法的对比：

| 比较项 | execSQL() 方法 | insert() 方法 |
| --- | --- | --- |
| 安全性 | 容易导致 SQL 注入 | 自动防范 SQL 注入 |
| 动态性 | 不支持动态参数绑定，需手动拼接 SQL | 支持动态参数绑定，通过 ContentValues 设置 |
| 代码简洁性 | 简单直观，适合简单固定的插入操作 | 稍复杂，但更适合动态场景 |
| 推荐使用场景 | 插入操作简单且数据不包含动态内容时 | 推荐常用，安全且灵活 |

注意事项：

1. 事务控制： 在批量插入操作中，建议使用事务以提高效率：

db.beginTransaction();

try {

// 插入多条数据

db.insert(...);

db.setTransactionSuccessful(); // 提交事务

} finally {

db.endTransaction();

}

1. 数据库连接管理： 确保及时关闭 SQLiteDatabase 对象，避免内存泄漏。
2. 防止 SQL 注入： 推荐优先使用 insert() 方法，以免手动拼接 SQL 时发生安全问题。

3. SQLiteHelper的作用是什么？怎么使用它？

SQLiteHelper 的作用

SQLiteHelper 是开发 Android 应用时对 SQLite 数据库操作的一个工具类，它的正式名称是 SQLiteOpenHelper。它用于管理应用中的 SQLite 数据库，提供了以下功能：

1. 数据库的创建和升级：
   * 自动管理数据库的创建。
   * 支持数据库版本控制，便于在版本更新时修改表结构或数据。
2. 数据库的打开和关闭：
   * 简化了获取 SQLiteDatabase 对象（可读或可写数据库）的过程。
   * 管理数据库连接，避免资源泄漏。
3. 封装了复杂的数据库操作：
   * 提供了回调方法（如 onCreate() 和 onUpgrade()）以定制数据库的初始化和升级逻辑。

SQLiteOpenHelper 的常见方法

* 构造函数： 用于初始化 SQLiteOpenHelper 的实例。

java

复制代码

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

* + context：上下文对象。
  + name：数据库文件名。
  + factory：游标工厂，通常传 null。
  + version：数据库版本号，用于管理数据库升级。
* 回调方法：
  + onCreate(SQLiteDatabase db)：首次创建数据库时调用，用于初始化表结构。
  + onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)：当数据库版本号变化时调用，用于数据库升级。
  + （可选）onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)：用于处理降级操作。
* 数据库访问方法：
  + getReadableDatabase()：获取可读数据库。
  + getWritableDatabase()：获取可写数据库。

如何使用 SQLiteOpenHelper

步骤：

1. 创建一个自定义类继承 SQLiteOpenHelper： 实现 onCreate() 和 onUpgrade() 方法。
2. 在 onCreate() 方法中定义数据库表结构： 使用 CREATE TABLE 语句创建数据表。
3. 在 onUpgrade() 方法中定义数据库升级逻辑： 使用 ALTER TABLE 或 DROP TABLE 等 SQL 修改表结构。
4. 实例化自定义的 SQLiteHelper 类，调用 getWritableDatabase() 或 getReadableDatabase() 方法获取数据库对象。

4. ContentProvider的作用是什么？如何使用它？

ContentProvider 的作用

ContentProvider 是 Android 提供的一个组件，用于在不同应用之间安全地共享数据。它的作用主要包括：

1. 跨应用数据共享：
   * ContentProvider 允许一个应用将自己的数据（如 SQLite 数据库、文件或其他数据源）共享给其他应用。
   * 通过统一的 URI 机制（content:// 开头的路径）来访问数据，避免直接操作底层数据源。
2. 数据的抽象和封装：
   * 提供了一套标准的 CRUD 接口（增删改查），隐藏了具体的存储实现（如数据库、文件等）。
3. 统一的访问方式：
   * 通过 ContentResolver 访问数据，避免了调用应用内部 API 的复杂性。

ContentProvider 的常见使用场景

1. 系统数据共享：
   * 系统中很多核心功能依赖于 ContentProvider，例如访问联系人、短信、媒体文件（图片、视频等）。
     + 示例：访问联系人数据的 content://contacts/people。
2. 应用间数据共享：
   * 当一个应用需要将数据暴露给其他应用时，可以通过自定义 ContentProvider 实现安全的数据共享。
3. 模块化开发：
   * 在一个应用的多个模块中，通过 ContentProvider 来组织和管理数据访问。

ContentProvider 的核心方法

当你创建一个 ContentProvider 时，需要实现以下几个方法来操作数据源：

| 方法 | 作用 |
| --- | --- |
| onCreate() | 初始化 ContentProvider（如创建或打开数据库）。 |
| query() | 查询数据，返回 Cursor 对象。 |
| insert(Uri uri, ContentValues values) | 插入数据，返回新数据的 Uri。 |
| update(Uri uri, ContentValues values, String selection, String[] selectionArgs) | 更新数据，返回更新的行数。 |
| delete(Uri uri, String selection, String[] selectionArgs) | 删除数据，返回删除的行数。 |
| getType(Uri uri) | 返回指定 URI 对应的数据类型，用于 MIME 类型声明（如单条记录或多条记录）。 |

如何使用 ContentProvider

1. 创建自定义 ContentProvider

实现步骤：

1. 创建一个类继承 ContentProvider： 重写其核心方法（onCreate、query、insert 等）。
2. 注册到 AndroidManifest.xml 文件： 声明 ContentProvider，并定义权限及访问路径。
3. 定义 URI 和操作接口：
   * URI 用于标识共享数据的路径（通常通过 UriMatcher 来解析 URI）。
   * CRUD 接口实现对底层数据（如 SQLite）的操作。

五、实验小结

**1 编程遇到的问题及解决方案**

1. 属性和概念理解问题

1. 问题：不理解 ContentProvider 中 URI 的用途及作用
   * 现象： 在实现 query() 方法时，不清楚如何通过 URI 区分不同的数据操作（如操作全表或单行记录）。
   * 解决方案：
     + 学习并使用 UriMatcher 来解析 URI，并对不同的 URI 模式定义常量（如 STUDENTS 和 STUDENT\_ID）。
     + 确保在每个操作（如 query() 或 delete()）中都正确使用 UriMatcher 来判断 URI 的类型。
2. 问题：对 getType() 方法的 MIME 类型不理解
   * 现象： getType() 方法返回错误或不符合标准，导致外部应用无法识别数据类型。
   * 解决方案：
     + 学习 Android MIME 类型规则：
       - vnd.android.cursor.dir/ 表示一组数据（如表中的所有行）。
       - vnd.android.cursor.item/ 表示单个数据（如表中的一行）。
     + 根据 URI 模式，分别返回对应的 MIME 类型字符串。

2. 数据库相关问题

1. 问题：SQLite 数据库操作时表名或列名拼写错误
   * 现象： 调用 insert()、query() 等方法时，报错提示 "no such table" 或 "no such column"。
   * 解决方案：
     + 确保表名、列名定义一致，将常量（如 TABLE\_NAME 和列名）统一管理，避免硬编码。
     + 调试时，通过 adb 工具查看数据库结构，确认表是否正确创建：

adb shell

sqlite3 /data/data/com.example.myapp/databases/MyDatabase.db

1. 问题：onUpgrade() 方法未正确实现导致数据丢失
   * 现象： 升级应用后，数据库被清空。
   * 解决方案：
     + 学习 onUpgrade() 的工作原理。不要直接删除旧表（DROP TABLE），而是通过 ALTER TABLE 修改表结构，或将旧表数据备份后迁移到新表。

3. ContentProvider 调试过程中的问题

1. 问题：query() 方法返回的 Cursor 数据异常
   * 现象： 查询返回的 Cursor 无法正常遍历，或者返回空结果。
   * 解决方案：
     + 检查查询语句是否正确，如表名和查询条件是否匹配。
     + 在 query() 方法中打印调试日志，输出 SQL 查询语句和参数：

Log.d("Query", "SQL Query: " + selection + ", Args: " + Arrays.toString(selectionArgs));

* + - 验证是否正确设置了 Cursor.setNotificationUri()，确保数据变化时能够通知 UI。

1. 问题：插入数据后，URI 返回错误
   * 现象： 调用 insert() 后，返回的 URI 不包含新插入行的 ID。
   * 解决方案：
     + 使用 ContentUris.withAppendedId() 方法构建包含 ID 的返回 URI：

Uri resultUri = ContentUris.withAppendedId(CONTENT\_URI, id);

1. 问题：ContentProvider 的权限设置错误
   * 现象： 外部应用无法访问 ContentProvider，提示权限不足。
   * 解决方案：
     + 在 AndroidManifest.xml 中检查 android:exported 属性是否正确设置为 true。
     + 如果需要控制访问权限，为 ContentProvider 添加权限声明：

<provider

android:name=".MyContentProvider"

android:authorities="com.example.myapp.provider"

android:exported="true"

android:permission="com.example.myapp.READ\_WRITE\_PERMISSION" />

4. 调用 ContentProvider 时的问题

1. 问题：调用 ContentResolver 时 URI 拼写错误
   * 现象： 使用 getContentResolver().query() 调用时，报错 "Unknown URI"。
   * 解决方案：
     + 确保调用时使用的 URI 与 ContentProvider 中定义的 URI 匹配（包括 AUTHORITY 和路径）。
     + 使用 Uri.parse() 时避免多余的 / 或拼接错误。
2. 问题：外部应用使用 ContentProvider 时数据类型不匹配
   * 现象： 插入数据或查询数据时，提示类型不匹配（如 int 写入 TEXT 列）。
   * 解决方案：
     + 外部应用在使用 ContentValues 时，确保数据类型与数据库定义一致。
     + 调试时，打印 ContentValues 内容，验证写入的数据。

5. 数据通知机制的问题

1. 问题：UI 未能及时更新
   * 现象： 插入、更新或删除数据后，UI 数据未刷新。
   * 解决方案：
     + 确保在每个操作（如 insert()、delete()）后调用 getContext().getContentResolver().notifyChange(uri, null)。
     + 在 Activity 或 Fragment 中，使用 ContentObserver 监听数据变化：

getContentResolver().registerContentObserver(MyContentProvider.CONTENT\_URI, true, new ContentObserver(new Handler()) {

@Override

public void onChange(boolean selfChange) {

// 数据发生变化，刷新 UI

}

});

**2 实验收获及体会**

实验收获及体会

1. 掌握了 ContentProvider 的作用和实现方式
   * 理解了 ContentProvider 在应用间数据共享中的重要作用。
   * 熟悉了 ContentProvider 的核心方法（如 query()、insert()、delete() 等）及其实现流程。
2. 学习了 URI 和 UriMatcher 的使用
   * 知道了如何使用 UriMatcher 匹配不同的 URI 来执行针对性的数据库操作。
   * 理解了 URI 的结构（content://authority/path）及其对数据访问的作用。
3. 掌握了 SQLite 数据库在 ContentProvider 中的应用
   * 学习了通过 SQLiteHelper 管理数据库的创建、升级以及表结构定义。
   * 理解了如何在 ContentProvider 中结合 SQLite 实现数据的增删改查操作。
4. 理解了数据共享中的安全性和权限控制
   * 学会了通过在 AndroidManifest.xml 中配置权限和 android:exported 属性来管理数据共享的访问权限。
   * 知道了如何为 ContentProvider 添加自定义权限，保护数据安全。
5. 掌握了数据通知机制
   * 理解了 ContentResolver.notifyChange() 的作用，以及如何通过 ContentObserver 实现对数据变化的监听和 UI 自动刷新。
6. 增强了调试能力
   * 学会了使用日志 (Log) 调试 ContentProvider 中的数据操作流程。
   * 熟悉了通过 adb 命令调试 SQLite 数据库，验证表结构和数据完整性。
7. 掌握了 ContentResolver 的使用
   * 学会了如何通过 ContentResolver 调用 ContentProvider 提供的接口，完成数据的增删改查。
8. 提高了对组件化开发的理解
   * 理解了 ContentProvider 在模块化开发中的价值，通过它实现组件间的松耦合和数据共享。

体会

* ContentProvider 是 Android 中非常重要的组件，尤其在数据共享和模块化开发中具有关键作用。通过实验，深刻体会到其抽象性和标准化接口带来的便利。
* 实验过程中，一些调试问题让我意识到基础概念的重要性，例如 URI 结构和 MIME 类型的理解。
* 学到了处理数据共享时需要关注安全性和兼容性，未来在实际开发中可以更高效地设计数据接口。

**3 不足之处及下一步需改进的地方**

 对 ContentProvider 的理解不够深入

* 在刚开始实验时，对 URI 和 MIME 类型的概念理解不透彻，导致方法实现时遇到困惑。
* 改进方向： 多阅读官方文档，研究系统提供的 ContentProvider（如联系人、媒体数据）源码，加深对机制的理解。

 调试能力不足

* 在调试 ContentProvider 时，未能及时通过日志或工具发现问题，例如 SQL 查询语句的错误未及时暴露。
* 改进方向： 增强调试意识，及时添加日志输出，并熟练掌握 adb 工具对数据库和 URI 进行排查。

 数据权限控制不够完善

* 在实验中，对权限的设置较为宽松，没有对外部访问进行足够的限制，容易导致数据泄露风险。
* 改进方向： 进一步学习权限管理，为 ContentProvider 添加更细粒度的权限控制。

 缺乏对数据通知机制的全面实践

* 对 ContentObserver 的使用还不熟悉，实验中未全面实现 UI 自动更新的功能。
* 改进方向： 在下一步实验中，完善数据变化监听机制，学习使用 ContentObserver 和 LiveData 实现 UI 实时刷新。

 实验覆盖面不足

* 实验主要关注了 SQLite 数据库结合 ContentProvider 的实现，忽略了其他数据源（如文件、网络数据）的共享方式。
* 改进方向： 尝试实现文件和其他非结构化数据的 ContentProvider，拓展对数据共享机制的认识。

 对性能优化关注较少

* 未考虑大量数据操作时可能带来的性能问题（如分页查询、大量插入等）。
* 改进方向： 学习如何优化 ContentProvider 的性能，例如通过异步查询（AsyncQueryHandler）或批量插入提高效率。

 对多应用交互测试不充分

* 实验中主要在同一应用内调用 ContentProvider，缺乏跨应用数据访问的实际测试。
* 改进方向： 创建另一个测试应用程序，实际测试数据共享功能，并分析潜在的兼容性问题。

 对错误处理的关注不够

* 实验中对异常（如 URI 匹配失败、数据库操作失败等）的处理较为简单，未能提供详细的错误信息。
* 改进方向： 在方法实现中补充更多异常处理逻辑，并通过日志记录错误详情，提升代码的健壮性。