Android第八次课

写在最前面

快乐的时光总是这么短暂，在网校学习的日子，充实而具有挑战。我相信，这一学期，无论是Java还是Android，大家相比同阶段的其他人，都成长了许多。毕竟跳出舒适圈开始是兴奋的，中间是痛苦的，最后才是享受的。

本次课程的安排

- 复习前面所讲的一些东西
- Android中数据库的使用
- Android中动态权限

RecyclerView

- 添加依赖
- 在布局中设置
- 构建RecyclerView的item布局(注意item布局中的高度问题)
- 自己实现的Adapter继承`RecyclerView.Adapter`并传入泛型（需要传入一个`RecyclerView.ViewHolder`或其子类的类型）ViewHolder是一个内部类。里面有每个itemview的控件的实例
- 重写onCreateViewHolder方法，返回一个真正inflate好item布局的viewHolder
- 重写onBindViewHolder方法，对每一项holder的控件进行赋值
- 重写getItemCount，返回所有item的个数

运行时权限

- 用户不需要在安装软件的时候一次性授权所有申请的权限，而是可以在软件的使用过程中再对某一项权限申请进行授权。
- 分类：普通权限和**危险权限**
- 完整的权限列表：https://blog.csdn.net/lianyi68/article/details/78588565
- 我们在进行运行时权限处理时使用的是**权限名**，但是用户一旦同意授权了，那么该权限所对应的**权限组**中所有的其他权限也会同时被授权。
- 在Android6.0及以上的系统在使用**危险权限**的时候必须进行运行时权限处理，就是动态处理

以下权限都需要在运行时判断：

```
身体传感器
日历
摄像头
通讯录
地理位置
麦克风
电话
短信
存储空间
```

在AndroidManifest.xml中声明权限

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.thinkpad.runtimepermissiontest">
    <!--声明打电话的权限-->
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

在主逻辑中动态申请权限

```
if (ContextCompat.checkSelfPermission(MainActivity.this,Manifest
                        .permission.CALL_PHONE)!=
PackageManager.PERMISSION_GRANTED){
                    //调用ActivityCompat.requestPermissions(...)方法来向用户申请权限
                    //参数一：Activity的实例；参数二：String数组-存放申请的权限名；参数
三：请求码
                    ActivityCompat.requestPermissions(MainActivity.this,
                            new String[]{Manifest.permission.CALL_PHONE},1);
```

申请完权限后的回调函数

授权的结果会封装在grantResults参数中。

```
//不论用户同意或者拒绝授权，都会回调onRequestPermissionsResult(...)方法
    //授权的结果封装在grantResults中
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        switch(requestCode){
            case 1:
                if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED){
                    call();
                }else{
                    Toast.makeText(this,"这是你选的嘛，偶
像",Toast.LENGTH_SHORT).show();
                }
```

```
                break;
            default:
        }
    }
```

数据的存储

SharedPreferences存储

> SharedPreferences储存是使用键值对的方式来储存数据的，储存时每一个数据对应一个键，读取时通过键读取值。

- **写文件**

```
private void save() {
        String data = editText.getText().toString();
        //第一步：获取Sharedpreferences对象
        SharedPreferences sharedPreferences =
getSharedPreferences("data",MODE_PRIVATE);
        //第二步：获取SharedPreferences.Editor对象
        SharedPreferences.Editor editor = sharedPreferences.edit();
        //第三步，添加数据
        editor.putString("data",data);
        //第四步：提交
        editor.apply();
    }
```

若要查看文件保存的地方，在打开虚拟机时点击右下角竖着的Device File Explorer,找到/data/data/包名/shared_prefs,我们写的文件就储存在这里

- **读文件**

```
  private String Load() {
        //第一步：创建SharedPreferences 类型对象
        SharedPreferences sharedPreferences =
getSharedPreferences("data",MODE_PRIVATE);
        //第二步：获取
        String data = sharedPreferences.getString("data","");
        return data;
    }
```

SQLite数据库存储（需要掌握一点点SQL语句，先可以模仿。）

Android为了能更好更方便的管理数据库，专门提供了一个SQLiteOpenHelper帮助类，借助这个类可完成对数据库的创建和升级。

- **创建自己的帮助类**

```
public class MySQLiteOPenHelper extends SQLiteOpenHelper {
    //写建表语句
    public static final String CREATR_BOOK ="create table Book(" +
            "id integer primary key autoincrement,"
  //将id设为主键，并用autoincrement关键字表示id列自增长
            + "author text,"    // text文本类型
            + "price real,"     //real 浮点型
            + "pages integer,"  //integer 整型
            + "name text)";     //blob 二进制型
    public static final String CREATE_CATEFORY = "create table Category("+
            "id integer primary key autoincrement,"
            +"name text,"
            +"code integer)";

    private Context context;
    //有两个构造方法可供重写，一般选择参数少的即可
    //参数 第一个:上下文，第二个：数据库名，
   //第三个：允许我们在查询数据时返回自定义Cursor,一般传入null
   //第四个：当前的版本号
    public MySQLiteOPenHelper(@Nullable Context context, @Nullable String name,
@Nullable SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
        this.context = context;
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL(CREATR_BOOK);
        //这个方法用于创建表
        sqLiteDatabase.execSQL(CREATE_CATEFORY);
        Toast.makeText(context,"create success",Toast.LENGTH_SHORT).show();
    }

    @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);
    }
}
```

- **实例化帮助类**

```
MySQLiteOPenHelper sqLiteOPenHelper = new
   MySQLiteOPenHelper(this,"BookStore.db",null,1);
```

- **创建数据库**

```
//这两个方法都表示创建或打开一个现有的数据库
sqLiteOPenHelper.getWritableDatabase();
//sqLiteOPenHelper.getReadableDatabase()
```

- **添加数据**

  添加数据主要调用SQLiteDatabase的insert方法，此方法接收三个参数，

  第一个是表名

  第二个参数用于在未指定添加数据的情况下给某些可为空的列自动赋值null,一般用不到这个方法，直接传入null，

  第三个参数是ContentValues对象，用于存放添加的数据

```
//获取SQLiteDatabase对象
SQLiteDatabase db = sqLiteOPenHelper.getReadableDatabase();
//获取ContentValues对象，存放数据
ContentValues values = new ContentValues();
values.put("name","123");
values.put("pages",234);
values.put("price",16);
values.put("author","Day");
db.insert("Book",null,values);
values.clear(); //如果要再次组装数据，需要把values清空
values.put("name","aaa");
values.put("pages",12);
values.put("price",230);
values.put("author","god");
db.insert("Book",null,values);
```

- **更新数据**

  更新数据主要调用SQLiteDatabase的update方法，此方法接收四个参数，

  第一个是表名

  第二个参数是ContentValues对象，用于存放更新的数据

  第三，四个参数是用于约束更新某一行或某几行的数据，不指定的话默认更新所有行

```
SQLiteDatabase sqLiteDatabase = sqLiteOPenHelper.getReadableDatabase();

ContentValues values1 = new ContentValues();
values1.put("price",29);

sqLiteDatabase.update("Book",values1,"name = ?",
  new String[]{"123"});
```

- **删除数据**

  删除数据主要调用SQLiteDatabase的delete方法，此方法接收三个参数，

  第一个是表名

  第二，三个参数是用于约束删除某一行或某几行的数据，不指定的话默认更新所有行

```
SQLiteDatabase sqLiteDatabase1 = sqLiteOPenHelper.getReadableDatabase();
sqLiteDatabase1.delete("Book","name = ?",new String[]{"bbb"});
```

- **查询数据**

  查询数据主要调用SQLiteDatabase的query方法，此方法最短的重载接收7个参数

query()方法参数	描述
table	指定查询的表名
columns	指定查询的列名
selection	指定where的约束条件
selectionArgs	为where中的占位符提供具体的值
groupBy	指定需要group by的列
having	对group by后的结果进一步约束
orderBy	指定查询结果的排序方法

我们在使用时不必为每条语句都指定所有的参数，此方法返回一个CurSor对象，查询到的数据可从这个对象中取出

```
SQLiteDatabase sqLiteDatabase2 = sqLiteOPenHelper.getReadableDatabase();
//查询所有数据
Cursor cursor = sqLiteDatabase2.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{
    //通过Cursor的getColumnIndex()方法获取到某一列在表中对应位置的索引
        String name = cursor.getString(cursor.getColumnIndex("name"));
        String author = cursor.getString(cursor.getColumnIndex("author"));
        String price = cursor.getString(cursor.getColumnIndex("price"));
        Log.d("MAinActivity",name+"     "+author+"      "+price);
       }while (cursor.moveToNext());
}
cursor.close();
```

- **升级数据库**

```
sqLiteOPenHelper = new
  MySQLiteOPenHelper(this,"BookStore.db",null,2);
```

把创建类实例时的版本号加1（版本号加一后会回调onUpgrade()方法）

```
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);
    }
```

在onUpgrade()中通过DROP语句删除原来存在的表，并调用onCreate()方法（如果直接调用onCreate()方法，原来表存在，在创建时直接报错）

具体的实例Demo讲解