目录

[6.0后读写权限 1](#_Toc141820360)

[NDK 1](#_Toc141820361)

[安装AS环境还要装JDK 1](#_Toc141820362)

[如何通过回调接口在MVVM中实现数据更新 2](#_Toc141820363)

[解决方案： 3](#_Toc141820364)

[在adapter中排序 3](#_Toc141820365)

[对map进行遍历有三种方法 3](#_Toc141820366)

[回调方法实现 3](#_Toc141820367)

[只有Activity能创建一个窗体，所以dialog的context要传入activity 10](#_Toc141820368)

[String格式化显示 10](#_Toc141820369)

[editText.getText()返回值是editable而不是String 11](#_Toc141820370)

[安卓实现隐藏当前页面 11](#_Toc141820371)

[自定义弹窗内容 11](#_Toc141820372)

[文件读写 12](#_Toc141820373)

[实现计时器 12](#_Toc141820374)

[获取系统时间 13](#_Toc141820375)

[EditText的属性 14](#_Toc141820376)

[在有莫名其妙功能实现不了的时候检查是否有内存泄漏 14](#_Toc141820377)

# 6.0后读写权限

安卓6.0以上开始读写权限需要动态申请

# NDK

NDK原生工具包，支持c++等编译

JDK、SDK、NDK

# 安装AS环境还要装JDK

# 如何通过回调接口在MVVM中实现数据更新

1. 在View层定义一个接口，包含回调方法

public interface ViewModelCallback {

void onCallback();

}

1. 在ViewModel中创建一个方法来设置回调接口的实例，并且接受一个接口实例作为参数。

private ViewModelCallback callback;

public void setCallback(ViewModelCallback callback) {

this.callback = callback;

}

3、在ViewModel类中的适当位置，执行需要回调的操作，并在适当的时候调用回调接口中的方法。

public void doSomething() {

// 执行需要回调的操作

// 在适当的时候调用回调方法

if (callback != null) {

callback.onCallback();

}

}

4. 在View层中，实现回调接口，并将其实例传递给ViewModel对象。

viewModel.setCallback(this);

5、在View层中，根据需要调用ViewModel对象的回调方法，以执行相应的操作

// 调用ViewModel中的方法

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_my);

//…//

viewModel.doSomething();

}

@Override

public void onCallback() {

// 处理回调操作

}

# 解决方案：

在ViewModel实现回调方法更新数据。

定义一个属性记录是否更新数据，如果还没更新就使用默认数据，如果更新了再使用新的数据。

# 在adapter中排序

想在adapter中使用排序的话可以在onBindViewHolder中将数据进行排序之后再输入position绑定

# 对map进行遍历有三种方法

1、使用list创建keyset之后利用key来遍历（推荐）

1. 使用list<map.entry<kay, value>>创建entry数组进行遍历（推荐）
2. 使用遍历器interator

在观察数据源更新的情况中去设置数据到adapter内有可能会导致adapter内外的数据不一致出现IndexOutOfBoundsException的错误，

好像是安卓原生的错误，

解决方法：

1、自定义包装类manager重写layoutChild方法，捕获异常

<https://blog.csdn.net/xiaoshuxgh/article/details/80225516>

2、使用自定义的manager

# 回调方法实现

1. 定义一个回调接口和需要完成的函数（可以是类也可以定义在后面的同个类中）
2. 定义一个类持有接口对象、设置接口对象set方法、创建方法m调用接口内的方法。
3. 定义另一个类持有前面类的实例并调用他的set方法传入接口实例、实现接口方法、并在合适位置调用m

Tips：可以使用匿名类实现

1. 定义B类实现接口和方法传递数据并持有A类将自己设置为A类的callBack并调用A类方法（告诉A类需要方法调用我）

定义接口(public interfact)，预定义需要完成的函数

定义A类，使用接口和该接口中的抽象方法，同时在初始化A时请求函数B作为callback的参数

定义B类(implements interface)，设定函数的具体动作，并将B当做参数传给A，使A在回调时可以找到函数的位置

在 Android 中，回调方法通常通过接口（Interface）来实现。以下是一个简单的示例，演示了如何使用回调方法来实现在 Fragment 或 Activity 中处理点击事件的功能：

// 定义一个回调接口

public interface OnButtonClickListener {

void onButtonClick();

}

// 在 Fragment 或 Activity 中实现该接口

public class MyFragment extends Fragment {

private OnButtonClickListener onButtonClickListener;

// 设置回调监听器

public void setOnButtonClickListener(OnButtonClickListener listener) {

this.onButtonClickListener = listener;

}

// 模拟按钮点击事件

private void simulateButtonClick() {

// 检查是否设置了回调监听器

if (onButtonClickListener != null) {

// 调用回调方法

onButtonClickListener.onButtonClick();

}

}

// 其他代码...

}

// 在另一个 Fragment 或 Activity 中实现回调方法

public class OtherFragment extends Fragment {

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

// 创建 MyFragment 实例

MyFragment myFragment = new MyFragment();

// 设置回调监听器

myFragment.setOnButtonClickListener(new OnButtonClickListener() {

@Override

public void onButtonClick() {

// 处理按钮点击事件

// 在这里实现你需要的逻辑

}

});

// 其他代码...

}

// 其他代码...

}

在上述示例中，我们首先定义了一个回调接口 OnButtonClickListener，其中声明了一个 onButtonClick() 方法用于处理按钮点击事件。在 MyFragment 类中，我们使用 setOnButtonClickListener() 方法来设置回调监听器，并在 simulateButtonClick() 方法中检查监听器是否被设置，然后调用回调方法。

在另一个 Fragment 或 Activity 中（例如 OtherFragment），我们创建 MyFragment 实例，然后设置回调监听器为匿名内部类，实现了 onButtonClick() 方法来处理按钮点击事件的逻辑。

请根据你的具体需求进行调整，以实现你想要的回调功能。

非常抱歉，对于之前提供的示例中的 simulateButtonClick() 方法没有给出完整的解释。这个方法是用来模拟按钮点击事件的，目的是演示在触发某个事件时如何调用回调方法。

在实际使用中，你可以将 simulateButtonClick() 方法放在适当的时机来触发回调方法的调用。例如，在 Fragment 或 Activity 中的某个按钮被点击时，你可以调用 simulateButtonClick() 方法来模拟按钮点击事件，进而触发回调方法的执行。

以下是一个更新后的示例代码，演示了如何在按钮点击时触发回调方法的调用：

public class MyFragment extends Fragment {

private OnButtonClickListener onButtonClickListener;

// 设置回调监听器

public void setOnButtonClickListener(OnButtonClickListener listener) {

this.onButtonClickListener = listener;

}

// 模拟按钮点击事件

private void simulateButtonClick() {

// 检查是否设置了回调监听器

if (onButtonClickListener != null) {

// 调用回调方法

onButtonClickListener.onButtonClick();

}

}

// 其他代码...

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

// 假设有一个按钮

Button button = view.findViewById(R.id.button);

// 设置按钮点击事件监听器

button.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

// 按钮被点击时，调用模拟按钮点击事件的方法

simulateButtonClick();

}

});

// 其他代码...

}

}

在上述示例中，我们在 MyFragment 的 onViewCreated() 方法中，给按钮设置了一个点击事件的监听器。当按钮被点击时，会调用 simulateButtonClick() 方法，并触发回调方法的执行。

请注意，在实际使用中，你可以根据自己的需求和业务逻辑，将 simulateButtonClick() 方法放在合适的地方来触发回调方法的调用，而不仅仅是在一个模拟的按钮点击事件中。

你的意思是在这个点击事件内调用这个方法之后在点击的时候会触发另一个Fragment或Activity内的回调方法吗？

是的，你理解得很对。

在上述示例中，当按钮被点击时，在 simulateButtonClick() 方法中触发了回调方法 onButtonClick() 的执行。

假设你在其他的 Fragment 或 Activity 中实现了 onButtonClick() 方法，并将该方法作为回调监听器传递给 MyFragment 的实例，那么在按钮点击时，就会触发其他 Fragment 或 Activity 中的回调方法的执行。

以下是一个示例代码，演示了如何在其他 Fragment 中实现回调方法：

public class OtherFragment extends Fragment implements OnButtonClickListener {

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

// 创建 MyFragment 实例

MyFragment myFragment = new MyFragment();

// 设置回调监听器

myFragment.setOnButtonClickListener(this);

// 其他代码...

}

// 实现回调方法

@Override

public void onButtonClick() {

// 在这里实现回调方法的逻辑

// 当按钮被点击时，该方法会被调用

}

// 其他代码...

}

在上述示例中，OtherFragment 实现了 OnButtonClickListener 接口，并重写了回调方法 onButtonClick()。然后，在 OtherFragment 的 onViewCreated() 方法中，创建了 MyFragment 实例，并将 OtherFragment 自身作为回调监听器传递给 MyFragment。

当在 MyFragment 中的按钮被点击时，会调用 simulateButtonClick() 方法，触发回调方法的执行。这时，OtherFragment 中实现的 onButtonClick() 方法会被调用。

你可以根据自己的业务逻辑，在实际使用中实现回调方法，并在合适的地方设置回调监听器。这样，当按钮点击时，就可以触发其他 Fragment 或 Activity 中的回调方法的执行。

那么这样做有什么实际的作用吗，或者说能拿来干什么呢？

在 Android 开发中，使用回调方法可以实现各种功能和交互。下面列举了一些常见的用途：

1. 事件处理：你可以使用回调方法来处理用户交互事件，例如按钮点击、列表项点击、滑动手势等。当事件发生时，回调方法会被触发，让你能够执行相应的逻辑操作。

2. 解耦组件：通过回调方法，你可以将一些功能和逻辑从一个组件中分离出来，使组件之间保持解耦。例如，一个 Fragment 可以通过回调方法将某个事件通知给其所属的 Activity 或其他 Fragment，从而达到不同组件之间的通信和交互。

3. 定制化逻辑：回调方法可以让你自定义组件的逻辑行为。通过设置回调监听器，并在回调方法中实现你所需的功能，你可以根据实际需求扩展和定制组件的行为。

4. 异步操作：回调方法可以用于异步操作的通知和结果处理。例如，在网络请求的回调方法中处理请求的响应数据，或者在后台任务完成时通知主线程。

5. 插件和扩展：通过提供回调接口，你可以允许其他开发者创建插件或扩展你的应用程序。他们可以实现回调方法来自定义你的应用的行为和功能。

总之，回调方法为你提供了一种有效的方式来实现模块之间的通信、处理用户交互、自定义逻辑和扩展应用功能。通过合理地使用回调方法，你可以增强应用的灵活性、可维护性和可扩展性。

# 只有Activity能创建一个窗体，所以dialog的context要传入activity

# String格式化显示

String.format(Locale.getDefault(), "%02d小时%02d分钟%02d秒", year, month, day);

Locale.getDefault()

%02d小时%02d分钟%02d秒作为格式字符串。这个格式字符串中包含了三个格式化占位符 %02d，每个占位符表示一个整数值。这些占位符的含义如下：

%：格式说明符的起始标识。

0：表示在数字不够两位数时，用0进行填充。

2：表示最终显示的数字总宽度为2位数。

d：表示将参数视为十进制整数。

Locale.getDefault()用于获取当前设备的默认地区设置，它作为String.format方法的第一个参数。根据地区不同，时间单位的显示方式可能也有所差异。使用Locale.getDefault()可以保证在不同的地区中，格式化结果的显示方式是符合当地习惯的

# editText.getText()返回值是editable而不是String

当需要在Fragment和基类BaseActivity中得到相同的viewModelProvider时，可以在onAttach方法中获取BaseActivity的实例并调用getViewModelProvider方法。

Tips：如果直接调用requireActivity是无法调用getViewModelProvider的，我也不知道为什么。

最后是这样实现的

@Override

public void onAttach(@NonNull Context context) {

super.onAttach(context);

if (context instanceof BaseActivity) {

BaseActivity baseActivity = (BaseActivity) context;

viewModelProvider = baseActivity.getViewModelProvider();

} else {

viewModelProvider = new ViewModelProvider(requireActivity());

}

}

# 安卓实现隐藏当前页面

singleTask单例创建activity+moveTaskToBack(true);

如果是false则只有在根页面才有效

# 自定义弹窗内容

Dialog dialog = new Dialog(this);

dialog.setContentView(R.layout.point\_dialog\_over\_view);

Tips：

隐藏自定义标题：

dialog.requestWindowFeature(Window.FEATURE\_NO\_TITLE);

获取dialog中的控件进行自定义：

Button btnYes = dialog.findViewById(R.id.button\_yes);

btnYes.(view -> {

dialog.dismiss(); //隐藏dialog

});

# 文件读写

记得用线程去运行：

new Thread(() -> {

//......

});

创建File文件：

外部存储路径Environment.getExternalStorageDirectory()

内部存储路径Context.getFilesDir()

应用的外部存储文件目录Context.getExternalFilesDir()

应用的外部存储缓存目录Context.getExternalCacheDir()

File externalDir = new File(Environment.getExternalStorageDirectory(), "数据采集2/点记录");

if (!externalDir.exists()) {

boolean mkdirsResult = externalDir.mkdirs(); // 创建目录（如果不存在）

if (!mkdirsResult) {

// 目录创建失败

Toast.makeText(this, "目录创建失败", Toast.LENGTH\_SHORT).show();

return; // 结束处理

}

}

创建FileWriter对象：

FileWriter writer = new FileWriter(file);

writer.write(content);

writer.close();

# 实现计时器

private Runnable timerRunnable = new Runnable() {

@Override

public void run() {

timerSeconds++;

int hours = timerSeconds / 3600;

int minutes = (timerSeconds % 3600) / 60;

int seconds = timerSeconds % 60;

String timeText = String.format(Locale.getDefault(), "%02d小时%02d分钟%02d秒", hours, minutes, seconds);

viewBinding.tvRecordTime.setText("记录时间：" + timeText);

handler.postDelayed(this, 1000); // 延迟1秒后再次运行

}

};

在onCreate中启动

handler.postDelayed(timerRunnable, 1000); // 延迟1秒后开始运行

# 获取系统时间

Android API 26（Android 8.0）及更高版本：

使用LocalDateTime获取时间日期：

LocalDateTime currentDateTime = null;

int year = 0;

int month = 0;

int day = 0;

int hour = 0;

int minute = 0;

if (android.os.Build.VERSION.SDK\_INT >= android.os.Build.VERSION\_CODES.O) {

currentDateTime = LocalDateTime.now();

year = currentDateTime.getYear();

month = currentDateTime.getMonthValue();

day = currentDateTime.getDayOfMonth();

hour = currentDateTime.getHour();

minute = currentDateTime.getMinute();

}

旧版本：

使用Calendar获取时间日期：

// Calendar calendar = Calendar.getInstance();

// int year = calendar.get(Calendar.YEAR);

// int month = calendar.get(Calendar.MONTH);

// int day = calendar.get(Calendar.DAY\_OF\_MONTH);

// int hour = calendar.get(Calendar.HOUR);

// int minute = calendar.get(Calendar.MINUTE);

# EditText的属性

**android:inputType**

可以控制用户在输入框中输入的内容

text：标准文本输入，可以包含任何字符。

textCapSentences：自动将句子的首字母大写。

textEmailAddress：电子邮件地址输入类型。

textPassword：密码输入类型，字符显示为隐藏的圆点或星号。

number：数字输入。

phone：电话号码输入。

datetime：日期和时间输入。

time：时间输入。

**android:inputType**

只是指示预期的输入类型，并不能完全限制用户的输入。输入框中的文本可以受到键盘和输入法的影响，无法阻止用户通过其他方式输入非指定类型的字符。因此，在处理用户输入时仍然需要进行适当的验证和过滤。

**android:maxLines="1"**

**android:inputType** =”text”

实现单行模式，点击换行键能够隐藏键盘

多行模式换行键换行

需要和inputType配合使用

**android:imeOptions="actionNone"**

将换行键禁用

# 在有莫名其妙功能实现不了的时候检查是否有内存泄漏

包括dialog是否dismiss、viewModel的生命周期、activity的持有、Toast是否有未显示的。

如果有内存泄漏的情况可能会导致功能跑不起来