对话框

本文内容

关键类

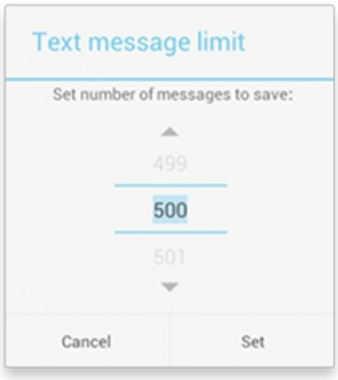
- `DialogFragment`
- `AlertDialog`

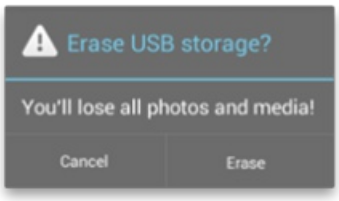
另请参阅

- 对话框设计指南
- 选取器（日期/时间对话框）

对话框是提示用户作出决定或输入额外信息的小窗口。 对话框不会填充屏幕，通常用于需要用户采取行动才能继续执行的模式事件。

> **对话框设计**
>
> 如需了解有关如何设计对话框的信息（包括语言建议），请阅读对话框设计指南。





`Dialog` 类是对话框的基类，但您应该避免直接实例化 `Dialog`，而是使用下列子类之一：

`AlertDialog`

此对话框可显示标题、最多三个按钮、可选择项列表或自定义布局。

`DatePickerDialog` 或 `TimePickerDialog`

此对话框带有允许用户选择日期或时间的预定义 UI。

这些类定义您的对话框的样式和结构，但您应该将 DialogFragment 用作对话框的容器。DialogFragment 所需的所有控件，而不是调用 Dialog 对象上的方法。

使用 DialogFragment 管理对话框可确保它能正确处理生命周期事件，如用户按*“返回”*按钮或旋转屏幕时。 此外，DialogFragment 类还允许您将对话框的 UI 作为嵌入式组件在较大 UI 中重复使用，就像传统 Fragment 一样（例如，当您想让对话框 UI 在大屏幕和小屏幕上具有不同外观时）。

避免使用 ProgressDialog

Android 包括另一种名为 ProgressDialog 的对话框类，可显示具有进度条的对话框。不过，如需指示加载进度或不确定的进度，则应改为遵循进度和 Activity 的设计指南，并在您的布局中使用 ProgressBar。

本指南的后文将描述如何将 DialogFragment 与 AlertDialog 对象结合使用。 如果您想创建一个日期或时间选取器，应改为阅读选取器指南。

注：由于 DialogFragment 类最初是通过 Android 3.0（API 级别 11）添加的，因此本文描述的是如何使用支持库附带的 DialogFragment 类。 通过将该库添加到您的应用，您可以在运行 Android 1.6 或更高版本的设备上使用 DialogFragment 以及各种其他 API。如果您的应用支持的最低版本是 API 级别 11 或更高版本，则可使用 DialogFragment 的框架版本，但请注意，本文中的链接适用于支持库 API。 使用支持库时，请确保您导入的是 `android.support.v4.app.DialogFragment` 类，而*不是* `android.app.DialogFragment`。

创建对话框片段

您可以完成各种对话框设计—包括自定义布局以及对话框设计指南中描述的布局—通过扩展 DialogFragment 并在 onCreateDialog() 回调方法中创建 AlertDialog。

例如，以下是一个在 DialogFragment 内管理的基础 AlertDialog：

```
public class FireMissilesDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
               .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // FIRE ZE MISSILES!
                   }
               })
               .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // User cancelled the dialog
                   }
               });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

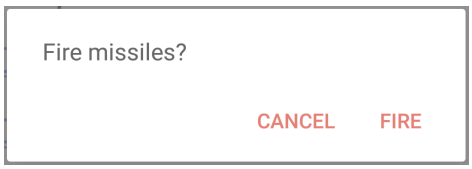


图 1. 一个包含消息和两个操作按钮的对话框。

现在，当您创建此类的实例并调用该对象上的 show() 时，对话框将如图 1 所示。

下文将详细描述如何使用 AlertDialog.Builder API 创建对话框。

根据对话框的复杂程度，您可以在 DialogFragment 中实现各种其他回调方法，包括所有基础片段生命周期方法。

构建提醒对话框

您可以通过 AlertDialog 类构建各种对话框设计，并且该类通常是您需要的唯一对话框类。如图 2 所示，提醒对话框有三个区域：

图 2. 对话框的布局。

1. **标题**

   这是可选项，只应在内容区域被详细消息、列表或自定义布局占据时使用。 如需陈述的是一条简单消息或问题（如图 1 中的对话框），则不需要标题。

2. **内容区域**

   它可以显示消息、列表或其他自定义布局。

3. **操作按钮**

   对话框中的操作按钮不应超过三个。

`AlertDialog.Builder` 类提供的 API 允许您创建具有这几种内容（包括自定义布局）的 `AlertDialog`。

要想构建 `AlertDialog`，请执行以下操作：

```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

// 2. Chain together various setter methods to set the dialog characteristics
builder.setMessage(R.string.dialog_message)
       .setTitle(R.string.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

以下主题介绍如何使用 `AlertDialog.Builder` 类定义各种对话框属性。

添加按钮

要想添加如图 2 所示的操作按钮，请调用 `setPositiveButton()` 和 `setNegativeButton()` 方法：

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
// Add the buttons
builder.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
               // User clicked OK button
           }
       });
builder.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
               // User cancelled the dialog
           }
       });
// Set other dialog properties
...

// Create the AlertDialog
AlertDialog dialog = builder.create();
```

`set...Button()` 方法需要一个按钮标题（由字符串资源提供）和一个 `DialogInterface.OnClickListener`，后者用于定义用户按下该按钮时执行的操作。

您可以添加三种不同的操作按钮：

肯定

> 您应该使用此按钮来接受并继续执行操作（“确定”操作）。

否定

> 您应该使用此按钮来取消操作。

中性

> 您应该在用户可能不想继续执行操作，但也不一定想要取消操作时使用此按钮。 它出现在肯定按钮和否定按钮之间。 例如，实际操作可能是“稍后提醒我”。

对于每种按钮类型，您只能为 `AlertDialog` 添加一个该类型的按钮。也就是说，您不能添加多个“肯定”按钮。

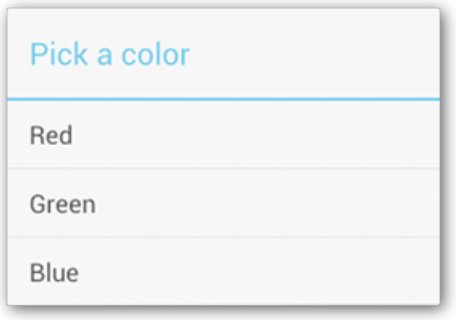


图 3. 一个包含标题和列表的对话框。

添加列表

可通过 `AlertDialog` API 提供三种列表：

- 传统单选列表
- 永久性单选列表（单选按钮）
- 永久性多选列表（复选框）

要想创建如图 3 所示的单选列表，请使用 `setItems()` 方法：

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    builder.setTitle(R.string.pick_color)
           .setItems(R.array.colors_array, new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int which) {
               // The 'which' argument contains the index position
               // of the selected item
           }
    });
    return builder.create();
}
```

由于列表出现在对话框的内容区域，因此对话框无法同时显示消息和列表，您应该通过 `setTitle()` 为对话框设置标题。要想指定列表项，请调用`setItems()` 来传递一个数组。或者，您也可以使用 `setAdapter()` 指定一个列表。 这样一来，您就可以使用 `ListAdapter` 以动态数据（如来自数据库的数据）支持列表。

如果您选择通过 `ListAdapter` 支持列表，请务必使用 `Loader`，以便内容以异步方式加载。使用适配器构建布局和加载程序指南中对此做了进一步描述。

> **注：**默认情况下，触摸列表项会清除对话框，除非您使用的是下列其中一种永久性选择列表。

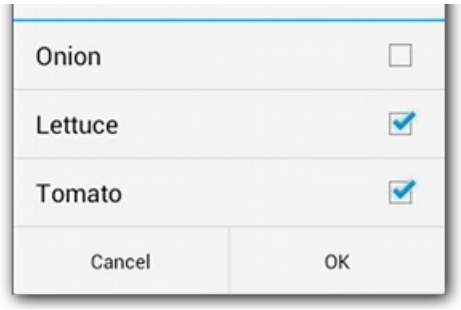


图 4. 多选项列表。

添加永久性多选列表或单选列表

要想添加多选项（复选框）或单选项（单选按钮）列表，请分别使用 `setMultiChoiceItems()` 或 `setSingleChoiceItems()` 方法。

例如，以下示例展示了如何创建如图 4 所示的多选列表，将选定项保存在一个 `ArrayList` 中：

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mSelectedItems = new ArrayList();  // Where we track the selected items
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    // Set the dialog title
    builder.setTitle(R.string.pick_toppings)
    // Specify the list array, the items to be selected by default (null for none),
    // and the listener through which to receive callbacks when items are selected
           .setMultiChoiceItems(R.array.toppings, null,
                      new DialogInterface.OnMultiChoiceClickListener() {
               @Override
               public void onClick(DialogInterface dialog, int which,
                       boolean isChecked) {
                   if (isChecked) {
                       // If the user checked the item, add it to the selected items
                       mSelectedItems.add(which);
                   } else if (mSelectedItems.contains(which)) {
                       // Else, if the item is already in the array, remove it
                       mSelectedItems.remove(Integer.valueOf(which));
                   }
               }
           })
    // Set the action buttons
           .setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
               @Override
               public void onClick(DialogInterface dialog, int id) {
                   // User clicked OK, so save the mSelectedItems results somewhere
                   // or return them to the component that opened the dialog
                   ...
               }
           })
           .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
               @Override
               public void onClick(DialogInterface dialog, int id) {
                   ...
               }
           });

    return builder.create();
}
```

尽管传统列表和具有单选按钮的列表都能提供“单选”操作，但如果您想持久保存用户的选择，则应使用 `setSingleChoiceItems()`。也就是说，如果稍后再次打开对话框时系统应指示用户的当前选择，那么您就需要创建一个具有单选按钮的列表。

创建自定义布局

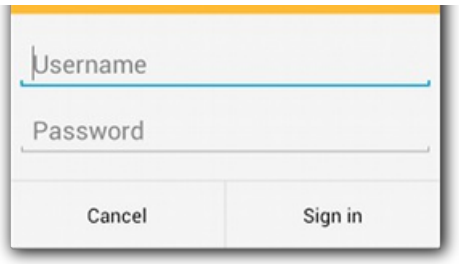


图 5. 自定义对话框布局。

如果您想让对话框具有自定义布局，请创建一个布局，然后通过调用 `AlertDialog.Builder` 对象上的 `setView()` 将其添加到 `AlertDialog`。

默认情况下，自定义布局会填充对话框窗口，但您仍然可以使用 `AlertDialog.Builder` 方法来添加按钮和标题。

例如，以下是图 5 中对话框的布局文件：

res/layout/dialog_signin.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ImageView
        android:src="@drawable/header_logo"
        android:layout_width="match_parent"
        android:layout_height="64dp"
        android:scaleType="center"
        android:background="#FFFFBB33"
        android:contentDescription="@string/app_name" />
    <EditText
        android:id="@+id/username"
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="@string/username" />
    <EditText
        android:id="@+id/password"
        android:inputType="textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="16dp"
        android:fontFamily="sans-serif"
        android:hint="@string/password"/>
</LinearLayout>
```

> **提示**：默认情况下，当您将 `EditText` 元素设置为使用 `"textPassword"` 输入类型时，字体系列将设置为固定宽度。因此，您应该将其字体系列更改为 `"sans-serif"`，以便两个文本字段都使用匹配的字体样式。

要扩展 `DialogFragment` 中的布局，请通过 `getLayoutInflater()` 获取一个 `LayoutInflater` 并调用 `inflate()`，其中第一个参数是布局资源 ID，第二个参数是布局的父视图。然后，您可以调用 `setView()` 将布局放入对话框。

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    // Get the layout inflater
    LayoutInflater inflater = getActivity().getLayoutInflater();

    // Inflate and set the layout for the dialog
    // Pass null as the parent view because its going in the dialog layout
    builder.setView(inflater.inflate(R.layout.dialog_signin, null))
    // Add action buttons
           .setPositiveButton(R.string.signin, new DialogInterface.OnClickListener() {
               @Override
               public void onClick(DialogInterface dialog, int id) {
                   // sign in the user ...
               }
           })
           .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int id) {
                   LoginDialogFragment.this.getDialog().cancel();
               }
           });
    return builder.create();
}
```

提示：如果您想要自定义对话框，可以改用对话框的形式显示 `Activity`，而不是使用 `Dialog` API。 只需创建一个 Activity，并在 `<activity>` 清单文件元素中将其主题设置为 `Theme.Holo.Dialog`：

```
<activity android:theme="@android:style/Theme.Holo.Dialog" >
```

就这么简单。Activity 现在会显示在一个对话框窗口中，而非全屏显示。

将事件传递回对话框的宿主

当用户触摸对话框的某个操作按钮或从列表中选择某一项时，您的 `DialogFragment` 可能会自行执行必要的操作，但通常您想将事件传递给打开该对话框的 Activity 或片段。 为此，请定义一个界面，为每种点击事件定义一种方法。然后在从该对话框接收操作事件的宿主组件中实现该界面。

例如，以下 `DialogFragment` 定义了一个界面，通过该界面将事件传回给宿主 Activity：

```
public class NoticeDialogFragment extends DialogFragment {

    /* The activity that creates an instance of this dialog fragment must
     * implement this interface in order to receive event callbacks.
     * Each method passes the DialogFragment in case the host needs to query it. */
    public interface NoticeDialogListener {
        public void onDialogPositiveClick(DialogFragment dialog);
        public void onDialogNegativeClick(DialogFragment dialog);
    }

    // Use this instance of the interface to deliver action events
    NoticeDialogListener mListener;

    // Override the Fragment.onAttach() method to instantiate the NoticeDialogListener
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        // Verify that the host activity implements the callback interface
        try {
            // Instantiate the NoticeDialogListener so we can send events to the host
            mListener = (NoticeDialogListener) activity;
        } catch (ClassCastException e) {
            // The activity doesn't implement the interface, throw exception
            throw new ClassCastException(activity.toString()
                    + " must implement NoticeDialogListener");
        }
    }
    ...
}
```

对话框的宿主 Activity 会通过对话框片段的构造函数创建一个对话框实例，并通过实现的 `NoticeDialogListener` 界面接收对话框的事件：

```
public class MainActivity extends FragmentActivity
                          implements NoticeDialogFragment.NoticeDialogListener{
    ...

    public void showNoticeDialog() {
        // Create an instance of the dialog fragment and show it
        DialogFragment dialog = new NoticeDialogFragment();
        dialog.show(getSupportFragmentManager(), "NoticeDialogFragment");
    }

    // The dialog fragment receives a reference to this Activity through the
    // Fragment.onAttach() callback, which it uses to call the following methods
    // defined by the NoticeDialogFragment.NoticeDialogListener interface
    @Override
    public void onDialogPositiveClick(DialogFragment dialog) {
        // User touched the dialog's positive button
        ...
    }

    @Override
    public void onDialogNegativeClick(DialogFragment dialog) {
        // User touched the dialog's negative button
        ...
    }
}
```

由于宿主 Activity 会实现 `NoticeDialogListener`—由以上显示的 `onAttach()` 回调方法强制执行 — 因此对话框片段可以使用界面回调方法向 Activity 传递点击事件：

```
public class NoticeDialogFragment extends DialogFragment {
    ...

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Build the dialog and set up the button click handlers
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
               .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // Send the positive button event back to the host activity
                       mListener.onDialogPositiveClick(NoticeDialogFragment.this);
                   }
               })
               .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // Send the negative button event back to the host activity
                       mListener.onDialogNegativeClick(NoticeDialogFragment.this);
                   }
               });
        return builder.create();
    }
}
```

显示对话框

如果您想显示对话框，请创建一个 `DialogFragment` 实例并调用 `show()`，以传递对话框片段的 `FragmentManager` 和标记名称。

您可以通过从 `FragmentActivity` 调用 `getSupportFragmentManager()` 或从 `Fragment` 调用 `getFragmentManager()` 来获取 `FragmentManager`。例如：

```
public void confirmFireMissiles() {
    DialogFragment newFragment = new FireMissilesDialogFragment();
    newFragment.show(getSupportFragmentManager(), "missiles");
}
```

第二个参数 `"missiles"` 是系统用于保存片段状态并在必要时进行恢复的唯一标记名称。 该标记还允许您通过调用 `findFragmentByTag()` 获取片段的句柄。

全屏显示对话框或将其显示为嵌入式片段

您可能采用以下 UI 设计：您想让一部分 UI 在某些情况下显示为对话框，但在其他情况下全屏显示或显示为嵌入式片段（也许取决于设备使用大屏幕还是小屏幕）。`DialogFragment` 类便具有这种灵活性，因为它仍然可以充当嵌入式 `Fragment`。

但在这种情况下，您不能使用 `AlertDialog.Builder` 或其他 `Dialog` 对象来构建对话框。如果您想让 `DialogFragment` 具有嵌入能力，则必须在布局中定义对话框的 UI，然后在 `onCreateView()` 回调中加载布局。

以下示例 `DialogFragment` 可以显示为对话框或嵌入式片段（使用名为 `purchase_items.xml` 的布局）：

```
public class CustomDialogFragment extends DialogFragment {
    /** The system calls this to get the DialogFragment's layout, regardless
        of whether it's being displayed as a dialog or an embedded fragment. */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        // Inflate the layout to use as dialog or embedded fragment
        return inflater.inflate(R.layout.purchase_items, container, false);
    }

    /** The system calls this only when creating the layout in a dialog. */
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // The only reason you might override this method when using onCreateView() is
        // to modify any dialog characteristics. For example, the dialog includes a
        // title by default, but your custom layout might not need it. So here you can
        // remove the dialog title, but you must call the superclass to get the Dialog.
        Dialog dialog = super.onCreateDialog(savedInstanceState);
        dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
        return dialog;
    }
}
```

以下代码可根据屏幕尺寸决定将片段显示为对话框还是全屏 UI：

```
public void showDialog() {
    FragmentManager fragmentManager = getSupportFragmentManager();
    CustomDialogFragment newFragment = new CustomDialogFragment();

    if (mIsLargeLayout) {
        // The device is using a large layout, so show the fragment as a dialog
        newFragment.show(fragmentManager, "dialog");
    } else {
        // The device is smaller, so show the fragment fullscreen
        FragmentTransaction transaction = fragmentManager.beginTransaction();
        // For a little polish, specify a transition animation
        transaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);
        // To make it fullscreen, use the 'content' root view as the container
        // for the fragment, which is always the root view for the activity
        transaction.add(android.R.id.content, newFragment)
                   .addToBackStack(null).commit();
    }
}
```

如需了解有关执行片段事务的详细信息，请参阅片段指南。

在本示例中，`mIsLargeLayout` 布尔值指定当前设备是否应该使用应用的大布局设计（进而将此片段显示为对话框，而不是全屏显示）。 设置这种布尔值的最佳方法是声明一个布尔资源值，其中包含适用于不同屏幕尺寸的备用资源值。 例如，以下两个版本的布尔资源适用于不同的屏幕尺寸：

res/values/bools.xml

```
<!-- Default boolean values -->
<resources>
    <bool name="large_layout">false</bool>
</resources>
```

res/values-large/bools.xml

```
<!-- Large screen boolean values -->
<resources>
    <bool name="large_layout">true</bool>
</resources>
```

然后，您可以在 Activity 的 `onCreate()` 方法执行期间初始化 `mIsLargeLayout` 值：

```
boolean mIsLargeLayout;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mIsLargeLayout = getResources().getBoolean(R.bool.large_layout);
}
```

将 Activity 显示为大屏幕上的对话框

相对于在小屏幕上将对话框显示为全屏 UI，您可以通过在大屏幕上将 `Activity` 显示为对话框来达到相同的效果。您选择的方法取决于应用设计，但当应用已经针对小屏幕进行设计，而您想要通过将短生存期 Activity 显示为对话框来改善平板电脑体验时，将 Activity 显示为对话框往往很有帮助。

要想仅在大屏幕上将 Activity 显示为对话框，请将 `Theme.Holo.DialogWhenLarge` 主题应用于 `<activity>` 清单文件元素：

```
<activity android:theme="@android:style/Theme.Holo.DialogWhenLarge" >
```

如需了解有关通过主题设置 Activity 样式的详细信息，请参阅**样式和主题**指南。

清除对话框

当用户触摸使用 `AlertDialog.Builder` 创建的任何操作按钮时，系统会为您清除对话框。

系统还会在用户触摸某个对话框列表项时清除对话框，但列表使用单选按钮或复选框时除外。 否则，您可以通过在 `DialogFragment` 上调用`dismiss()` 来手动清除对话框。

如需在对话框消失时执行特定操作，则可以在您的 `DialogFragment` 中实现 `onDismiss()` 方法。

您还可以*取消*对话框。这是一个特殊事件，它表示用户显式离开对话框，而不完成任务。 如果用户按“*返回*”按钮，触摸对话框区域外部的屏幕，或者您在 `Dialog` 上显式调用 `cancel()`（例如，为了响应对话框中的“取消”按钮），就会发生这种情况。

如上例所示，您可以通过在您的 `DialogFragment` 类中实现`onCancel()` 来响应取消事件。

> **注：**系统会在每个调用 `onCancel()` 回调的事件发生时立即调用 `onDismiss()`。不过，如果您调用 `Dialog.dismiss()` 或 `DialogFragment.dismiss()`，系统会调用 `onDismiss()`，而*不会*调用 `onCancel()`。因此，当用户在对话框中按“*肯定*”按钮，从视图中移除对话框时，您通常应该调用 `dismiss()`。