GitBook

Documentation

Published
with GitBook

Table of Contents

安卓开发文档·中文版

开发文档预览

https://android.developerdocumentation.cn

加入我们

我们是一个业余的翻译兴趣小组，如果你想加入我们，可以联系
root@developerdocumentation.cn

目前成员

misparking/jackwaiting/ionesmile/jarylan/edtj/ifeegoo

入门指南

欢迎来到 Android 开发者培训。这里提供了说明如何完成特定任务的培训课程，并带有可以在您的应用中重用的代码示例。在左侧导航栏顶部，您可以看到课程划分成多个单元。

下面的培训指南列出了 Android 应用开发的一些关键点。如果您是 Android 应用开发新手，应按顺序完成所有这些课程。

如果您更喜欢互动视频，则可以使用各种在线视频课程。

构建您的第一个应用

欢迎开发 Android 应用！

本课讲述如何构建您的第一个 Android 应用。您将学习如何使用 Android Studio 创建 Android 项目和运行可调试版本的应用。您还将了解一些 Android 应用设计的基础知识，包括如何构建简单的用户界面和处理用户输入。

在开始本课的学习之前，请下载并安装 Android Studio。

开始-->

创建 Android 项目

这篇课程将会告诉你如何使用Android Studio创建新的Android项目和介绍项目中的一些文件。

1.在 Android Studio 中，创建新的项目

- 如果你未打开任何项目，请在Weclome to Android Studio的窗口中，点击Start a new Android Studio project
- 如果你已打开项目，请选择File > New Project

2.在 New Project 输入以下内容

- Application Name: "My First App"
- Company Domain："example.com"
  Android Studio 会为你提供默认的软件包名称和项目位置，但您也可根据需求去编辑这些内容

3.点击 Next

4.在 Target Android Devices 屏幕中，保留默认并点击 Next。

Minimum Required SDK 是指您的应用所能支持的最低 Android 版本，由API级别表示。为了能支持尽可能多的设备，您应该设置成可让你的应用提供核心功能可用的最低版本。如果应用的任何功能只能在更新版本的 Android上运行，并且不属于核心功能集的关键功能，则可仅当在支持该功能的版本上运行时启用该功能（请参阅支持不同平台版本）。

5.在 Add an Activity to Mobile 屏幕中，选择 Empty Activity，然后点击 Next。

6.在 Customize the Activity 屏幕中，保留默认值并点击Finish。

经过一些处理后,Android Studio 将打开并显示包含默认文件的"Hello World"应用。在以下课程中，你将向其中一些文件添加功能。

下面让我们花一点时间回顾一下最重要的文件：首先，请确保已打开 Project 窗口（选择 `View > Tool Windows > Project` ），并从顶部的下拉列表中选择 Android 视图。随后，您可以看到下列文件：

`app > java > com.example.myfirstapp > MainActivity.java`
完成新项目向导后，该文件将显示在 Android Studio 中。 它包含您之前创建的 Activity 的类定义。当您构建并运行应用时，Activity 会启动，并加载显示“Hello world!”的布局文件。

`app > res > layout > activity_main.xml`
此 XML 文件定义您的 Activity 的布局。它包含带有文本“Hello world!”的 TextView 元素。

`app > manifests > AndroidManifest.xml`
清单文件描述应用的基本特性并定义其每个组件。 随着课程学习的深入和为应用添加组件的增多，您会反复用到这个文件。

`Gradle Scripts > build.gradle`
Android Studio 使用 Gradle 来编译和构建您的应用。您的项目的每个模块都有相应的 build.gradle 文件，整个项目也有相应的 build.gradle 文件。 通常您只关心模块（在本例中，为 app 或称应用模块）的 build.gradle 文件。如需了解有关此文件的详细信息，请参阅使用 Gradle 构建您的项目。

要运行该应用，请继续学习下一课。

> 翻译：@edtj
> 原始文档：https://developer.android.google.cn/training/basics/firstapp/creating-project.html

运行你的 App

在上一课，我们创建了一个“Hello World”的项目。现在运行App在真机或模拟器上，如果你还没有一个真机，请直接看 在模拟器上运行。

在真机上运行

环境准备，将你的设备调整为如下状态：
1，通过USB数据线连接手机到你的电脑上。如果你的电脑系统是Windows，你可能需要为你的电脑安装一个合适的USB驱动。驱动安装帮助，请参见 OEM USB Drivers 文档。

2，能够使用 **USB**调试 你的手机，打开 设置 **>** 开发者选项

> 提示：在Android4.2以及更高的版本上， 开发者选项 默认是隐藏的。打开 设置 **>** 关于手机 ，连续点击 版本号 七次。返回到设置界面去找 开发者选项。

在Android Studio上运行App的方式如下：

1. 在Android Studio上选中项目，然后点击 **Run** 在工具栏上。
2. 在 **Select Deployment Target** 的对话框中，选择设备并点击 **OK**。

Android开始安装App在设备上，并会启动它。

在模拟器上运行

如果你之前没有运行App在模拟器上，你需要先创建一个 Android模拟器（AVD）。配置一个类型如Android手机、平板、穿戴、TV在你的Android模拟器上。

创建Android模拟器的方式如下：

1，打开Android模拟器管理窗口，工具》 **Android**》 **Android**模拟器管理，或者点击工具栏上的快捷图标。

2，在 你的虚拟设备 页面中，点击 创建虚拟设备 。

3，在 选择硬件 页面中，选择一款手机，如Nexus 6，接着点击 **Next**。

4，在 系统镜像 页面中，为模拟器选择一个想要的系统，接着点击 **Next**。
如果你选中的那个系统镜像没有安装，请点击“下载”链接进行安装。

5，检查配置设置（对于第一个AVD，保持原样的设置），点击 **Finish**。
关于使用Android虚拟机的更多信息，请参见 创建和管理虚拟设备 。

在Android Studio上运行App的方式如下：

1，在 **Android Studio** 上选中项目，然后点击 **Run** 在工具栏上。

2，在**Select Deployment Target**的对话框中，选择设备并点击 **OK**。

让这个模拟器启动你需要等待一会。之后你需要解锁屏幕。当你解锁后，“我的第一个App”就展现在模拟器上了。

这些是告诉你如何创建并运行App在模拟器上！加油，继续 下一课。

> 翻译：@iOnesmile
> 原始文档：https://developer.android.com/training/basics/firstapp/running-app.html

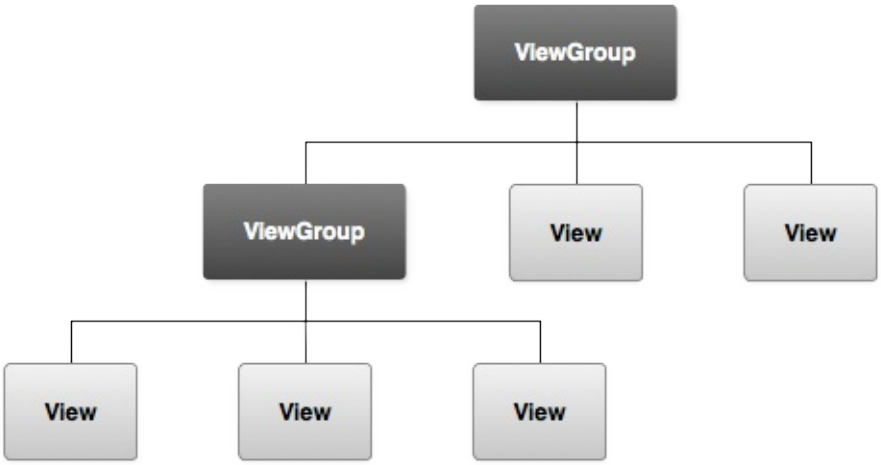
构建简单的用户界面

在本课中，你将学习如何创建一个 XML 格式的布局，其中包含一个文本字段和一个按钮。 在下一课中，在按下此按钮时，你的应用会将文本字段的内容发送给另一个 Activity 作为响应。

Android 应用的图形界面使用 View 对象和 ViewGroup 对象层次结构而构建。View 对象通常为按钮或文本字段之类的 UI 小部件。而 ViewGroup 对象则为不可见的视图容器，它们定义子视图的布局，比如是网格布局还是垂直列表布局。

Android 提供对应于 View 和 ViewGroup 子类的 XML 词汇，以便你使用 UI 元素层次结构以 XML 格式定义 UI。

布局是 ViewGroup 的子类。在本练习中，你将学习创建 LinearLayout。



创建线性布局

1.在 Android Studio 的 Project 窗口中，打开 app > res > layout > activity_main.xml。 此 XML 文件定义 Activity 的布局。它包含默认的“Hello World”文本视图。

2.在布局编辑器中，当你打开一个布局文件时，首先显示的是设计编辑器。 在本课中，你将直接使用 XML，因此请点击窗口底部的 Text 标签以切换到文本编辑器。

3.删除所有内容并插入以下 XML：

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">
    </LinearLayout>
```

LinearLayout 是一个视图组（ViewGroup 的子类），它会按照 android:orientation 属性的指定，将子视图设置为垂直或水平方向布局。LinearLayout 的每个子视图都会按照它们各自在 XML 中的出现顺序显示在屏幕上。

其他两个属性 android:layout_width 和 android:layout_height 则是所有视图的必备属性，用于指定它们的尺寸。

LinearLayout 是布局中的根视图，因此应将宽度和高度设置为 "match_parent"，从而填满可供应用使用的整个屏幕区域。 该值表示视图应扩大其宽度或高度，以匹配父视图的宽度或高度。

如需了解有关布局属性的详细信息，请参阅布局指南。

添加文本字段

在 activity_main.xml 文件的 <LinearLayout 元素内，添加以下 <EditText 元素：

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
</LinearLayout>
```

无需担心出现 @string/edit_message 错误；你很快就会修复该错误。

下面说明了你添加的 <EditText 中的属性：

android:id

这会为视图赋予唯一的标识符，你可以使用该标识符从应用代码中引用对象，例如读取和操作对象（下一课中将有介绍）。

从 XML 引用任何资源对象时，都需要使用 @ 符号。请在该符号后依次输入资源类型（本例中为 id）、斜杠和资源名称 (edit_message)。

只有在第一次定义资源 ID 时，才需要在资源类型之前加一个加号 (+)。当你编译应用时，SDK 工具会使用 ID 名称在项目的 R.java 文件中新建一个引用 EditText 元素的资源 ID。一旦以此方式声明资源 ID，其他对该 ID 的引用皆无需使用加号。只有在指定新资源 ID 时才必须使用加号，对于字符串或布局等具体资源则不必如此。如需了解有关资源对象的详细信息，请参阅边栏。

> 资源对象
>
> 资源对象是一个唯一的整型名称，它与应用资源（如位图、布局文件或字符串）相关联。
>
> 每个资源在你项目的 R.java 文件中都定义有相应的资源对象。你可以使用 R 类中的对象名称来引用你的资源，例如当你需要为 android:hint 属性指定字符串值时，就可以这样做。你也可以使用 android:id 属性创建任意与视图相关联的资源 ID，从而可以从其他代码中引用该视图。
>
> SDK 工具会在你每次编译应用时生成 R.java 文件。切勿手动修改该文件。
>
> 如需了解详细信息，请阅读提供资源指南。

android:layout_width 和 android:layout_height

"wrap_content" 值并不规定宽度和高度的具体大小，而是指定根据需要缩放视图，使其适合视图的内容。如果你要改用 "match_parent"，则 EditText 元素将填满屏幕，因为它会匹配父 LinearLayout 的大小。如需了解详细信息，请参阅布局指南。

android:hint

这是文本字段为空时显示的默认字符串。"@string/edit_message" 并非使用硬编码字符串作为其值，而是引用另一个文件中定义的一个字符串资源。由于它引用的是一个具体资源（而不仅仅是标识符），因此不需要加号。不过，由于你尚未定义字

符串资源，所以最初会出现编译错误。 在下一节中，你可以通过定义字符串来修正该错误。

```
注：此字符串资源与元素 ID edit_message 同名。不过，对资源的引用始终都按资源类
```

添加字符串资源

默认情况下，你的 Android 项目在 res > values > strings.xml 位置包含一个字符串资源文件。你需要在此处添加两个新字符串。

1.从 Project 窗口中，打开 res > values > strings.xml。

2.添加两个字符串，使你的文件如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My First App</string>
    <string name="edit_message">Enter a message</string>
    <string name="button_send">Send</string>
</resources>
```

对于用户界面中的文本，务必将每个字符串都指定为资源。 字符串资源允许你在单一位置管理所有 UI 文本，从而简化文本的查找和更新。 此外，将字符串外部化还可让你为每个字符串资源提供替代定义，从而将你的应用本地化为不同的语言。

如需了解有关使用字符串资源将应用本地化为其他语言的详细信息，请参阅支持不同设备课程。

添加按钮

返回到 activity_main.xml 文件并在 <EditText 后添加一个按钮。你的文件应如下所示：

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
        <EditText android:id="@+id/edit_message"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:hint="@string/edit_message" />
        <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/button_send" />
</LinearLayout>
```

> 注：此按钮不需要 android:id 属性，因为不需要从 Activity 代码中引用它。

如图 2 所示，该布局当前设计为根据内容大小调整 EditText 和 Button 小部件的尺寸。

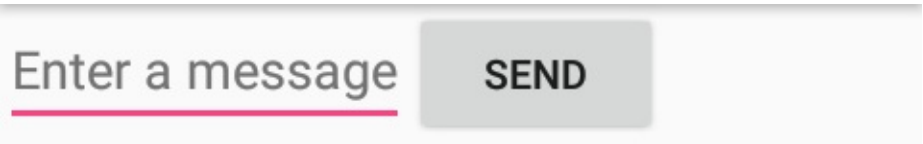


图 2. EditText 小组件和 Button 小部件的宽度均设置为 "wrap_content"。

这对按钮很适合，但不太适合文本字段，因为用户键入的内容可能更长。 最好让文本字段填满未使用的屏幕宽度。 你可以在 LinearLayout 内通过 weight 属性来实现此目的，该属性可使用 android:layout_weight 属性来指定。

Weight 值是一个数字，用于指定每个视图与其他同级视图在剩余空间中的占比。 这有点像饮料配方中各种成分的比例：“2 份苏打、1 份糖浆”是指饮料中三分之二是苏打。例如，如果你将一个视图的 weight 值指定为 2，将另一个视图的 weight 值指定为 1，总和是 3，那么第一个视图将填满剩余空间的 2/3，而第二个视图则填满其余部分。 如果你添加了第三个视图，将其 weight 值指定为 1，那么现在第一个视图（weight 值为 2）将获得 1/2 的剩余空间，其余两个视图则各占 1/4。

所有视图的默认 weight 值都为 0，所以如果你仅将一个视图的 weight 值指定为大于 0，那么等到其他所有视图都获得所需空间后，该视图便会填满所有剩余空间。

使输入框填满屏幕宽度

在 activity_main.xml 中，修改 <EditText ，使这些属性如下所示：

```
<EditText android:id="@+id/edit_message"
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

将宽度设置为零 (0dp) 可提高布局性能，这是因为如果将宽度设置为 "wrap_content"，则会要求系统计算宽度，而该计算最终毫无意义，因为 weight 值还需要计算另一个宽度，才能填满剩余空间。

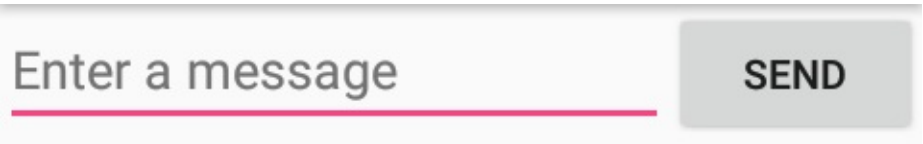


图 3. EditText 小部件获得了布局的所有 weight，因此它填满了 LinearLayout 中的剩余空间。

完整的 activity_main.xml 布局文件现在看上去应该像下面这样：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/and
   xmlns:tools="http://schemas.android.com/tools"
   android:orientation="horizontal"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />
</LinearLayout>
```

运行你的应用

要查看应用目前在设备或模拟器上的外观，请点击工具栏中的 Run 。

要添加诸如响应按钮和启动另一个 Activity 等应用行为，请继续学习下一课。

> 备注
> 翻译: @jarylan
> 原始文档: https://developer.android.com/training/basics/firstapp/building-ui.html

启动另一个 Activity

完成上一课的学习后，您已构建了一个显示一个 Activity（单一屏幕）并带有一个文本字段和一个按钮的应用。 在本课中，您将为 MainActivity 添加一些代码，这些代码可在用户点击“Send”按钮时启动一个新的 Activity。

响应 Send 按钮

1、在文件 res > layout > activity_main.xml 中，将 android:onClick 属性添加到 Button 元素，如下所示：

```
<Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/button_send"
      android:onClick="sendMessage" />
```

每次用户点击按钮时，此属性均会提示系统调用 Activity 中的 sendMessage() 方法。

2、在文件 java > com.example.myfirstapp > MainActivity.java 中，添加 sendMessage() 方法存根，如下所示：

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
       }

    /** Called when the user clicks the Send button */
    public void sendMessage(View view) {
        // Do something in response to button
    }
}
```

要让系统将此方法与为 android:onClick 指定的方法名称匹配，签名必须与所示内容完全相同。具体而言，该方法必须：

- 是公共方法
- 具有空返回值
- 以 View 作为唯一参数（这将是之前点击的 View）

接下来，您需要填写此方法以读取文本字段的内容，并将该文本传递给另一个 Activity。

构建一个 **Intent**

Intent 是指在相互独立的组件（如两个 Activity）之间提供运行时绑定功能的对象。Intent 表示一个应用“执行某项操作的 Intent”。 您可以将 Intent 用于各种任务，但在本课程中，Intent 用于启动另一个 Activity。

在 MainActivity.java 中，将如下所示代码添加到 sendMessage()：

```
    public class MainActivity extends AppCompatActivity {
        public final static String EXTRA_MESSAGE = "com.example.myf
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
        }

        /** Called when the user clicks the Send button */
        public void sendMessage(View view) {
            Intent intent = new Intent(this, DisplayMessageActivity
            EditText editText = (EditText) findViewById(R.id.edit_m
            String message = editText.getText().toString();
            intent.putExtra(EXTRA_MESSAGE, message);
            startActivity(intent);
        }
    }
```

Android Studio 将显示无法解析符号错误，这是因为此代码引用了未导入的类。通过按 Alt + Enter（在 Mac 上，则按 Option + Return），您可使用 Android Studio 的“导入类”功能来解决其中一些问题。您的导入应按如下所示结束：

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
```

仍出现 **DisplayMessageActivity** 错误，但没关系，您将在下一部分中修复该错误。

sendMessage() 中会执行许多操作，因此我们来解释下具体会执行哪些操作。

Intent 构造函数采用两个参数：

- Context 是第一个参数（之所以使用 this ，是因为 Activity 类是 Context 的子类）
- 应用组件的 Class，系统应将 Intent（在本例中，为应启动的 Activity）传递至该类。

  注：引用 DisplayMessageActivity 会在 Android Studio 中引发错误，因为该类尚不存在。可以暂时忽略该错误，因为您很快要创建该类。

putExtra() 方法将 EditText 的值添加到 Intent。Intent 能够以名为 extra 的键值对形式携带数据类型。您的键是一个公共常量 EXTRA_MESSAGE，因为下一个 Activity 将使用该键来检索文本值。为 Intent extra 定义键时最好使用应用的软件包名称作为前缀。这可以确保在您的应用与其他应用交互过程中这些键始终保持唯一。

startActivity() 方法将启动 Intent 指定的 DisplayMessageActivity 实例。现在，您需要创建类。

创建第二个 **Activity**

1. 在 Project 窗口中，右键点击 app 文件夹并选择 New > Activity > Empty Activity。

2. 在 Configure Activity 窗口中，为 Activity Name 输入 “DisplayMessageActivity”，然后点击 Finish

Android Studio 自动执行三项操作：

- 使用必需的 onCreate() 方法的实现创建类 DisplayMessageActivity.java。
- 创建对应的布局文件 activity_display_message.xml
- 在 AndroidManifest.xml 中添加必需的 元素。

如果运行应用并在第一个 Activity 上点击“Send”按钮，则将启动第二个 Activity，但它为空。这是因为第二个 Activity 使用模板提供的默认空布局。

显示消息

现在，您将修改第二个 Activity，以显示第一个 Activity 传递的消息。

1. 在 DisplayMessageActivity.java 中，向 onCreate() 方法添加下列代码：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_display_message);

   Intent intent = getIntent();
   String message = intent.getStringExtra(MainActivity.EXTRA_M
   TextView textView = new TextView(this);
   textView.setTextSize(40);
   textView.setText(message);

   ViewGroup layout = (ViewGroup) findViewById(R.id.activity_d
   layout.addView(textView);
}
```

按 Alt + Enter（在 Mac 上，则按 Option + Return）导入缺少的类。 您的导入应按如下所示方式结束：

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.TextView;
```

此处会执行大量操作，因此我们接着解释：

1. 调用 getIntent() 采集启动 Activity 的 intent。无论用户如何导航到目的地，每个 Activity 都由一个 Intent 调用。 调用 getStringExtra() 将检索第一个 Activity 中的数据。

1. 您可以编程方式创建 TextView 并设置其大小和消息。

2. 您可将 TextView 添加到 R.id.activity_display_message 标识的布局。您可将布局投射到 ViewGroup，因为它是所有布局的超类且包含 addView() 方法。

注：早期版本的 **Android Studio** 生成的 **XML** 布局可能不包括 **android:id** 属性。如果布局没有 **android:id** 属性，则调用 **findViewById()** 将失败。如果出现这种情况，请打开 **activity_display_message.xml** 并将属性 **android:id="@+id/activity_display_message"** 添加到布局元素。

您现在便可运行该应用。当应用打开后，请在文本字段键入消息，然后点击 Send。第二个 Activity 将替换屏幕中的第一个 Activity，并显示您在第一个 Activity 中输入的消息。

就这么简单，您的第一个 Android 应用已成功诞生！

备注：

翻译：JackWaiting[https://github.com/JackWaiting] 原始文：https://developer.android.com/training/basics/firstapp/starting-activity.html#RespondToButton

aaaaaaaa

兼容不同的设备

全世界的Android设备有着各种各样的大小和尺寸。通过各种各样的设备类型，能使我们通过自己的 app 接触到广大的用户群体。为了能在各种 Android 平台上使用，我们的 app 需要兼容各种不同的设备类型。你需要考虑一些如不同的语言，屏幕尺寸，Android 的系统版本等重要因素。

本课程会教我们如何使用基础的平台功能，利用替代资源和其他功能，使 app 仅用一个程序包(APK)，就能向用 Android 兼容设备的用户提供最优的用户体验。

Lessons

适配不同的语言

学习如何使用字符串替代资源实现支持多国语言。

适配不同的屏幕

学习如何根据不同尺寸分辨率的屏幕来优化用户体验。

适配不同的系统版本

学习如何在使用新的用户编程接口(API)时向下兼容旧版本Android。

支持不同的语言

在你的应用中把UI字符串存放到外部文件中并通过代码获取，一直是一个好的办法。Android 在每个 Android 项目中都提供一个资源目录，从而简化了这一过程。

如果你是使用 Android SDK Tools创建的工程（请阅读创建 Android 项目），则会在工程的根目录中创建一个 `res/` 目录。此目录中存放着各类资源的子目录，还包含一些默认文件如 `res/values/strings.xml` ，用于存放你的字符串值。

创建不同的语言区域目录和字符串文件

为了支持多国语言，需要在 `res/` 中创建一个额外的 `valuesres/` 目录，并在目录名称末尾加上连字符和ISO 语言代码。例如， `values-es/` 目录包含的简单资源用于语言代码为“es”的语言区域。Android 应用在运行时会根据设备的语言区域设置加载相应的资源。如需了解详细信息，请参阅提供备用资源。

一旦你决定了为哪些语言提供支持，便可创建资源子目录和字符串资源文件。 例如：

```
MyProject/
      res/
         values/
               strings.xml
          values-es/
               strings.xml
          values-fr/
                strings.xml
```

将各个语言区域的字符串值添加到相应文件中。

在运行时，Android 系统会根据当前为用户设备设置的语言区域使用相应的字符串资源集。

例如，以下是一些面向不同语言的不同字符串资源文件。

英语（默认语言区域)， `/values/strings.xml` ：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">My Application</string>
    <string name="hello_world">Hello World!</string>
</resources>
```

西班牙语，`/values-es/strings.xml`：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mi Aplicación</string>
    <string name="hello_world">Hola Mundo!</string>
</resources>
```

法语，`/values-fr/strings.xml`：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
    <string name="hello_world">Bonjour le monde !</string>
</resources>
```

> **Note**: You can use the locale qualifier (or any configuration qualifer) on any resource type, such as if you want to provide localized versions of your bitmap drawable. For more information, see Localization.

使用字符串资源

你可以在源代码和其他XML文件中通过`<string>`元素的`name`属性来引用自己的字符串资源。

在源代码中可以通过`R.string.<string_name>`语法来引用一个字符串资源，很多方法都可以通过这种方式来接受字符串.

例如:

```
// Get a string resource from your app's Resources
String hello = getResources().getString(R.string.hello_world);

// Or supply a string resource to a method that requires a string
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

在其他XML文件中，只要 XML 属性接受字符串值，你就可以使用@string/语法来引用字符串资源。
例如:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

> 翻译：@misparking
> 原始文档：https://developer.android.com/training/basics/supporting-devices/languages.html

支持不同屏幕

Android 使用两种常规属性来归类设备屏幕：尺寸和密度。你应当期望你的应用安装到有屏幕的设备上，能够适配其屏幕尺寸和密度。所以，你应当通过引入一些备选资源来优化你在不同屏幕尺寸和密度上的应用显示。

- 四种常规尺寸: 小, 一般, 大, 超大
- 四种常规密度：低 (ldpi), 中等 (mdpi), 高 (hdpi), 超高 (xhdpi)

针对不同的屏幕的适配，你需要将这些不同的布局和图片的备选资源放置在单独的目录中，类似不同语言字符串的处理。

同样，你应当意识到屏幕的方向（横屏或竖屏）是被当做不同的屏幕尺寸，所以，很多应用需要针对不同的屏幕方向进行布局的用户体验优化。

创建不同的布局

为了优化不同屏幕尺寸上的用户体验，你应当针对你想要支持的的每一个屏幕尺寸创建一个唯一的布局 `XML` 文件。每一个布局文件应当被放置到正确的资源目录，以 `-<屏幕尺寸>` 的后缀来命名。例如，针对大屏幕的唯一布局应当存在 `res/layout-large/` 目录底下。

> 备注：Android 会自动的调整你布局的比例，来正确的适配屏幕。所以，你不必担心不同屏幕尺寸上布局中的 UI 元素的绝对尺寸，反而需要注意的是关注影响用户体验的布局结构（例如重要视图之间的尺寸或位置）。

例如，这个工程包含一个默认的布局和一个针对大屏幕的备用布局：

```
MyProject/
    res/
        layout/
            main.xml
        layout-large/
            main.xml
```

文件名称必须完全一样，但是他们的内容是不同的，根据相关屏幕尺寸提供不同优化的 UI。

通常以以下方法在你的应用中简单的引入布局文件：

```
@Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
}
```

当你的应用运行时，系统会自动根据当前屏幕尺寸从布局目录中载入正确的布局文件。更多关于 Android 如何选择合适资源的信息，可以参见资源提供向导。

另外一个例子，这里有一个工程，有针对横屏的备用布局：

```
MyProject/
    res/
        layout/
            main.xml
        layout-land/
            main.xml
```

默认情况下，`layout/main.xml` 文件是针对竖屏的。

如果你想针对横屏并且是大屏幕提供一个指定的布局，你就需要同时使用 `large` 和 `land` 标签：

```
MyProject/
    res/
        layout/              # default (portrait)
            main.xml
        layout-land/         # landscape
            main.xml
        layout-large/        # large (portrait)
            main.xml
        layout-large-land/   # large landscape
            main.xml
```

> 备注：Android 3.2 及以上系统，支持一个先进的方法来定义屏幕尺寸，它支持基于最小的以 `dip` 为单位的宽度和高度，来指定屏幕尺寸资源。这篇教程不包含此新技术。更多信息，请参阅多屏幕设计。

创建不同的图片

你应当针对低、中、高和超高密度的屏幕分别提供正确缩放比例的图片。这会帮助你在所有的屏幕密度上获得好的图片质量和性能。

你应当通过矢量格式的原始资源来生成对应目睹的以下尺寸比例的图片：

- 超高密度: 2.0
- 高密度: 1.5
- 中等密度: 1.0 (基准)
- 低密度: 0.75

这就意味着，如果你针对超高密度设备生成的是一个 200x200 尺寸图片的话，那么针对高密度、中等密度和低密度，需要分别生成 150x150、100x100、75x75 尺寸的图片。

然后，将这些图片文件放置到正确的图片资源目录中：

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

任何时候，只要你引用 `@drawable/awesomeimage` 资源，系统会根据当前的屏幕密度选择合适的图片。

> 备注：低密度（ldpi）资源不一定总是必须的。当你只提供高分辨率资源，系统将会自动缩放 1.5 倍来适配低分辨率屏幕。

更多关于创建应用图标资源细节和指导原则，请参见图标设计指导。

备注：
翻译：@ifeegoo
原始文档：https://developer.android.com/training/basics/supporting-devices/screens.html

支持不同平台版本

尽管最新的 Android 版本给你的应用提供极好的 API，但是也仍然需要继续支持旧的 Android 版本，直到更多的设备更新到最新的系统。这节课将教你在使用最新的 API 的同时，继续支持旧版本的系统。

更新的平台版本的表格展示了基于访问 Google Play 商店设备的数量，以及运行在活跃设备上的每一个 Android 版本的分布情况。通常，支持 约 90% 的活跃设备，以及保证你的应用采用最新的版本编译，是一个良好的习惯。

> 备注：为了能够针对一些 Android 版本提供最好的特性和功能，你应当在应用中使用 Android Support Library，他可以让你在旧版本的平台上使用一些最新的平台 API。

指定最新和目标 API 级

AndroidManifest.xml 文件中描述了你的应用细节和标识了支持的 Android 版本。一般通过 `<uses-sdk>` 元素的 `minSdkVersion` 和 `targetSdkVersion` 属性来分别指定你应用适配的最低版本的 API 以及你设计和测试应用的最高版本 API。

例如：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/androi
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="
    ...
</manifest>
```

新的 Android 版本发布的时候，一些样式和特征可能会发生变化。为了允许你的应用使用这些变化，并且确保你的应用适配每一个用户设备的样式，你应当将 `targetSdkVersion` 的值设置成当前最新可用的 Android 版本。

运行时检测系统版本

Android 在 Build 常量类中提供了针对每一个平台版本的唯一代码。请在你的应用中使用这些代码来构建条件，以确保依赖于更高级别的 API 的代码在当前系统可用的情况下执行。

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb or higher to use Actior
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

> 备注：Android 解析 XML 资源的时候，会忽略当前设备不支持的 XML 属性。所以你可以安全的使用只有新版本的系统才支持的 XML 属性，而不必担心旧版本的系统因为这些代码而挂掉。例如，如果你设置 `targetSdkVersion="11"`，那么你的应用会在 Android 3.0 或者以上版本默认包含 ActionBar。然后如果你需要在 Action Bar 上添加目录条目，你需要在你的目录 XML 资源中设置 `android:showAsAction="ifRoom"`，在跨版本的 XML 文件中这么做是安全的，因为旧版本的 Andorid 系统会简单的忽略 `showAsAction` 属性。（也就是说，你不需要再在 `res/menu-v11/` 目录底下单独的给出一个 xml 文件）。

使用平台样式和主题

Android 提供了让你的应用有潜在的操作系统的视觉和感觉的用户体验主题。这些主题可以通过主配置文件加入到你的应用。通过使用这些内置的样式和主题，你的应用将会自然跟随每一个最新版本的 Android 系统的视觉和感觉设计。

让你的 Activity 看起来像一个对话框：

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

让你的 Activity 背景透明：

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

载入通过 `/res/values/styles.xml` 文件自定义的主题：

```
<activity android:theme="@style/CustomTheme">
```

在 `<application>` 元素中添加 `android:theme` 让你整个应用（所有Activity）都是用同一个主题：

```
<application android:theme="@style/CustomTheme">
```

更多关于创建和使用主题的信息，请阅读样式和主题指导。

备注：
翻译：@ifeegoo
原始文档：https://developer.android.com/training/basics/supporting-devices/platforms.html

#

启动 Activity

不同于使用 main() 方法启动应用的其他编程范例，Android 系统会通过调用对应于其生命周期中特定阶段的特定回调方法在 Activity 实例中启动代码。有一系列可启动 Activity 的回调方法，以及一系列可分解 Activity 的回调方法。

本课程概述了最重要的生命周期方法，并向您展示如何处理创建 Activity 新实例的第一个生命周期回调

了解生命周期回调

在 Activity 的生命周期中，系统会按类似于阶梯金字塔的顺序调用一组核心的生命周期方法。 也就是说，Activity 生命周期的每个阶段就是金字塔上的一阶。 当系统创建新 Activity 实例时，每个回调方法会将 Activity 状态向顶端移动一阶。 金字塔的顶端是 Activity 在前台运行并且用户可以与其交互的时间点。

当用户开始离开 Activity 时，系统会调用其他方法在金字塔中将 Activity 状态下移，从而销毁 Activity。 在有些情况下，Activity 将只在金字塔中部分下移并等待（比如，当用户切换到其他应用时），Activity 可从该点开始移回顶端（如果用户返回到该 Activity），并在用户停止的位置继续。

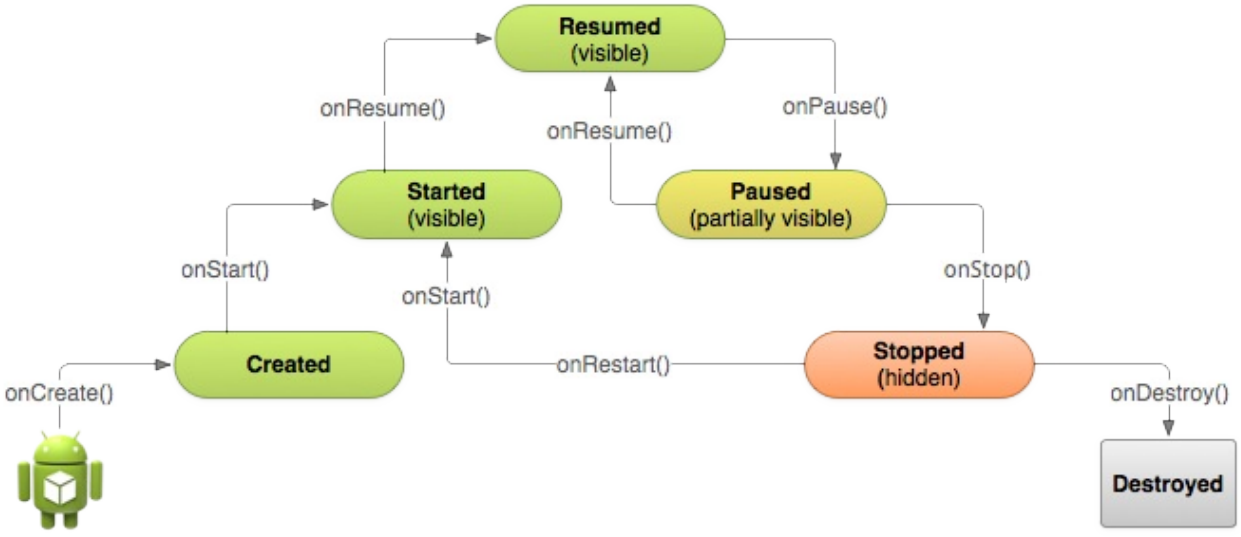


图 1. 简化的 Activity 生命周期图示，以阶梯金字塔表示。此图示显示，对于用于将 Activity 朝顶端的“继续”状态移动一阶的每个回调，有一种将 Activity 下移一阶的回调方法。 Activity 还可以从“暂停”和“停止”状态回到继续状态。

根据 Activity 的复杂程度，您可能不需要实现所有生命周期方法。 但是，了解每个方法并实现确保您的应用按照用户期望的方式运行的方法非常重要。 正确实现您的 Activity 生命周期方法可确保您的应用按照以下几种方式良好运行，包括：

- 如果用户在使用您的应用时接听来电或切换到另一个应用，它不会崩溃。
- 在用户未主动使用它时不会消耗宝贵的系统资源。
- 如果用户离开您的应用并稍后返回，不会丢失用户的进度。
- 当屏幕在横向和纵向之间旋转时，不会崩溃或丢失用户的进度。

正如您将要在以下课程中要学习的，存在 Activity 会在图 1 所示不同状态之间过渡的几种情况。但是，这些状态中只有三种可以是静态。也就是说，Activity 只能在三种状态之一下存在很长时间。

继续

```
在这种状态下，Activity 处于前台，且用户可以与其交互。（有时也称为“运行”状态。
```

暂停

```
在这种状态下，Activity 被在前台中处于半透明状态或者未覆盖整个屏幕的另一个 Act
```

停止

```
在这种状态下，Activity 被完全隐藏并且对用户不可见；它被视为处于后台。 停止时，
```

其他状态（“创建”和“开始”）是瞬态，系统会通过调用下一个生命周期回调方法从这些状态快速移到下一个状态。 也就是说，在系统调用 onCreate() 之后，它会快速调用 onStart()，紧接着快速调用 onResume()。

基本 Activity 生命周期部分到此为止。现在，您将开始学习特定生命周期行为的一些知识。

指定您的应用的启动器 Activity

当用户从主屏幕选择您的应用图标时，系统会为您已声明为“启动器”（或“主要”）Activity 的应用中的 Activity 调用 onCreate() 方法。这是作为 您的应用的用户界面主入口的 Activity。

您可以在 Android 清单文件 AndroidManifest.xml 中定义哪个 Activity 用作主 Activity，该文件位于您项目目录的根目录中。

您的应用的主 Activity 必须使用 （包括 MAIN 操作和 LAUNCHER 类别）在清单文件中声明。例如：

```
<activity android:name=".MainActivity" android:label="@string/app_
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" /
    </intent-filter>
</activity>
```

> 注: 当您使用 Android SDK工具创建新 Android 项目时，默认的项目文件包括使用过滤器在宣示说明中声明的 Activity 类。

如果未对您的 Activity 之一声明 MAIN 操作或 LAUNCHER 类别，那么您的应用图标将不会出现在应用的主屏幕列表中。

创建一个新实例 大多数应用包含若干个不同的 Activity，用户可通过这些 Activity 执行不同的操作。无论 Activity 是用户单击您的应用图标时创建的主 Activity 还是您的应用在响应用户操作时开始的其他 Activity，系统都会通过调用其 onCreate() 方法创建 Activity 的每个新实例。

您必须实现 onCreate() 方法执行只应在 Activity 整个生命周期出现一次的基本应用启动逻辑。 例如，您的 onCreate() 的实现应定义用户界面并且可能实例化某些类范围变量。

例如，onCreate() 方法的以下示例显示执行 Activity 某些基本设置的一些代码，比如声明用户界面（在 XML 布局文件中定义）、定义成员变量，以及配置某些 UI。

```
TextView mTextView; // Member variable for text view in the layout

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set the user interface layout for this Activity
    // The layout file is defined in the project res/layout/main_ac
    setContentView(R.layout.main_activity);

    // Initialize member TextView so we can manipulate it later
    mTextView = (TextView) findViewById(R.id.text_message);

    // Make sure we're running on Honeycomb or higher to use Actior
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // For the main activity, make sure the app icon in the act
        // does not behave as a button
        ActionBar actionBar = getActionBar();
        actionBar.setHomeButtonEnabled(false);
    }
}
```

> 注意: 使用 SDK_INT 可防止旧版系统以这种方式仅在 Android 2.0（API 级别 5）和更高级别执行新 API 工作。较旧版本会遇到运行时异常。

一旦 onCreate() 完成执行操作，系统会相继调用 onStart() 和 onResume() 方法。您的 Activity 从不会驻留在“已创建”或“已开始”状态。在技术上，Activity 会在 onStart() 被调用时变得可见，但紧接着是 onResume()，且 Activity 保持“继续”状态，直到有事情发生使其发生变化，比如当接听来电时，用户导航至另一个 Activity，或设备屏幕关闭。

在接下来的其他课程中，您将看到其他启动方法（onStart() 和 onResume()）在用于从“暂停”或“停止”状态继续 Activity 时如何在您的 Activity 生命周期中发挥作用。

> 注: onCreate() 方法包括一个称为 savedInstanceState 的参数，将在有关重新创建 Activity的后续课程中讨论该参数。

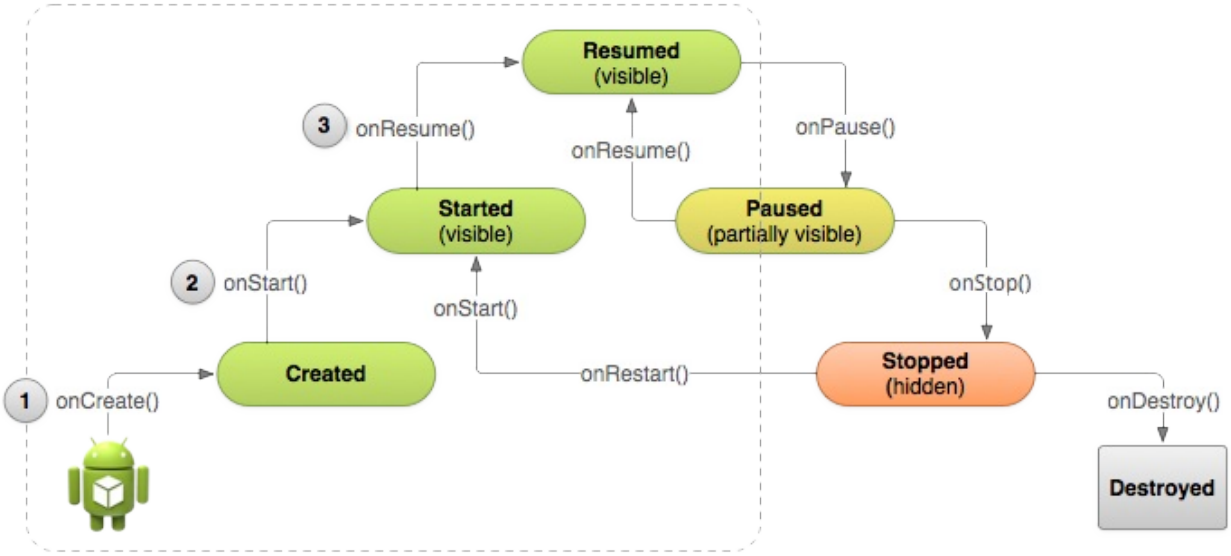


图 2. Activity 生命周期结构的另一个图示，其重点放在创建 Activity 的新实例时系统依次调用的三大回调上:onCreate()、onStart() 和 onResume()。一旦这一系列回调完成，Activity 就进入“继续”状态，此时用户可与 Activity 进行交互，直至用户切换到其他 Activity。

销毁 Activity 当 Activity 的第一个生命周期回调是 onCreate() 时，它最近的回调是 onDestroy()。系统会对您的 Activity 调用此方法，作为您的 Activity 实例完全从系统内存移除的最终信号。

大多数应用不需要实现此方法，因为局部类引用与 Activity 一同销毁，并且您的 Activity 应在 onPause() 和 onStop() 期间执行大多数清理操作。但是，如果您的 Activity 包含您在 onCreate() 期间创建的后台线程或其他如若未正确关闭可能导致内存泄露的长期运行资源，您应在 onDestroy() 期间终止它们。

```
@Override
public void onDestroy() {
    super.onDestroy();  // Always call the superclass

    // Stop method tracing that the activity started during onCreat
    android.os.Debug.stopMethodTracing();
}
```

> 注: 在所有情况下，系统在调用 onPause() 和 onStop() 之后都会调用 onDestroy()，只有一个例外：当您从 onCreate() 方法内调用 finish() 时。在某些情况下，比如当您的 Activity 作为临时决策工具运行以启动另一个 Activity 时，您可从 onCreate() 内调用 finish() 来销毁 Activity。在这种情况下，系统会立刻调用 onDestroy()，而不调用任何其他 生命周期方法。

暂停和继续 Activity

在使用应用期间，应用有时会失去焦点，同时会导致 Activity 暂停。例如，当应用运行在 多窗口模式 下，只会有一个应用拥有焦点，其它应用都会被系统暂停。同样的，当一个透明的 Activity 打开（如：对话框），被覆盖的 Activity 将会暂停，在 Activity 被挡住时会一直没有焦点，它保持暂停状态。

然而，一旦 Activity 被完全挡住并且不可见，它会是 停止 状态（在下一课讨论）。

当窗体进入到暂停状态，系统会回调 Activity 的 onPause()) 方法，在这个方法中，你可以停止不应该在暂停时继续运行的操作，或持久化相关信息在用户离开应用时。如果用户在 Activity 暂停状态下返回，系统会恢复 Activity 并调用 onResume()) 方法。

> 注：在系统调用 Activity 的 onPause()) 方法时，可能表示 Activity 会暂停一会之后再次获取焦点，或应用在多窗口模式；也有可能这个方法被调用后用户退出了 Activity。

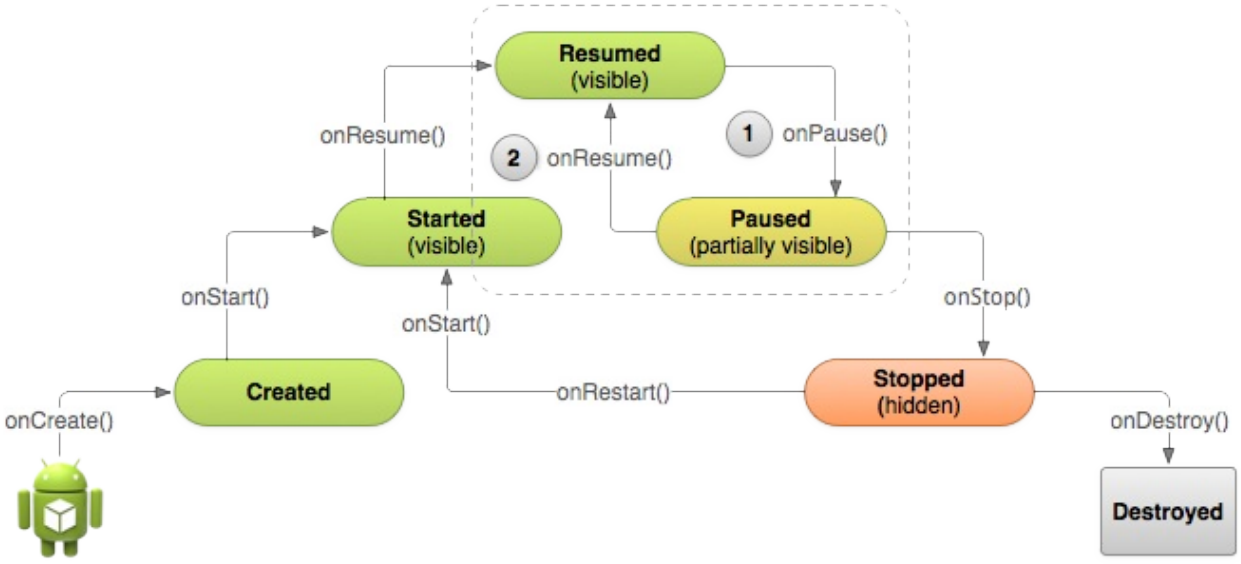


图**1**，当半透明的窗体挡住 Activity时，系统会回调 onPause()) 方法，这时 Activity 保持在 onPause()① 状态，当用户返回时系统回调 onResume())②。

暂停 Activity

当系统调用 Activity 的 onPause()) 时，从技术上来说它是部分可见。但是大多数情况下是用户将要离开 Activity，马上会变成停止状态。你通常应该使用 onPause()) 回调：

- 检查 Activity 是否可见。如果不可见，请停止动画或其他正在进行的可能消耗 CPU 的操作。记住，从 Android 7.0 开始，暂停的应用可能会在多窗口模式下运行。 在本示例中，你可能不想停止动画或视频播放。
- 释放系统资源，比如广播接收器、传感器手柄（如 GPS），或当你的 Activity 暂停时用户不需要它们且可能影响电池寿命的其他资源。

例如，如果你使用 Camera，onPause()) 方法是释放它的好位置。

```
@Override
public void onPause() {
    super.onPause();  // 第一步总是回调父类的方法

    // 释放相机资源，因为在暂停时不需要它
    // 并且其它 Activity 可能需要使用它
    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
    }
}
```

一般情况下，你不应该使用 onPause()) 永久性存储用户更改（比如输入表单的个人信息）。 只有在你确定用户希望自动保存这些更改的情况（比如，草拟电子邮件时）下，才能在 onPause()) 中永久性存储用户更改。但你应避免在 onPause()) 期间执行 CPU 密集型工作，比如向数据库写入信息，因为这会拖慢向下一 Activity 过渡的过程（你应改为在 onStop()) 间执行高负载关机操作）。

你应通过相对简单的方式在 onPause()) 方法中完成大量操作，这样才能加快在你的 Activity 确实停止的情况下用户向下一个目标过渡的速度。

> 注：当你的 Activity 暂停时，Activity 实例将驻留在内存中并且在 Activity 继续时被再次调用。你无需重新初始化在执行任何导致进入“继续”状态的回调方法期间创建的组件。

继续 Activity

当用户从“暂停”状态回到 Activity 时，系统会调用 onResume()) 方法。

请注意，每当你的 Activity 进入前台时系统便会调用此方法，包括它初次创建之时。 同样地，你应实现 onResume()) 以初始化你在 onPause()) 期间释放的组件，并执行每当 Activity 进入“继续”状态时必须进行的任何其他初始化操作（比如开始动画和初始化只在 Activity 具有用户焦点时使用的组件）。

onResume()) 的以下示例对应于以上的 onPause()) 示例，因此它初始化 Activity 暂停时释放的照相机。

```
@Override
public void onResume() {
    super.onResume();  // 第一步总是回调父类的方法

    // 获取相机实例
    if (mCamera == null) {
        initializeCamera(); // 调用本地方法初始化相机
    }
}
```

> 翻译：@iOnesmile
> 原始文档：https://developer.android.com/training/basics/activity-lifecycle/pausing.html

停止和重启 Activity

正确停止和重启 Activity 是 Activity 生命周期中的重要过程，其可确保你的用户知晓应用始终保持活跃状态并且不会丢失进度。Activity 停止和重启的场景主要有以下几种：

- 用户打开“最近应用”窗口并从你的应用切换到另一个应用。你的应用中当前位于前台的 Activity 将停止。 如果用户从主屏幕启动器图标或“最近应用”窗口返回到你的应用，将会重启 Activity。
- 用户在你的应用中执行开始新 Activity 的操作。当第二个 Activity 创建好后，当前 Activity 便停止。 如果用户之后按了返回按钮，第一个 Activity 会重新开始。
- 用户在其手机上使用你的应用的同时接听来电。

Activity 类提供两种生命周期方法： onStop()) 和 onRestart())，这些方法允许你专门处理 Activity 句柄停止和重新启动的方式。 不同于识别部分 UI 阻挡的暂停状态，停止状态保证 UI 不再可见，且用户的焦点在另外的 Activity（或完全独立的应用）中。

> 注：因为系统在 Activity 停止时会将你的 Activity 实例保留在系统内存中，你根本无需实现 onStop()) 和 onRestart()) 或甚至 onStart()) 方法。对于大多数相对简单的 Activity 而言，Activity 将停止并重新启动，并且你可能只需使用 onPause()) 暂停正在进行的操作，并从系统资源断开连接。

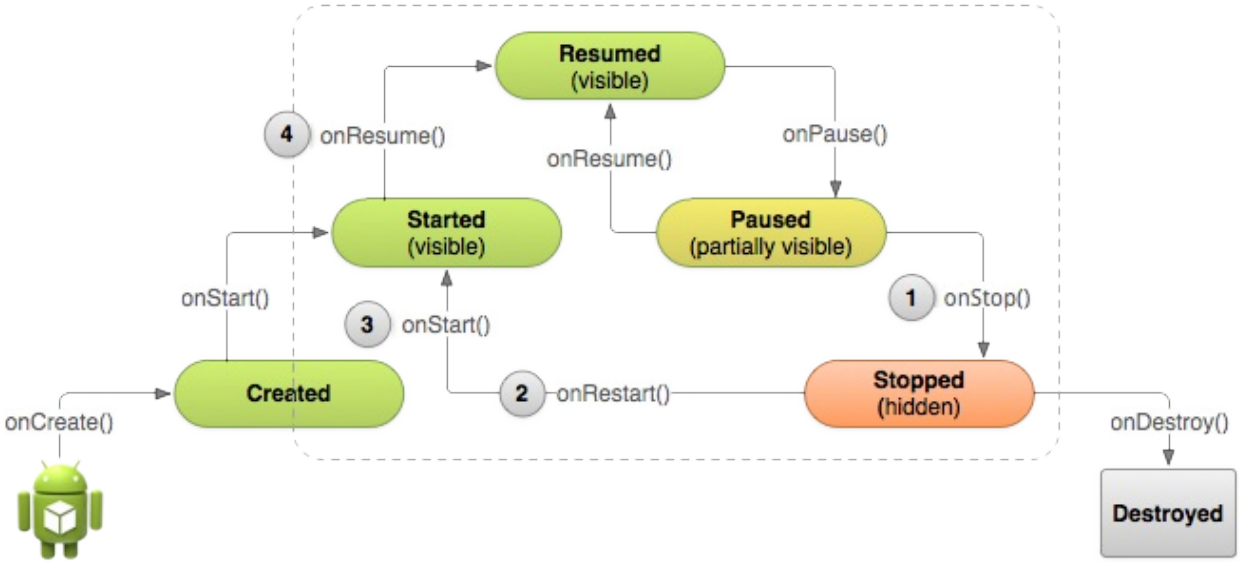


图 1. 用户离开 Activity 时，系统会调用 onStop()) 停止 Activity (1)。如果用户在 Activity 停止时返回，系统会调用 onRestart()) (2)，紧接着调用 onStart()) (3) 和 onResume()) (4)。请注意，无论什么场景导致 Activity 停止，系统始终会在调用 onStop()) 之前调用 onPause())。

停止 Activity

当你的 Activity 收到 onStop() 方法的调用时，它不再可见，并且应释放几乎所有用户不使用时不需要的资源。 一旦你的 Activity 停止，如果需要恢复系统内存，系统可能会销毁该实例。 在极端情况下，系统可能会仅终止应用进程，而不会调用 Activity 的最终 onDestroy()) 回调，因此你使用 onStop() 释放可能泄露内存的资源非常重要。

尽管 onPause() 方法在 onStop()之前调用，你应使用 onStop() 执行更大、占用更多 CPU 的关闭操作，比如向数据库写入信息。

例如，此处是将草稿笔记内容保存在永久存储中的 onStop() 的实现：

```
@Override
protected void onStop() {
    super.onStop();  // Always call the superclass method first

    // Save the note's current draft, because the activity is stopp
    // and we want to be sure the current note progress isn't lost.
    ContentValues values = new ContentValues();
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText()
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle

    getContentResolver().update(
        mUri,    // The URI for the note to update.
        values,  // The map of column names and new values to apply
        null,    // No SELECT criteria are used.
        null     // No WHERE columns are used.
        );
}
```

当你的 Activity 停止时， Activity 对象将驻留在内存中并在 Activity 继续时被再次调用。 你无需重新初始化在任何导致进入“继续”状态的回调方法过程中创建的组件。系统还会在布局中跟踪每个 View 的当前状态，如果用户在 EditText 小部件中输入文本，该内容会保留，因此你无需保存即可恢复它。

> 注：即使系统在 Activity 停止时销毁了 Activity，它仍会保留 Bundle（键值对的二进制大对象）中的 View 对象（比如 EditText 中的文本），并在用户导航回 Activity 的相同实例时恢复它们（下一堂课讲述更多有关在你的 Activity 被销毁且重新创建的情况下使用 Bundle 保存其他数据状态的知识）。

启动/重启 Activity

当你的 Activity 从停止状态返回前台时，它会接收对 onRestart() 的调用。系统还会在每次你的 Activity 变为可见时调用 onStart() 方法（无论是正重新开始还是初次创建）。但是，只会在 Activity 从停止状态继续时调用 onRestart() 方法，因此你可以使用它执行只有在 Activity 之前停止但未销毁的情况下可能必须执行的特殊恢复工作。

应用需要使用 onRestart() 恢复 Activity 状态的情况并不常见，因此没有适用于一般应用群体的任何方法指导原则。 但是，因为你的 onStop() 方法应基本清理所有 Activity 的资源，你将需要在 Activity 重新开始时重新实例化它们。 但是，你还需要在你的 Activity 初次创建时重新实例化它们（没有 Activity 的现有实例）。 出于此原因，你应经常使用 onStart() 回调方法作为 onStop() 方法的对应部分，因为系统会在它创建你的 Activity 以及从停止状态重新启动 Activity 时调用 onStart()。

例如，因为用户可能在回到它之前已离开应用很长时间，onStart() 方法是验证所需系统功能是否已启用的理想选择：

```
@Override
protected void onStart() {
    super.onStart();  // Always call the superclass method first

    // The activity is either being restarted or started for the fi
    // so this is where we should make sure that GPS is enabled
    LocationManager locationManager =
            (LocationManager) getSystemService(Context.LOCATION_SER
    boolean gpsEnabled = locationManager.isProviderEnabled(Locatior

    if (!gpsEnabled) {
        // Create a dialog here that requests the user to enable GP
        // with the android.provider.Settings.ACTION_LOCATION_SOURC
        // to take the user to the Settings screen to enable GPS wh
    }
}

@Override
protected void onRestart() {
    super.onRestart();  // Always call the superclass method first

    // Activity being restarted from stopped state
}
```

当系统销毁你的 Activity 时，它会调用你的 Activity 的 onDestroy() 方法。因为你通常应已使用 onStop() 释放大多数你的资源，到你接收对 onDestroy() 的调用时，大多数应用无需做太多操作。此方法是你清理可导致内存泄露的资源的最后一种方

法，因此你应确保其他线程被销毁且其他长期运行的操作（比如方法跟踪）也会停止。

> 备注:
> 翻译 ：@jarylan
> 原始文档:https://developer.android.com/training/basics/activity-lifecycle/stopping.html#Start

重新创建 Activity

有些情况下，您的 Activity 会因正常应用行为而销毁，比如当用户按 返回按钮或您的 Activity 通过调用 finish()示意自己的销毁。 如果 Activity 当前被停止或长期未使用，或者前台 Activity 需要更多资源以致系统必须关闭后台进程恢复内存，系统也可能会销毁 Activity。

当您的 Activity 因用户按了返回 或 Activity 自行完成而被销毁时，系统的 Activity 实例概念将永久消失，因为行为指示不再需要 Activity。 但是，如果系统因系统局限性（而非正常应用行为）而销毁 Activity，尽管 Activity 实际实例已不在，系统会记住其存在，这样，如果用户导航回实例，系统会使用描述 Activity 被销毁时状态的一组已保存数据创建 Activity 的新实例。 系统用于恢复先前状态的已保存数据被称为“实例状态”，并且是 Bundle 对象中存储的键值对集合。

注意：每次用户旋转屏幕时，您的 **Activity** 将被销毁并重新创建。 当屏幕方向变化时，系统会 销毁并重新创建前台 **Activity**，因为屏幕配置已更改并且您的 **Activity** 可能需要加载备用资源 **(**比如布局**)**。

默认情况下，系统会使用 Bundle 实例状态保存您的 Activity 布局（比如，输入到 EditText 对象中的文本值）中有关每个 View 对象的信息。 这样，如果您的 Activity 实例被销毁并重新创建，布局状态便恢复为其先前的状态，且您无需代码。 但是，您的 Activity 可能具有您要恢复的更多状态信息，比如跟踪用户在 Activity 中进度的成员变量。

注：为了 **Android** 系统恢复 **Activity** 中视图的状态，每个视图必须具有 **android:id**属性提供的唯一 **ID**。

要保存有关 Activity 状态的其他数据，您必须替代 onSaveInstanceState() 回调方法。当用户要离开 Activity 并在 Activity 意外销毁时向其传递将保存的 Bundle 对象时，系统会调用此方法。 如果系统必须稍后重新创建 Activity 实例，它会将相同的 Bundle 对象同时传递给 onRestoreInstanceState() 和 onCreate() 方法。

图 2. 当系统开始停止您的 Activity 时，它会调用 onSaveInstanceState()(1)，因此，您可以指定您希望在 Activity 实例必须重新创建时保存的额外状态数据。如果 Activity 被销毁且必须重新创建相同的实例，系统将在 (1) 中定义的状态数据同时传递给 onCreate() 方法 (2) 和 onRestoreInstanceState() 方法 (3)。

保存 **Activity** 状态

当您的 Activity 开始停止时，系统会调用 onSaveInstanceState() 以便您的 Activity 可以保存带有键值对集合的状态信息。 此方法的默认实现保存有关 Activity 视图层次的状态信息，例如 EditText 小部件中的文本或ListView 的滚动位置。

要保存 Activity 的更多状态信息，您必须实现 onSaveInstanceState() 并将键值对添加至 Bundle 对象。 例如：

```
    static final String STATE_SCORE = "playerScore";
    static final String STATE_LEVEL = "playerLevel";
    ...

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        // Save the user's current game state
        savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
        savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

        // Always call the superclass so it can save the view hiera
        super.onSaveInstanceState(savedInstanceState);
    }
```

注意：始终调用 **onSaveInstanceState()** 的超类实现，以便默认实现可以保存视图层次结构的状态。

恢复 Activity 状态 当您的 Activity 在先前销毁之后重新创建时，您可以从系统向 Activity 传递的 Bundle 恢复已保存的状态。onCreate() 和 onRestoreInstanceState() 回调方法均接收包含实例状态信息的相同 Bundle。

因为无论系统正在创建 Activity 的新实例还是重新创建先前的实例，都会调用 onCreate() 方法，因此您必须在尝试读取它之前检查状态 Bundle 是否为 null。 如果为 null，则系统将创建 Activity 的新实例，而不是恢复已销毁的先前实例。

例如，此处显示您如何可以在 onCreate() 中恢复一些状态数据：

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // Always call the supe

        // Check whether we're recreating a previously destroyed in
        if (savedInstanceState != null) {
            // Restore value of members from saved state
            mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
            mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
        } else {
            // Probably initialize members with default values for
        }
        ...
    }
```

您可以选择实现系统在 onStart() 方法之后调用的 onRestoreInstanceState()，而不是在 onCreate() 期间恢复状态。系统只在存在要恢复的已保存状态时调用 onRestoreInstanceState()，因此您无需检查 Bundle 是否为 null：

```
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        // Always call the superclass so it can restore the view hi
        super.onRestoreInstanceState(savedInstanceState);

        // Restore state members from saved instance
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    }
```

注意：始终调用 **onRestoreInstanceState()** 的超类实现，以便默认实现可以恢复视图层次结构的状态。

要了解更多有关因运行时重启事件（例如屏幕旋转时）而重新创建 Activity 的信息，请阅读处理运行时变更。

使用 Fragment 构建动态的 UI

为了在 Android 上为用户提供动态的、多窗口的交互体验，需要将 UI 组件和 Activity 操作封装成模块进行使用，这样我们就可以在 Activity 中对这些模块进行切入切出操作。可以用 Fragment 创建这些模块，Fragment 就像一个嵌套的 Activity，拥有自己的布局（Layout）并管理自己的生命周期。

Fragment 定义了自己的布局后，它可以在 Activity 中与其他 Fragment 生成不同的组合，从而为不同的屏幕尺寸生成不同的布局（小屏幕一次也许只能显示一个 Fragment，大屏幕则可以显示更多）。

本章将展示如何用 Fragment 创建动态界面，并在不同屏幕尺寸的设备上优化 APP 的用户体验。本章内容支持 Android 1.6 以上的设备。

课程

创建 Fragment

学习如何创建 Fragment，以及实现其生命周期内的基本功能。

构建灵活的 UI

学习如何针对不同的屏幕尺寸用 Fragment 构建不同的布局。

与其他 Fragment 交互

学习如何在 Fragment 与 Activity 或多个 Fragment 间进行交互。

创建 Fragment

你可以把 fragment 当做 activity 的一个组成模块，它有自己的生命周期并接收输入事件，当 activity 运行时可以进行添加或者删除(类似一个"子 activity"，可以在不同的activity 中重复使用)。这个课程主要说明如何继承 Support Library 中的 Fragment ，从而使 APP 可以兼容到系统为Android 1.6的低版本。

在开始学习这个课程之前，必须在 Android 项目中先引用 Support Library。如果你之前未使用过Support Library,可以参考 Support Library Setup 文档使用 v4 库。当然你也可以使用包含 app bar的 **v7 appcompat** 库。该库兼容了 Android 2.1 (API level 7)，也包含了 Fragment API。

创建 Fragment 类

创建一个 fragment时，需要继承 Fragment 类，然后结合 APP 的逻辑去重写关键的生命周期方法，Activity 也是类似这种处理方式。
其中一个区别是：当创建一个 Fragment时，必须重写 onCreateView()) 回调去定义布局，实际上这是唯一一个为了使 fragment 运行起来的回调。例如下面这个自定义布局的示例 fragment：

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup co
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container,
    }
}
```

和 Activity 一样，当 Fragment 从 Activity 添加或者移除、或 Activity 生命周期发生变化时，Fragment 通过生命周期回调函数管理其状态。例如，当 Activity 的 onPause()) 被调用时，它内部所有 Fragment 的 onPause() 方法也会被触发。

更多关于 Fragment 的声明周期和回调方法，详见 Fragments 开发指南。

使用 XML 在 Activity 中添加 Fragment

fragments 是可重用、可模块化的 UI 组件，每一个 fragment 实例都必须关联一个父 FragmentActivity，你可以在 Activity 的 XML 布局文件中逐个定义 Fragment 来实现这种关联。

> 注: FragmentActivity 是 Support Library 提供的一种特殊 Activity，用于处理 API 11 版本以下的 Fragment。如果 APP 中的最低版本大于等于 11，则可以使用普通的 Activity。

这里有一个布局文件的例子，即当手机屏幕属于“large”时（用目录名称中的 large 字符来区分），在一个 activity 中添加二个 fragment。

```
res/layout-large/news_articles.xml
```

```
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

    <fragment android:name="com.example.android.fragments.Headlines
              android:id="@+id/headlines_fragment"
              android:layout_weight="1"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

    <fragment android:name="com.example.android.fragments.ArticleFr
              android:id="@+id/article_fragment"
              android:layout_weight="2"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

    </LinearLayout>
```

> 提示：更多关于不同屏幕尺寸不同布局的信息，请阅读兼容不同屏幕尺寸。

然后将这个布局文件用到 Activity 中：

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);
    }
}
```

如果使用 v7 appcompat library，应该使 activity 继承 AppCompatActivity，它是 FragmentActivity 的子类。更多信息请参阅 Adding the App Bar。

> 注:当通过定义 XML 布局文件在 activity 中添加一个 fragment 时，你不能动态的移除 fragment。如果想在用户交互时对 fragment 进行切入和切出，必须在 activity 第一次启动时就把 fragment 添加进来，这些将在下节课讲解。·
> 翻译：@misparking
> 原始文档：https://developer.android.com/training/basics/fragments/creating.html

创建灵活的 UI

当你的应用设计成支持较大范围的屏幕尺寸的时候，你可以通过不同布局配置的 Fragment 来优化基于可用的屏幕空间的用户体验。

例如，在一个手持设备上，针对一个单窗口 UI ，一次显示一个 Fragment ，可能是正确的。相反,你可以在一个拥有更宽屏幕尺寸的平板电脑上，并排的设置多个 Fragment，这样可以展示更多信息给用户。

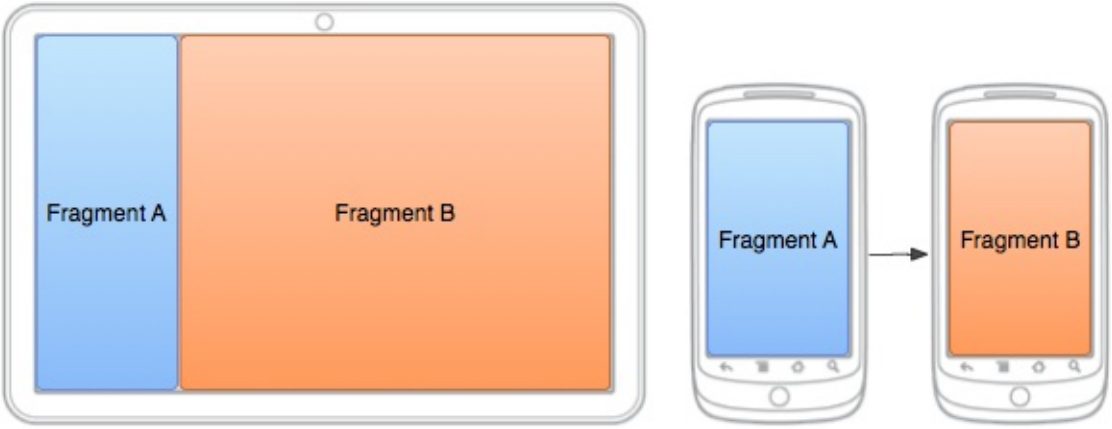


图 1.在相同 Activity 不同屏幕尺寸上，通过不同的配置来显示。在大屏幕上，两个 Fragment 并排展示，但是在手持设备上，一次只展示一个 Fragment，当用户到另外一个 Fragment 的时候，当前 Fragment 会被替换。

FragmentManager 类提供了往运行中的 Activity 中添加、移除、替换 Fragment 的方法，这些能够让你创建一个动态的体验。

在运行的 Activity 中添加 Fragment

除了之前课程中提到的可以在 Activity 的布局文件中通过 `<fragment>` 元素来定义 Fragment 在 Activity 运行时添加一个 Fragment。如果你想要在 Activity 的生命周期中修改 Fragment，那么这个是必要的。

执行诸如添加和移除 Fragment 的事务，你应当使用 FragmentManager 来创建一个 FragmentTransaction ，它提供了添加、移除、替换和执行其它 Fragment 事务的 API。

如果你的 Activity 允许移除和替换 Fragment，你应当在 Activity 的 onCreate()) 方法中，添加 Fragment 的初始化。

在处理 Fragment 的时候，尤其是在运行时添加 Fragment，有一个重要的规则，就是你的 Activity 的布局必须包含一个你能够插入一个 Fragment 的 View 容器。

以下布局是之前课程中的一次添加一个 Fragment 的备选。为了替换 Fragment，Activity 中的布局包含了一个空的 FrameLayout ，作为 Fragment 的容器。

请注意，文件名和之前课程中讲的布局文件相同，但是这个布局目录没有 `large` 标识，所以这个布局是在设备屏幕尺寸小于 `large` 的时候使用，因为这种屏幕不能同时容下两个 Fragment。

`res/layout/news_articles.xml:`

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/andr
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

在你的 Activity 里面，通过使用支持库 API 调用 getSupportFragmentManager()) 方法来获得一个 FragmentManager 对象。然后调用 beginTransaction()) 方法来创建一个 FragmentTransaction 对象，最后通过调用 add()) 方法来添加一个 Fragment。

你可以通过相同的 FragmentTransaction 对象在 Activity 中执行多个 Fragment 事务。当你准备好修改的时候，你需要调用 commit()) 方法。

For example, here's how to add a fragment to the previous layout:

例如，下面有一个例子展示如何在初始布局中添加一个 Fragment：

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);

        // Check that the activity is using the layout version with
        // the fragment_container FrameLayout
        if (findViewById(R.id.fragment_container) != null) {

            // However, if we're being restored from a previous sta
            // then we don't need to do anything and should return
            // we could end up with overlapping fragments.
            if (savedInstanceState != null) {
                return;
            }

            // Create a new Fragment to be placed in the activity l
            HeadlinesFragment firstFragment = new HeadlinesFragment

            // In case this activity was started with special instr
            // Intent, pass the Intent's extras to the fragment as
            firstFragment.setArguments(getIntent().getExtras());

            // Add the fragment to the 'fragment_container' FrameLa
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.fragment_container, firstFragment).co
        }
    }
}
```

因为这个 Fragment 是在运行时添加到 FrameLayout 容器中的，而不是通过 Activity 的布局中的 `<fragment>` 元素添来定义的，所以这个 Activity 可以通过替换当前 Fragment 来追加新的 Fragment。

Fragment 的替换

Fragment 的替换流程和添加流程差不多，只是不是调用 add()) 方法，而是 replace()) 方法。

Keep in mind that when you perform fragment transactions, such as replace or remove one, it's often appropriate to allow the user to navigate backward and "undo" the change. To allow the user to navigate backward through the fragment transactions, you must call addToBackStack() before you commit the FragmentTransaction.

请记住，当你执行 Fragment 事务的时候，例如替换和移除一个 Fragment，通常采取的正确方法是允许用户向后或者撤销操作。为了允许用户向后操作，你应当在你提交 FragmentTransaction 之前，调用 addToBackStack()) 方法。

> 备注：当你移除或者替换一个 Fragment 的，并且是在回退栈中添加了事务的话，这个 Fragment 是被移除和停止了（但是并没有被销毁掉）。当用户回退到这个 Fragment 的时候，它有重启了。如果你没有将这个事务添加到回退栈中，这个 Fragment 在被移除或者替换的时候，被销毁了。

以下是 Fragment 替换的例子：

```
// Create fragment and give it an argument specifying the article i
ArticleFragment newFragment = new ArticleFragment();
Bundle args = new Bundle();
args.putInt(ArticleFragment.ARG_POSITION, position);
newFragment.setArguments(args);

FragmentTransaction transaction = getSupportFragmentManager().begin

// Replace whatever is in the fragment_container view with this fra
// and add the transaction to the back stack so the user can naviga
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

addToBackStack()) 方法提供了一个可选的 String 参数来指定事务的唯一名字。这个名字除非你通过 FragmentManager.BackStackEntry 的 API 来执行更高级的 Fragment 的操作，否则你是不需要的。

备注：
翻译：@ifeegoo
原始文档：https://developer.android.com/training/basics/fragments/fragment-ui.html

Fragments 之间的交互

为了重用 Fragment UI 组件，你应该建立一个完全独立的，模块化的组件，定义自己的布局和行为。一次定义可重用的 Fragments，你可以使其和 Activity 关联，并且在应用逻辑连接他们从而实现整体的复合界面。

通常你想让一个 Fragment 和另一个去交互，比如根据用户事件改变内容。所有 Fragment 和 Fragment 之间的交互是通过关联的 Activity 。两个 Fragments 不应该交互的。

定义一个接口

允许 Fragment 和关联的 Activity 交互，你可以定义一个接口在 Fragment 类并且在 Activity 中实现这个接口。这个 Fragment 在生命周期 onAttch() 方法获得这个接口的实现，然后可以调用接口方法与Activity交互。

这里是一个 Framengt 和 Activity 交互的例子：

```
public class HeadlinesFragment extends ListFragment {
    OnHeadlineSelectedListener mCallback;

    // Container Activity must implement this interface
    public interface OnHeadlineSelectedListener {
        public void onArticleSelected(int position);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        // This makes sure that the container activity has implemer
        // the callback interface. If not, it throws an exception
        try {
            mCallback = (OnHeadlineSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                    + " must implement OnHeadlineSelectedListener")
        }
    }

    ...
}
```

现在这个 Fragment 可以使用 `OnHeadlineSelectedListener` 接口的实例 `mCallback` 去调用 `onArticleSelected()` 方法(或者接口中其他方法)传递消息。例如，当用户单击列表项时，fragment 会调用下面的方法。这个 Fragment 使用回调接口去传递事件给父 Activity。

```
@Override
    public void onListItemClick(ListView l, View v, int position, I
        // Send the event to the host activity
        mCallback.onArticleSelected(position);
    }
```

实现这个接口

为了接收来自 Frament 的回调事件，activity 必须去实现这个在 Fragment 类中定义的接口。 例如，下面这个 Activity 实现了上面例子的接口：

```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the He
        // Do something here to display that article
    }
}
```

传递消息给一个 **Fragment**

这个主Activity可以通过 findFragmentById) 获得 Fragment 实例去传递消息给这个 fragment。然后直接调用 fragment 中的公开方法。

例如，假设上面 activity 可能包含其他 fragment 用于去显示上面回调方法返回的数据在指定选项中。在这个情况，这个 activity 可以通过回调方法接受的消息传递给其他要显示选项的 fragment：

```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the He
        // Do something here to display that article

        ArticleFragment articleFrag = (ArticleFragment)
                getSupportFragmentManager().findFragmentById(R.id.a

        if (articleFrag != null) {
            // If article frag is available, we're in two-pane layo

            // Call a method in the ArticleFragment to update its c
            articleFrag.updateArticleView(position);
        } else {
            // Otherwise, we're in the one-pane layout and must swa

            // Create fragment and give it an argument for the sele
            ArticleFragment newFragment = new ArticleFragment();
            Bundle args = new Bundle();
            args.putInt(ArticleFragment.ARG_POSITION, position);
            newFragment.setArguments(args);

            FragmentTransaction transaction = getSupportFragmentMar

            // Replace whatever is in the fragment_container view w
            // and add the transaction to the back stack so the use
            transaction.replace(R.id.fragment_container, newFragmer
            transaction.addToBackStack(null);

            // Commit the transaction
            transaction.commit();
        }
    }
}
```

> 翻译：@edtj
> 原始文档：https://developer.android.google.cn/training/basics/fragments/communicating.html

保存键值对

如果你需要保存较小的键值对集合，那么你可以使用 SharedPreferences 的 API。每个 SharedPreferences 对象会对应一个键值对文件，并且提供了简单的方法去读写它们。每个 SharedPreferences 被 Framework 管理，能够设置为私有的或公开的。

这一课将讲解用 SharedPreferences 的 API 如何存储和恢复简单类型的值。

> 注：SharedPreferences API 仅仅是用来读写键值对数据，请不要与 Preference API 混淆，它是用来构建应用设置界面的（尽管它会使用 SharedPreferences 来储存设置信息）。关于使用 Preference API 的信息，请参见 Settings 手册。

获取 **SharedPreferences** 的实例

你可以创建一个新的 SharedPreferences 文件，或打开一个已经存在的文件，有两个方法可用：

- getSharedPreferences()) —— 通过 SharedPreferences 的文件标识名来使用它，第一个参数表示它的文件名。你能在应用的任何 Context 中获取到这个方法。
- getPreferences()) —— 使用它在 Activity 中，如果你仅需要使用这个 Activity 的 SharedPreferences 文件。因为它默认会返回属于这个 Activity 的 SharedPreferences 文件，你不需要传入它的名字。

例如：下面的代码是执行在 Fragment 中，它打开 SharedPreferences 文件的标识名是 string 资源文件里 *R.string.preference_file_key* 的值，并且是设置私有模式，所以这个文件仅仅只能在当前应用中调用。

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
        getString(R.string.preference_file_key), Context.MODE_PRIVA
```

在命名 SharedPreferences 文件时，你应该使用在当前应用中的唯一的标识名，如：

"com.example.myapp.PREFERENCE_FILE_KEY"

另一种选择，你仅仅需要使用 Activity 中的 getPreferences() 方法来获取 SharedPreferences 文件：

```
SharedPreferences sharedPref = getActivity().getPreferences(Context
```

> 注意：如果你创建的 SharedPreferences 文件是 MODE_WORLD_READABLE 或 MODE_WORLD_WRITEABLE 模式，那么任何一个其它的应用只要知道你的标识名都能访问数据。

写 SharedPreferences 信息

在写 SharedPreferences 文件时，需要通过 edit()) 方法创建 SharedPreferences.Editor 对象。

通过 putInt()) 或 putString()) 等方法去写键值对数据，当调用 commit()) 方法时保存更改，如下：

```
SharedPreferences sharedPref = getActivity().getPreferences(Context
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score), newHighScore);
editor.commit();
```

读 SharedPreferences 信息

从 SharedPreferences 文件中读取数据时，调用如 getInt()) 或 getString()) 等方法，传一个你需要值的键，再传一个默认的值作为该键不存在时结果，如：

```
SharedPreferences sharedPref = getActivity().getPreferences(Context
int defaultValue = getResources().getInteger(R.string.saved_high_sc
long highScore = sharedPref.getInt(getString(R.string.saved_high_sc
```

> 翻译：@iOnesmile
> 原始文档：https://developer.android.google.cn/training/basics/data-storage/shared-preferences.html#WriteSharedPreference

保存文件

Android 使用与其他平台上基于磁盘的文件系统类似的文件系统。 本课程讲述如何使用 Android 文件系统通过 File API 读取和写入文件。

File 对象适合按照从开始到结束的顺序不跳过地读取或写入大量数据。 例如，它适合于图片文件或通过网络交换的任何内容。

本课程展示如何在你的应用中执行基本的文件相关任务。本课程假定你熟悉 Linux 文件系统的基础知识和 java.io 中的标准文件输入/输出 API。

选择内部或外部存储

所有 Android 设备都有两个文件存储区域："内部"和"外部"存储。这些名称在 Android 早期产生，当时大多数设备都提供内置的非易失性内存（内部存储），以及移动存储介质，比如微型 SD 卡（外部存储）。一些设备将永久性存储空间划分为"内部"和"外部"分区，即便没有移动存储介质，也始终有两个存储空间，并且无论外部存储设备是否可移动，API 的行为均一致。以下列表汇总了关于各个存储空间的实际信息。

> 内部存储：
>
> - 它始终可用。
> - 只有你的应用可以访问此处保存的文件。
> - 当用户卸载你的应用时，系统会从内部存储中移除你的应用的所有文件。
>
> 外部存储：
>
> - 它并非始终可用，因为用户可采用 USB 存储设备的形式装载外部存储，并在某些情况下会从设备中将其移除。
> - 它是全局可读的，因此此处保存的文件可能不受你控制地被读取。
> - 当用户卸载你的应用时，只有在你通过 getExternalFilesDir()) 将你的应用的文件保存在目录中时，系统才会从此处移除你的应用的文件。

当你希望确保用户或其他应用均无法访问你的文件时，内部存储是最佳选择。 对于无需访问限制以及你希望与其他应用共享或允许用户使用计算机访问的文件，外部存储是最佳位置。

> 注：在 Android N 之前，内部文件可以通过放宽文件系统权限让其他应用访问。而如今不再是这种情况。如果你希望让其他应用访问私有文件的内容，则你的应用可使用 FileProvider。 请参阅共享文件。
>
> 提示：尽管应用默认安装在内部存储中，但你可在你的清单文件中指定 android:installLocation 属性，这样你的应用便可安装在在外部存储中。当 APK 非常大且它们的外部存储空间大于内部存储时，用户更青睐这个选择。 如需了解详细信息，请参阅应用安装位置。

获取外部存储的权限

要向外部存储写入信息，你必须在你的清单文件中请求 WRITE_EXTERNAL_STORAGE 权限。

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNA
    ...
</manifest>
```

> 注意：目前，所有应用都可以读取外部存储，而无需特别的权限。 但这在将来版本中会进行更改。如果你的应用需要读取外部存储（但不向其写入信息），那么你将需要声明 READ_EXTERNAL_STORAGE 权限。要确保你的应用继续正常工作，你应在更改生效前声明此权限。
>
> ```
> <manifest ...>
>     <uses-permission android:name="android.permission.READ_EXTE
>     ...
> </manifest>
> ```
>
> 但是，如果你的应用使用 WRITE_EXTERNAL_STORAGE 权限，那么它也隐含读取外部存储的权限。

你无需任何权限，即可在内部存储中保存文件。 你的应用始终具有在其内部存储目录中进行读写的权限。

将文件保存在内部存储中

在内部存储中保存文件时，你可以通过调用以下两种方法之一获取作为 File 的相应目录：

```
getFilesDir()
    返回表示你的应用的内部目录的 File 。
getCacheDir()
返回表示你的应用临时缓存文件的内部目录的 File。 务必删除所有不再需要的文件并对
```

要在这些目录之一中新建文件，你可以使用 File() 构造函数，传递指定你的内部存储目录的上述方法之一所提供的 File。例如：

```
File file = new File(context.getFilesDir(), filename);
```

或者，你可以调用 openFileOutput() 获取写入到内部目录中的文件的 FileOutputStream。例如，下面显示如何向文件写入一些文本：

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;

try {
  outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
  outputStream.write(string.getBytes());
  outputStream.close();
} catch (Exception e) {
  e.printStackTrace();
}
```

或者，如果你需要缓存某些文件，你应改用 createTempFile()。例如，以下方法从 URL 提取文件名并正在你的应用的内部缓存目录中以该名称创建文件：

```
public File getTempFile(Context context, String url) {
    File file;
    try {
        String fileName = Uri.parse(url).getLastPathSegment();
        file = File.createTempFile(fileName, null, context.getCache
    } catch (IOException e) {
        // Error while creating file
    }
    return file;
}
```

> 注：你的应用的内部存储设备目录由你的应用在 Android 文件系统特定位置中的软件包名称指定。从技术上讲，如果你将文件模式设置为可读，那么，另一应用也可以读取你的内部文件。 但是，此应用也需要知道你的应用的软件包名称和文件名。 其他应用无法浏览你的内部目录并且没有读写权限，除非你明确将文件设置为可读或可写。 只要你为内部存储上的文件使用 MODE_PRIVATE，其他应用便从不会访问它们。

将文件保存在外部存储中

由于外部存储可能不可用—比如，当用户已将存储装载到电脑或已移除提供外部存储的 SD 卡时—因此，在访问它之前，你应始终确认其容量。 你可以通过调用 getExternalStorageState()) 查询外部存储状态。 如果返回的状态为 MEDIA_MOUNTED，那么你可以对你的文件进行读写。 例如，以下方法对于确定存储可用性非常有用：

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

尽管外部存储可被用户和其他应用进行修改，但你可在此处保存两类文件：

```
公共文件
    应供其他应用和用户自由使用的文件。 当用户卸载你的应用时，用户应仍可以使用这
    例如，你的应用拍摄的照片或其他已下载的文件。
私有文件
    属于你的应用且在用户卸载你的应用时应予删除的文件。 尽管这些文件在技术上可被
    例如，你的应用下载的其他资源或临时介质文件。
```

如果你要将公共文件保存在外部存储设备上，请使用 getExternalStoragePublicDirectory() 方法获取表示外部存储设备上相应目录的 File。该方法使用指定你想要保存以便它们可以与其他公共文件在逻辑上组织在一起的文件类型的参数，比如 DIRECTORY_MUSIC 或 DIRECTORY_PICTURES。例如：

```
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new File(Environment.getExternalStoragePublicDirect
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

如果你要保存你的应用专用文件，你可以通过调用 getExternalFilesDir() 并向其传递指示你想要的目录类型的名称，从而获取相应的目录。通过这种方法创建的各个目录将添加至封装你的应用的所有外部存储文件的父目录，当用户卸载你的应用时，系统会删除这些文件。

例如，你可以使用以下方法来创建个人相册的目录：

```
public File getAlbumStorageDir(Context context, String albumName) {
    // Get the directory for the app's private pictures directory.
    File file = new File(context.getExternalFilesDir(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

如果没有适合你文件的预定义子目录名称，你可以改为调用 getExternalFilesDir() 并传递 null。这将返回外部存储上你的应用的专用目录的根目录。

切记，getExternalFilesDir() 在用户卸载你的应用时删除的目录内创建目录。如果你正保存的文件应在用户卸载你的应用后仍然可用—比如，当你的应用是照相机并且用户要保留照片时—你应改用 getExternalStoragePublicDirectory()。

无论你对于共享的文件使用 getExternalStoragePublicDirectory() 还是对你的应用专用文件使用 getExternalFilesDir()，你使用诸如 DIRECTORY_PICTURES 的 API 常数提供的目录名称非常重要。这些目录名称可确保系统正确处理文件。 例如，保

存在 DIRECTORY_RINGTONES 中的文件由系统媒体扫描程序归类为铃声，而不是音乐。

查询可用空间

如果你事先知道你将保存的数据量，你可以查出是否有足够的可用空间，而无需调用 getFreeSpace() 或 getTotalSpace() 引起 IOException。这些方法分别提供目前的可用空间和存储卷中的总空间。 此信息也可用来避免填充存储卷以致超出特定阈值。

但是，系统并不保证你可以写入与 getFreeSpace() 指示的一样多的字节。如果返回的数字比你要保存的数据大小大出几 MB，或如果文件系统所占空间不到 90%，则可安全继续操作。否则，你可能不应写入存储。

> 注：保存你的文件之前，你无需检查可用空间量。 你可以尝试立刻写入文件，然后在 IOException 出现时将其捕获。 如果你不知道所需的确切空间量，你可能需要这样做。 例如，如果在保存文件之前通过将 PNG 图像转换成 JPEG 更改了文件的编码，你事先将不知道文件的大小。

删除文件

你应始终删除不再需要的文件。删除文件最直接的方法是让打开的文件参考自行调用 delete()。

```
myFile.delete();
```

如果文件保存在内部存储中，你还可以请求 Context 通过调用 deleteFile() 来定位和删除文件：

```
myContext.deleteFile(fileName);
```

> 注：当用户卸载你的应用时，Android 系统会删除以下各项：
>
> - 你保存在内部存储中的所有文件
> - 你使用 getExternalFilesDir() 保存在外部存储中的所有文件。
>
> 但是，你应手动删除使用 getCacheDir() 定期创建的所有缓存文件并且定期删除不再需要的其他文件。
>
> 备注:
> 翻译 ：@jarylan
> 原始文档:https://developer.android.com/training/basics/data-storage/files.html

在 SQL 数据库中保存数据

保存数据到数据库是理想的重复或结构化数据,如联系信息。 本课程假定您基本熟悉 SQL 数据库并且可帮助您开始在 Android 中使用 SQLite 数据库。 您在 Android 中使用数据库所需的 API 在 android.database.sqlite 软件包中提供。

定义架构和契约

SQL 数据库的主要原则之一是架构：数据库如何组织的正式声明。 架构体现于您用于创建数据库的 SQL 语句。您会发现它有助于创建伴随类，即契约类，其以一种系统性、自记录的方式明确指定您的架构布局。

契约类是用于定义 URI、表格和列名称的常数的容器。 契约类允许您跨同一软件包中的所有其他类使用相同的常数。 您可以在一个位置更改列名称并使其在您整个代码中传播。

组织契约类的一种良好方法是将对于您的整个数据库而言是全局性的定义放入类的根级别。 然后为枚举其列的每个表格创建内部类。

> 注：通过实现 BaseColumns 接口，您的内部类可继承名为 _ID 的主键字段，某些 Android 类（比如光标适配器）将需要内部类拥有该字段。 这并非必需项，但可帮助您的数据库与 Android 框架协调工作。

例如，该代码段定义了单个表格的表格名称和列名称：

```
public final class FeedReaderContract {
    // To prevent someone from accidentally instantiating the c
    // make the constructor private.
    private FeedReaderContract() {}

    /* Inner class that defines the table contents */
    public static class FeedEntry implements BaseColumns {
        public static final String TABLE_NAME = "entry";
        public static final String COLUMN_NAME_TITLE = "title";
        public static final String COLUMN_NAME_SUBTITLE = "subt
    }
}
```

使用 SQL 辅助工具创建数据库

在您定义了数据库的外观后，您应实现创建和维护数据库和表格的方法。 这里有一些典型的表格创建和删除语句：

```
private static final String TEXT_TYPE = " TEXT";
private static final String COMMA_SEP = ",";
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
    FeedEntry._ID + " INTEGER PRIMARY KEY," +
    FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE + COMMA_SEP +
    FeedEntry.COLUMN_NAME_SUBTITLE + TEXT_TYPE + " )";

private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE_NAME;
```

就像您在设备的内部存储中保存文件那样，Android 将您的数据库保存在私人磁盘空间，即关联的应用。 您的数据是安全的，因为在默认情况下，其他应用无法访问此区域。

SQLiteOpenHelper 类中有一组有用的 API。当您使用此类获取对您数据库的引用时，系统将只在需要之时而不是应用启动过程中执行可能长期运行的操作：创建和更新数据库。 您仅需调用 getWritableDatabase()) 或 getReadableDatabase()) 即

可。

> 注：由于它们可能长期运行，因此请确保您在后台线程中调用 getWritableDatabase() 或 getReadableDatabase()，比如使用 AsyncTask 或 IntentService。

要使用 SQLiteOpenHelper，请创建一个替换 onCreate()) 、 onUpgrade() 和 onOpen()) 回调方法的子类。您可能还希望实现 onDowngrade()，但这并非必需操作。

例如，下面是一个使用如上所示一些命令的 SQLiteOpenHelper 的实现：

```
    public class FeedReaderDbHelper extends SQLiteOpenHelper {
        // If you change the database schema, you must increment th
        public static final int DATABASE_VERSION = 1;
        public static final String DATABASE_NAME = "FeedReader.db";

        public FeedReaderDbHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(SQL_CREATE_ENTRIES);
        }
        public void onUpgrade(SQLiteDatabase db, int oldVersion, in
            // This database is only a cache for online data, so it
            // to simply to discard the data and start over
            db.execSQL(SQL_DELETE_ENTRIES);
            onCreate(db);
        }
        public void onDowngrade(SQLiteDatabase db, int oldVersion,
            onUpgrade(db, oldVersion, newVersion);
        }
    }
```

要访问您的数据库，请实例化 SQLiteOpenHelper 的子类：

```
    FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContex
```

将信息输入到数据库

通过将一个 ContentValues 对象传递至 insert() 方法将数据插入数据库：

```
    // Gets the data repository in write mode
    SQLiteDatabase db = mDbHelper.getWritableDatabase();

    // Create a new map of values, where column names are the keys
    ContentValues values = new ContentValues();
    values.put(FeedEntry.COLUMN_NAME_TITLE, title);
    values.put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle);

    // Insert the new row, returning the primary key value of the r
    long newRowId = db.insert(FeedEntry.TABLE_NAME, null, values);
```

insert() 的第一个参数即为表格名称。

第二个参数将指示框架在 ContentValues 为空（即，您没有 put 任何值）时执行哪些操作。如果指定列名称，则框架将插入一行并将该列的值设置为 null。如果指定 null（就像此代码示例中一样），则框架不会在没有值时插入行。

从数据库读取信息

要从数据库中读取信息，请使用 query() 方法，将其传递至选择条件和所需列。该方法结合 insert() 和 update 的元素，除非列列表定义了您希望获取的数据，而不是希望插入的数据。 查询的结果将在 Cursor 对象中返回给您。

```
    SQLiteDatabase db = mDbHelper.getReadableDatabase();

    // Define a projection that specifies which columns from the da
    // you will actually use after this query.
    String[] projection = {
        FeedEntry._ID,
        FeedEntry.COLUMN_NAME_TITLE,
        FeedEntry.COLUMN_NAME_SUBTITLE
        };

    // Filter results WHERE "title" = 'My Title'
    String selection = FeedEntry.COLUMN_NAME_TITLE + " = ?";
    String[] selectionArgs = { "My Title" };

    // How you want the results sorted in the resulting Cursor
    String sortOrder =
        FeedEntry.COLUMN_NAME_SUBTITLE + " DESC";

    Cursor c = db.query(
        FeedEntry.TABLE_NAME,                     // The table to c
        projection,                               // The columns to
        selection,                                // The columns fo
        selectionArgs,                            // The values for
        null,                                     // don't group th
        null,                                     // don't filter b
        sortOrder                                 // The sort order
        );
```

要查看游标中的某一行，请使用 Cursor 移动方法之一，您必须在开始读取值之前始终调用这些方法。 一般情况下，您应通过调用 moveToFirst()) 开始，其将“读取位置”置于结果中的第一个条目中。 对于每一行，您可以通过调用 Cursor 获取方法之一读取列的值，比如 getString()) 或 getLong())。对于每种获取方法，您必须传递所需列的索引位置，您可以通过调用 getColumnIndex()) 或 getColumnIndexOrThrow()) 获取。例如：

```
    cursor.moveToFirst();
    long itemId = cursor.getLong(
        cursor.getColumnIndexOrThrow(FeedEntry._ID)
    );
```

从数据库删除信息

要从表格中删除行，您需要提供识别行的选择条件。 数据库 API 提供了一种机制，用于创建防止 SQL 注入的选择条件。 该机制将选择规范划分为选择子句和选择参数。 该子句定义要查看的列，还允许您合并列测试。 参数是根据捆绑到子句的项进行测试的值。由于结果并未按照与常规 SQL 语句相同的方式进行处理，它不受 SQL 注入的影响。

```
    // Define 'where' part of query.
    String selection = FeedEntry.COLUMN_NAME_TITLE + " LIKE ?";
    // Specify arguments in placeholder order.
    String[] selectionArgs = { "MyTitle" };
    // Issue SQL statement.
    db.delete(FeedEntry.TABLE_NAME, selection, selectionArgs);
```

更新数据库

当您需要修改数据库值的子集时，请使用 update 方法。

更新表可将 insert() 的内容值语法与 delete() 的 where 语法相结合。

```
    SQLiteDatabase db = mDbHelper.getReadableDatabase();

    // New value for one column
    ContentValues values = new ContentValues();
    values.put(FeedEntry.COLUMN_NAME_TITLE, title);

    // Which row to update, based on the title
    String selection = FeedEntry.COLUMN_NAME_TITLE + " LIKE ?";
    String[] selectionArgs = { "MyTitle" };

    int count = db.update(
        FeedReaderDbHelper.FeedEntry.TABLE_NAME,
        values,
        selection,
        selectionArgs);
```

> 翻译：@JackWaiting
> 原始文档：https://developer.android.com/training/basics/data-storage/databases.html

使用户跳转到另一个应用

Android 中最重要的功能之一就是可以利用一个想要执行的意图，使用户可以从一个应用跳转至另一个应用。例如，如果有一个公司的地址并且想显示在地图上，你不需要创建一个 activity 来展示这个地图。只需要使用 Intent 发起一个请求来展示这个地址。Android 系统会启动能够展示地图的程序来展示此地址。

正如第一节课所讲：构建第一个应用时，我们必须使用 intents 去实现在不同的 activity 之间进切换。通常定义一个显式的 intent，它定义了需要启动的组件类名。然而，当想要有一个单独的应用去执行某个操作，如“查看地图”，则必须使用隐式 intent 了。

本课程主要讲解如何针对特定的操作创建一个隐式 intent，以及如何使用隐式intent 去启动在另一个应用中执行操作的 activity。

构建隐含 Intent

隐含 Intent 不声明要启动的组件的类名称，而是声明要执行的操作。 该操作指定您要执行的操作，比如查看、编辑、发送或 获取 某项。 Intent 通常还包含与操作关联的数据，比如您要查看的地址或您要发送的电子邮件消息。根据要创建的 Intent，数据可能是 Uri、多种其他数据类型之一，或 Intent 可能根本就不需要数据。

如果您的数据是 Uri，您可以使用一个简单的 Intent()) 构造函数来定义操作和数据。

例如，此处显示如何使用指定电话号码的 Uri 数据创建发起电话呼叫的 Intent：

```
Uri number = Uri.parse("tel:5551234");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```

当您的应用通过调用 startActivity()) 调用此 Intent 时，“电话”应用会发起向指定电话号码的呼叫。

这里有一些其他 Intent 及其操作和 Uri 数据对：

- 查看地图：

```
// Map point based on address
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+
// Or map point based on latitude/longitude
// Uri location = Uri.parse("geo:37.422219,-122.08364?z=14"); /
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

- 查看网页：

```
Uri webpage = Uri.parse("http://www.android.com");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```

其他类型的隐含 Intent 需要提供不同数据类型（比如，字符串）的“额外”数据。 您可以使用各种 putExtra()) 方法添加一条或多条 extra 数据。

默认情况下，系统基于所包含的 Uri 数据确定 Intent 需要的相应 MIME 类型。如果您未在 Intent 中包含 Uri，您通常应使用 setType()) 指定与 Intent 关联的数据的类型。 设置 MIME 类型可进一步指定哪些类型的 Activity 应接收 Intent。

此处有更多添加额外数据以指定所需操作的 Intent：

- 发送带附件的电子邮件：

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
// The intent does not have a URI, so declare the "text/plain"
emailIntent.setType(HTTP.PLAIN_TEXT_TYPE);
emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[] {"jon@exa
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email subject");
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message text");
emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("content://
// You can also attach multiple items by passing an ArrayList o
```

- 创建日历事件：

```
Intent calendarIntent = new Intent(Intent.ACTION_INSERT, Events
Calendar beginTime = Calendar.getInstance().set(2012, 0, 19, 7,
Calendar endTime = Calendar.getInstance().set(2012, 0, 19, 10,
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME,
calendarIntent.putExtra(Events.TITLE, "Ninja class");
calendarIntent.putExtra(Events.EVENT_LOCATION, "Secret dojo");
```

> 注：只有 API 级别为14 或更高级别支持此日历事件 Intent。

> 注： 可以更具体地自定义 Intent 是比较重要的。例如，如果你想要使用ACTION_VIEW intent 展示一张图片，应该指定 MIME 类型为 image/*。这可防止可“查看”数据的其他类型的应用（比如地图应用）被 Intent 触发。

验证是否存在接收 Intent 的应用

尽管 Android 平台保证某些 Intent 可以分解为内置应用之一（比如，“电话”、“电子邮件”或“日历”应用），但在调用 Intent 之前始终应该先验证是否有 App 接受这个 intent 的步骤。

> 注：如果您调用了 Intent，但设备上没有可用于接收处理 Intent 的应用，您的应用将崩溃。

为了验证是否有合适的 Activity 会响应这个intent，需要调用 queryIntentActivities()) 来获取到能够接收这个 Intent 的所有 Activity 列表。若返回的 List 非空，那么我们才可以安全的使用这个intent。例如：

```
PackageManager packageManager = getPackageManager();
List activities = packageManager.queryIntentActivities(intent,
        PackageManager.MATCH_DEFAULT_ONLY);
boolean isIntentSafe = activities.size() > 0;
```

如果 isIntentSafe 是 true，则至少有一个应用将响应该 Intent。 如果它是 false，则没有任何应用处理该 Intent。

> 注：我们必须在第一次使用之前做这个检查，若是不可行，则应该关闭这个功能。如果知道某个确切的应用能够处理这个 Intent，我们也可以向用户提供下载该应用的链接。(请参阅如何在 Google Play 链接产品).

启动具有 Intent 的 Activity

一旦您已创建您的 Intent 并设置 extra 信息，调用 startActivity() 将其发送给系统。如果系统识别可处理 Intent 的多个 Activity，它会为用户显示对话框供其选择要使用的应用，如图 1所示。

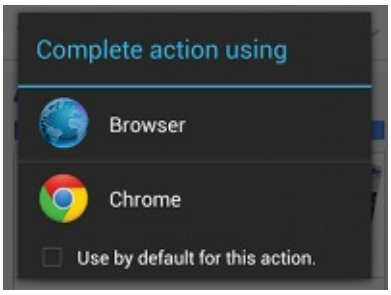


图1

如果只有一个 Activity 处理 Intent，系统会立即将其启动。

```
startActivity(intent);
```

此处显示完整的示例：如何创建查看地图的 Intent，验证是否存在处理 Intent 的应用，然后启动它：

```
// Build the intent
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mour
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);

// Verify it resolves
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities
boolean isIntentSafe = activities.size() > 0;

// Start an activity if it's safe
if (isIntentSafe) {
    startActivity(mapIntent);
}
```

显示应用选择器

注意，当您通过将您的 Intent 传递至 startActivity() 而启动 Activity 时，有多个应用响应 Intent，用户可以选择默认使用哪个应用（通过选中对话框底部的复选框；见图 1）。当执行用户通常希望每次使用相同应用进行的操作时，比如当打开网页（用户可能只使用一个网络浏览器）或拍照（用户可能习惯使用一个相机）时，这非常有用。

但是，如果要执行的操作可由多个应用处理并且用户可能 习惯于每次选择不同的应用 — 比如“共享”操作， 用户有多个应用分享项目 — 您应明确显示选择器对话框，如图 2 所示。选择器对话框 强制用户选择用于每次操作的 应用（用户不能对此操作选择默认的应用）。

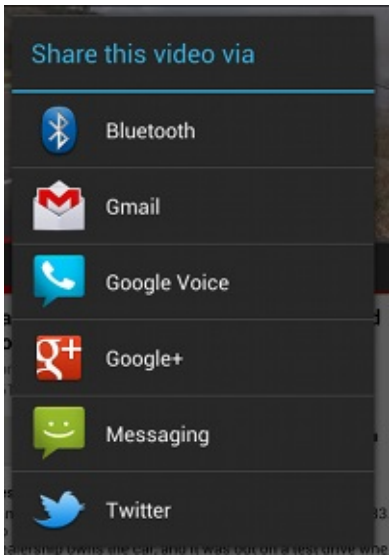


图**2**

要显示选择器，请使用 createChooser()) 创建Intent 并将其传递给 startActivity()。例如：

```
Intent intent = new Intent(Intent.ACTION_SEND);
...

// Always use string resources for UI text.
// This says something like "Share this photo with"
String title = getResources().getString(R.string.chooser_title);
// Create intent to show chooser
Intent chooser = Intent.createChooser(intent, title);

// Verify the intent will resolve to at least one activity
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

这将显示一个对话框，其中包含响应传递给 createChooser() 方法的 Intent 的应用列表，并且将提供的文本用作对话框标题。

> 翻译：@misparking
> 原始文档：https://developer.android.google.cn/training/basics/intents/sending.html#StartActivity

从 Activity 获取结果

启动另外一个 Activity 的方式，并不一定是单向的。你也可以启动另外一个 Activity，并且接收一个返回结果。可以通过调用 startActivityForResult()（而不是调用startActivity()）。

例如，你的应用可以启动一个相机应用，并且接收一个诸如拍摄的照片的结果。你也可以启动一个通讯录应用来让用户选择一个联系人，然后返回一个联系人细节信息的结果。

当然，能够响应的 Activity 必须被设计成能够返回结果的。这样，它会将结果以 Intent 对象来返回。你的 Activity 在 onActivityResult() 回调方法中接收结果。

> 备注：你可以在调用 startActivityForResult()，可以使用显式或隐式 Intent。当启动一个你自己的 Activity 来接收一个结果的时候，你可以使用显式 Intent 来确保你可以接收到期望的结果。

启动 Activity

当你启动一个 Activity 来获取一个结果的时候，对于 Intent 对象来说，没有什么特殊的，但是你需要传一个额外的整型参数给 startActivityForResult() 方法。

这个整型参数是用来标记你请求的“请求码”。当你接收到一个 Intent 结果的时候，回调会提供相同的请求码，以便你的应用能够正确的区分结果和决定如何处理结果。

例如，下面是一个启动一个让用户选择一个联系人的 Activity 的例子：

```
static final int PICK_CONTACT_REQUEST = 1;  // The request code
...
private void pickContact() {
    Intent pickContactIntent = new Intent(Intent.ACTION_PICK, Uri.p
    pickContactIntent.setType(Phone.CONTENT_TYPE); // Show user onl
    startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST)
}
```

接收结果

当用户在随后的 Activity 中完成操作并且返回之后，系统会调用你的 Activity 的 onActivityResult()) 方法。这个方法包含三个参数：

- 你之前传给 startActivityForResult()) 方法的请求码。
- 通过第二个 Activity 指定的结果码。结果码要么是 RESULT_OK，表示操作成功。或者是 RESULT_CANCELED，表示用户回退或者由于某些原因操作失败了。
- 携带结果数据的Intent 。

例如，下面是一个你可以处理“选择联系人” Intent 的结果：

```
@Override
protected void onActivityResult(int requestCode, int resultCode, In
    // Check which request we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // The user picked a contact.
            // The Intent's data Uri identifies which contact was s

            // Do something with the contact here (bigger example b
        }
    }
}
```

在这个例子中，由 Android 联系人应用返回的包含结果的 Intent 提供了可以标记用户选择的联系人的内容 Uri。

为了能够成功处理结果，你必须知道 Intent 结果的格式是什么。做到这一点，比较简单，只要 Acitivty 返回的结果是你自己 Activity 的。Android 平台的应用提供了他们自己的 API 能够让你处理指定的结果数据。例如，联系人应用总会返回一个带有能够识别所选联系人的内容 URI 的结果，相机应用会返回包含一个 Bitmap 对象的数据（参考类拍摄照片）。

追加：读取联系人数据

上面的代码展示了如何从联系人应用中返回的结果中读取数据，而不必深入细节，因为这个要求更多关于 Content Provider 的进一步的讨论。然而，如果你好奇的话，这里有进一步的代码教你如何从结果数据中获取到所选联系人的手机号码：

```
@Override
protected void onActivityResult(int requestCode, int resultCode, In
    // Check which request it is that we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // Get the URI that points to the selected contact
            Uri contactUri = data.getData();
            // We only need the NUMBER column, because there will b
            String[] projection = {Phone.NUMBER};

            // Perform the query on the contact to get the NUMBER c
            // We don't need a selection or sort order (there's onl
            // CAUTION: The query() method should be called from a
            // your app's UI thread. (For simplicity of the sample,
            // Consider using CursorLoader to perform the query.
            Cursor cursor = getContentResolver()
                    .query(contactUri, projection, null, null, null
            cursor.moveToFirst();

            // Retrieve the phone number from the NUMBER column
            int column = cursor.getColumnIndex(Phone.NUMBER);
            String number = cursor.getString(column);

            // Do something with the phone number...
        }
    }
}
```

> 备注：在 Android 2.3 （API level 9）之前，通过 Contacts Provider 执行查询时(如上面代码所示)需要你的应用声明 READ_CONTACTS 的权限（参见安全和权限）。然而从 Android 2.3 开始，联系人应用允许你的应用有一个读取联系人结果数据的临时权限。这个临时的权限仅适用于所请求的特定联系人，所以当你没有声明 READ_CONTACTS 权限的时候，就只能通过 Intent 中的 Uri 来查询特定联系人的数据。

允许其他应用启动您的 Activity

前两课重点讲述一方面：从您的应用启动另一个应用的 Activity。 但如果您的应用可以执行对另一个应用可能有用的操作，您的应用应准备好响应来自其他应用的操作请求。 例如，如果您构建一款可与用户的好友分享消息或照片的社交应用，您最关注的是支持 ACTION_SEND Intent 以便用户可以从另一应用发起“共享”操作并且启动您的应用执行该操作。

要允许其他应用启动您的 Activity，您需要在清单文件中为对应的 元素添加一个 元素。

当您的应用安装在设备上时，系统会识别您的 Intent 过滤器并添加信息至所有已安装应用支持的 Intent 内部目录。当应用通过隐含 Intent 调用 startActivity()) 或 startActivityForResult()) 时，系统会找到可以响应该 Intent 的 Activity。

添加 Intent 过滤器

为了正确定义您的 Activity 可处理的 Intent，您添加的每个 Intent 过滤器在操作类型和 Activity 接受的数据方面应尽可能具体。

如果 Activity 具有满足以下 Intent 对象条件的 Intent 过滤器，系统可能向 Activity 发送给定的 Intent：

操作

对要执行的操作命名的字符串。通常是平台定义的值之一，比如 ACTION_SEND 或 ACTION_VIEW。 使用 元素在您的 Intent 过滤器中指定此值。您在此元素中指定的值必须是操作的完整字符串名称，而不是 API 常量（请参阅以下示例）。

数据

与 Intent 关联的数据描述。 用 元素在您的 Intent 过滤器中指定此内容。使用此元素中的一个或多个属性，您可以只指定 MIME 类型、URI 前缀、URI 架构或这些的组合以及其他指示所接受数据类型的项。

> 注：如果您无需声明关于数据的具体信息 Uri（比如，您的 Activity 处理其他类型的“额外”数据而不是 URI 时），您应只指定 android:mimeType 属性声明您的 Activity 处理的数据类型，比如 `text/plain` 或 `image/jpeg` 。

类别

提供另外一种表征处理 Intent 的 Activity 的方法，通常与用户手势或 Activity 启动的位置有关。 系统支持多种不同的类别，但大多数都很少使用。 但是，所有隐含 Intent 默认使用 CATEGORY_DEFAULT 进行定义。 用 元素在您的 Intent 过滤器中指定此内容。

在您的 Intent 过滤器中，您可以通过声明嵌套在 元素中的具有相应 XML 元素的各项，来声明您的 Activity 接受的条件。

例如，此处有一个 Activity 与在数据类型为文本或图像时处理 ACTION_SEND Intent 的 Intent 过滤器：

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

每个入站 Intent 仅指定一项操作和一个数据类型，但可以在每个 中声明 、 和 元素的多个实例。

如果任何两对操作和数据的行为相斥，您应创建单独的 Intent 过滤器指定与哪种数据类型配对时哪些操作可接受。

比如，假定您的 Activity 同时处理 ACTION_SEND 和 ACTION_SENDTO Intent 的文本和图像。在这种情况下，您必须为两个操作定义两种不同的 Intent 过滤器，因为 ACTION_SENDTO Intent 必须使用数据 Uri 指定使用 send 或 sendto URI 架构的收件人地址。例如：

```
<activity android:name="ShareActivity">
    <!-- filter for sending text; accepts SENDTO action with sms UR
    <intent-filter>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:scheme="sms" />
        <data android:scheme="smsto" />
    </intent-filter>
    <!-- filter for sending text or images; accepts SEND action and
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

> 注： 为了接收隐含 Intent，您必须在 Intent 过滤器中包含 CATEGORY_DEFAULT 类别。方法 startActivity() 和startActivityForResult() 将按照已声明 CATEGORY_DEFAULT 类别的方式处理所有 Intent。如果您不在 Intent 过滤器中声明它，则没有隐含 Intent 分解为您的 Activity。

如需了解有关发送和接收 ACTION_SEND 执行社交共享行为的 Intent 的详细信息，请参阅有关从其他应用接收简单数据的课程。

处理您的 Activity 中的 Intent

为了决定在您的 Activity 执行哪种操作，您可读取用于启动 Activity 的 Intent。

当您的 Activity 启动时，调用 getIntent() 检索启动 Activity 的 Intent。您可以在 Activity 生命周期的任何时间执行此操作，但您通常应在早期回调时（比如，onCreate() 或 onStart()）执行。

例如：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    // Get the intent that started this activity
    Intent intent = getIntent();
    Uri data = intent.getData();

    // Figure out what to do based on the intent type
    if (intent.getType().indexOf("image/") != -1) {
        // Handle intents with image data ...
    } else if (intent.getType().equals("text/plain")) {
        // Handle intents with text ...
    }
}
```

返回结果

如果您想要向调用您的 Activity 的 Activity 返回结果，只需调用 setResult()) 指定结果代码和结果 Intent。当您的操作完成且用户应返回原始 Activity 时，调用 finish() 关闭（和销毁）您的 Activity。 例如：

```
// Create intent to deliver some kind of result data
Intent result = new Intent("com.example.RESULT_ACTION", Uri.parse('
setResult(Activity.RESULT_OK, result);
finish();
```

您必须始终为结果指定结果代码。通常，它是 RESULT_OK 或 RESULT_CANCELED。您之后可以根据需要为 Intent 提供额外的数据。

> 注： 默认情况下，结果设置为 RESULT_CANCELED。因此，如果用户在完成操作动作或设置结果之前按了返回按钮，原始 Activity 会收到“已取消”的结果。

如果您只需返回指示若干结果选项之一的整数，您可以将结果代码设置为大于 0 的任何值。如果您使用结果代码传递整数，且无需包括 Intent，则可调用 setResult() 且仅传递结果代码。例如：

setResult(RESULT_COLOR_RED); finish(); 在这种情况下，只有几个可能的结果，因此结果代码是一个本地定义的整数（大于 0）。 当您向自己应用中的 Activity 返回结果时，这将非常有效，因为接收结果的 Activity 可引用公共常数来确定结果代码的值。

> 注： 无需检查您的 Activity 是使用 startActivity() 还是 startActivityForResult() 启动的。如果启动您 Activity 的 Intent 可能需要结果，只需调用 setResult()。如果原始 Activity 已调用 startActivityForResult()，则系统将向其传递您提供给 setResult() 的结果；否则，会忽略结果。
>
> 翻译：@edtj
> 原始文档：https://developer.android.google.cn/training/basics/firstapp/creating-project.html

声明权限

所有的 Android 应用都运行在一个访问受限制的沙盒中。如果你的需要使用自己沙盒外的资源或信息，那么应用需要请求对应的权限。声明所需的权限在 AppManifest 文件的权限列表中。

根据权限的敏感程度，系统可能会自动授予权限，或者需要请求用户来授予权限。例如：你的应用需要打开设备闪光灯的权限，这个权限会被系统自动授予。但是如果你需要读取用户联系人，系统会询问用户是否授予该权限。根据不同系统版本，Android5.0 及以下用户授予权限在安装时，Android6.0 及以上在应用运行时由用户授予权限。

确定你的应用需要的权限

在开发应用中，你应该注意应用所需要请求的权限。通常，应用需要请求权限在使用非自身创建的信息或资源时，或在执行影响设备或其它应用行为的操作时。例如，如果应用需要访问互联网，使用设备相机，或打开和关闭 Wifi，都需要请求适当的权限。系统权限列表，参阅《正常与危险权限》。

应用只需要为直接执行的动作授予权限。当请求因外一个应用执行任务或提供信息时不需要权限。例如，如果应用需要去读取用户通讯录，则需要 READ_CONTACTS 权限。但如果通过 Intent 去请求通讯录应用的信息，你的应用不需要任何权限，但是通讯录应用需要该权限。更多信息，参阅《考虑使用 Intent》。

添加权限到清单文件

声明应用所需的权限，放 标签到应用清单中，作为顶级 <manifest> 标签的子项。例如，应用需要发送短信应该放入如下代码在清单文件中：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/androi
        package="com.example.snazzyapp">

    <uses-permission android:name="android.permission.SEND_SMS"/>


    <application ...>
        ...
    </application>

</manifest>
```

你声明的权限有多敏感决定了之后的系统行为。如果你的权限不涉及到用户隐私，系统会自动授予。如果权限访问到用户的敏感信息，系统会提示用户是否授予该权限。更多关于不同权限种类的信息，参阅《正常与危险权限》。

> 翻译：@iOnesmile
> 原始文档：https://developer.android.google.cn/training/permissions/declaring.html

在运行时请求权限

从 Android 6.0（API 级别 23）开始，用户开始在应用运行时向其授予权限，而不是在应用安装时授予。此方法可以简化应用安装过程，因为用户在安装或更新应用时不需要授予权限。它还让用户可以对应用的功能进行更多控制；例如，用户可以选择为相机应用提供相机访问权限，而不提供设备位置的访问权限。用户可以随时进入应用的“Settings”屏幕调用权限。

系统权限分为两类：正常权限和危险权限：

- 正常权限不会直接给用户隐私权带来风险。如果你的应用在其清单中列出了正常权限，系统将自动授予该权限。
- 危险权限会授予应用访问用户机密数据的权限。如果你的应用在其清单中列出了正常权限，系统将自动授予该权限。如果你列出了危险权限，则用户必须明确批准你的应用使用这些权限。

如需了解详细信息，请参阅正常权限和危险权限。

在所有版本的 Android 中，你的应用都需要在其应用清单中同时声明它需要的正常权限和危险权限，如声明权限中所述。不过，该声明的影响因系统版本和应用的目标 SDK 级别的不同而有所差异：

- 如果设备运行的是 Android 5.1 或更低版本，或者应用的目标 SDK 为 22 或更低：如果你在清单中列出了危险权限，则用户必须在安装应用时授予此权限；如果他们不授予此权限，系统根本不会安装应用。
- 如果设备运行的是 Android 6.0 或更高版本，或者应用的目标 SDK 为 23 或更高：应用必须在清单中列出权限，并且它必须在运行时请求其需要的每项危险权限。用户可以授予或拒绝每项权限，且即使用户拒绝权限请求，应用仍可以继续运行有限的功能。

> 注：从 Android 6.0（API 级别 23）开始，用户可以随时从任意应用调用权限，即使应用面向较低的 API 级别也可以调用。无论你的应用面向哪个 API 级别，你都应对应用进行测试，以验证它在缺少需要的权限时行为是否正常。

检查权限

如果你的应用需要危险权限，则每次执行需要这一权限的操作时你都必须检查自己是否具有该权限。用户始终可以自由调用此权限，因此，即使应用昨天使用了相机，它不能假设自己今天仍具有该权限。

要检查你是否具有某项权限，请调用 ContextCompat.checkSelfPermission() 方法。例如，以下代码段显示了如何检查 Activity 是否具有在日历中进行写入的权限：

```
// Assume thisActivity is the current activity
int permissionCheck = ContextCompat.checkSelfPermission(thisActivit
        Manifest.permission.WRITE_CALENDAR);
```

如果应用具有此权限，方法将返回 PackageManager.PERMISSION_GRANTED，并且应用可以继续操作。如果应用不具有此权限，方法将返回 PERMISSION_DENIED，且应用必须明确向用户要求权限。

请求权限

如果你的应用需要应用清单中列出的危险权限，那么，它必须要求用户授予该权限。Android 为你提供了多种权限请求方式。调用这些方法将显示一个标准的 Android 对话框，不过，你不能对它们进行自定义。

解释应用为什么需要权限

在某些情况下，你可能需要帮助用户了解你的应用为什么需要某项权限。例如，如果用户启动一个摄影应用，用户对应用要求使用相机的权限可能不会感到吃惊，但用户可能无法理解为什么此应用想要访问用户的位置或联系人。在请求权限之前，不妨为用户提供一个解释。请记住，你不需要通过解释来说服用户；如果你提供太多解释，用户可能发现应用令人失望并将其移除。

你可以采用的一个方法是仅在用户已拒绝某项权限请求时提供解释。如果用户继续尝试使用需要某项权限的功能，但继续拒绝权限请求，则可能表明用户不理解应用为什么需要此权限才能提供相关功能。对于这种情况，比较好的做法是显示解释。

为了帮助查找用户可能需要解释的情形，Android 提供了一个实用程序方法，即 shouldShowRequestPermissionRationale()。如果应用之前请求过此权限但用户拒绝了请求，此方法将返回 true。

> 注：如果用户在过去拒绝了权限请求，并在权限请求系统对话框中选择了 Don't ask again 选项，此方法将返回 false。如果设备规范禁止应用具有该权限，此方法也会返回 false。

请求你需要的权限

如果应用尚无所需的权限，则应用必须调用一个 requestPermissions() 方法，以请求适当的权限。应用将传递其所需的权限，以及你指定用于识别此权限请求的整型请求代码。此方法异步运行：它会立即返回，并且在用户响应对话框之后，系统会使用结果调用应用的回调方法，将应用传递的相同请求代码传递到 requestPermissions()。

以下代码可以检查应用是否具备读取用户联系人的权限，并根据需要请求该权限：

```
// Here, thisActivity is the current activity
if (ContextCompat.checkSelfPermission(thisActivity,
                Manifest.permission.READ_CONTACTS)
        != PackageManager.PERMISSION_GRANTED) {

    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisAct
            Manifest.permission.READ_CONTACTS)) {

        // Show an expanation to the user *asynchronously* -- don't
        // this thread waiting for the user's response! After the u
        // sees the explanation, try again to request the permissio

    } else {

        // No explanation needed, we can request the permission.

        ActivityCompat.requestPermissions(thisActivity,
                new String[]{Manifest.permission.READ_CONTACTS},
                MY_PERMISSIONS_REQUEST_READ_CONTACTS);

        // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
        // app-defined int constant. The callback method gets the
        // result of the request.
    }
}
```

> 注：当你的应用调用 requestPermissions() 时，系统将向用户显示一个标准对话框。你的应用无法配置或更改此对话框。如果你需要为用户提供任何信息或解释，你应在调用 requestPermissions() 之前进行，如解释应用为什么需要权限中所述。

处理权限请求响应

当应用请求权限时，系统将向用户显示一个对话框。当用户响应时，系统将调用应用的 onRequestPermissionsResult() 方法，向其传递用户响应。你的应用必须替换该方法，以了解是否已获得相应权限。回调会将你传递的相同请求代码传递给

requestPermissions()。例如，如果应用请求 READ_CONTACTS 访问权限，则它可能采用以下回调方法：

```
@Override
public void onRequestPermissionsResult(int requestCode,
        String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
            // If request is cancelled, the result arrays are empty
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRA

                // permission was granted, yay! Do the
                // contacts-related task you need to do.

            } else {

                // permission denied, boo! Disable the
                // functionality that depends on this permission.
            }
            return;
        }

        // other 'case' lines to check for other
        // permissions this app might request
    }
}
```

系统显示的对话框说明了你的应用需要访问的权限组；它不会列出具体权限。例如，如果你请求 READ_CONTACTS 权限，系统对话框只显示你的应用需要访问设备的联系人。用户只需要为每个权限组授予一次权限。如果你的应用请求该组中的任何其他权限（已在你的应用清单中列出），系统将自动授予应用这些权限。当你请求此权限时，系统会调用你的 onRequestPermissionsResult() 回调方法，并传递 PERMISSION_GRANTED，如果用户已通过系统对话框明确同意你的权限请求，系统将采用相同方式操作。

> 注：你的应用仍需要明确请求其需要的每项权限，即使用户已向应用授予该权限组中的其他权限。此外，权限分组在将来的 Android 版本中可能会发生变化。你的代码不应依赖特定权限属于或不属于相同组这种假设。

例如，假设你在应用清单中列出了 READ_CONTACTS 和 WRITE_CONTACTS。如果你请求 READ_CONTACTS 且用户授予了此权限，那么，当你请求 WRITE_CONTACTS 时，系统将立即授予你该权限，不会与用户交互。

如果用户拒绝了某项权限请求，你的应用应采取适当的操作。例如，你的应用可能显示一个对话框，解释它为什么无法执行用户已经请求但需要该权限的操作。

当系统要求用户授予权限时，用户可以选择指示系统不再要求提供该权限。这种情况下，无论应用在什么时候使用 requestPermissions() 再次要求该权限，系统都会立即拒绝此请求。系统会调用你的 onRequestPermissionsResult() 回调方法，并传递 PERMISSION_DENIED，如果用户再次明确拒绝了你的请求，系统将采用相同方式操作。这意味着当你调用 requestPermissions() 时，你不能假设已经发生与用户的任何直接交互。

> 备注:
> 翻译 ：@jarylan
> 原始文档:https://developer.android.com/training/permissions/requesting.html

权限使用说明

应用程序很容易通过权限请求来影响用户。如果用户发现应用程序使用起来令人沮丧，或者用户担心应用程序可能正在处理用户的信息，他们可能会避免使用应用程序或完全卸载它。以下最佳做法可以帮助您避免这种糟糕的用户体验。

考虑使用意图

在许多情况下，您可以在应用程序执行任务的两种方式之间进行选择。您可以让您的应用程序请求执行操作本身的权限。或者，您可以让应用程序使用意图让另一个应用程序执行任务。

例如，假设您的应用程序需要能够使用设备相机拍摄照片。您的应用程式可以要求相机权限，让您的应用程式直接存取相机。然后，您的应用程序将使用相机APIs控制相机并拍摄照片。这种方法使您的应用程序完全控制摄影过程，并允许您将相机用户界面合并到您的应用程序。

但是，如果您不需要此类完全控制，则可以使用ACTION_IMAGE_CAPTURE意图请求图像。当您发送意图时，系统提示用户选择相机应用程序（如果没有默认的相机应用程序）。用户使用所选的相机应用程序拍摄照片，该应用程序将该图片返回到应用程序的onActivityResult（）方法。

同样，如果您需要拨打电话，访问用户的联系人等，您可以通过创建相应的意图来完成此操作，也可以直接请求权限并访问相应的对象。每种方法都有优点和缺点。

如果您使用权限：

- 当您执行操作时，您的应用程序可完全控制用户体验。然而，这种广泛的控制增加了你的任务的复杂性，因为你需要设计一个合适的UI。
- 系统会提示用户在运行时或安装时授予一次权限（具体取决于用户的Android版本）。之后，您的应用程序可以执行操作，而不需要用户的额外交互。但是，如果用户未授予权限（或稍后撤消该权限），您的应用程序将无法完成任何操作。

如果你使用意图：

- 您不必为操作设计UI。处理意图的应用程序提供了UI。但是，这意味着您无法控制用户体验。用户可以与您从未见过的应用程序进行交互。
- 如果用户没有操作的默认应用程序，则系统提示用户选择应用程序。如果用户未指定默认处理程序，则他们可能必须在每次执行操作时都通过一个额外的对话框。

只需要您需要的权限

每次你请求一个权限，你强制用户做出决定。您应该尽量减少发出这些请求的次数。如果用户运行的是Android 6.0（API级别23）或更高版本，则每次用户尝试一些需要权限的新应用程序功能时，应用程序必须通过权限请求中断用户的工作。如果用户运行的是较早版本的Android，则用户必须在安装应用程序时授予每个应用程序的权限;如果列表太长或似乎不合适，用户可能决定不安装您的应用程序。出于这些原因，您应该最小化您的应用需要的权限的数量。

通常，您的应用程序可以避免通过使用意图请求权限。如果某个功能不是应用程序功能的核心部分，您应该考虑将工作交给另一个应用程序，如考虑使用意图中所述。

不要影响用户

如果用户运行的是Android 6.0（API级别23）或更高版本，则用户必须在运行应用程序时授予其权限。如果您同时面对许多权限请求，您可能会对用户造成压力，导致他们退出您的应用。相反，您应该在需要时请求权限。

在某些情况下，一个或多个权限可能对您的应用程序是绝对必要的。应用程式启动时请求所有权限可能有意义。例如，如果您制作了一个摄影应用程序，该应用程序将需要访问设备的相机。当用户第一次启动应用程序时，他们不会惊讶地被要求使用相机的权限。但如果同一个应用程序也有一个功能来与用户的联系人共享照片，你可能不应该在第一次启动时请求READ_CONTACTS权限。相反，请等到用户尝试使用“共享”功能，然后请求权限。

如果您的应用提供教程，在教程序列结束时请求应用程序的基本权限可能是有意义的。

解释为什么你需要权限

当您调用requestPermissions（）时系统显示的权限对话框说明您的应用程序需要什么权限，但不说为什么。在一些情况下，用户可能发现困惑。在调用requestPermissions（）之前，向用户解释为什么您的应用程序需要权限是一个好主意。

例如，摄影应用可能需要使用位置服务，因此可以对照片进行地理标记。典型的用户可能不知道照片可以包含位置信息，并且会困惑他们的摄影应用想要知道位置的原因。所以在这种情况下，应用程序在调用requestPermissions（）之前告诉用户此功能是一个好主意。

通知用户的一种方法是将这些请求合并到应用教程中。教程可以依次显示每个应用程序的功能，因为它可以解释需要什么权限。例如，摄影应用程序的教程可以演示其“与您的联系人共享照片”功能，然后告诉用户他们需要授予应用程序的权限才能查看用户的联系人。然后，应用程序可以调用requestPermissions（）来请求用户访问该访问。当然，并不是每个用户都会遵循教程，因此您仍然需要在应用程序正常操作期间检查并请求权限。

测试两个权限模型

从Android 6.0（API级别23）开始，用户在运行时授予和撤销应用权限，而不是在安装应用时执行此操作。因此，您必须在更广泛的条件下测试您的应用程式。在Android 6.0之前，你可以合理地假设，如果你的应用程序是在运行，它具有它在应用程序清单中声明的所有权限。在新的权限模式下，您不能再进行此假设。

以下提示将帮助您确定运行API级别23或更高版本的设备的与权限相关的代码问题：

- 确定您应用的当前权限和相关代码路径。
- 测试跨受权限保护的服务和数据的用户流。
- 使用授予或撤销权限的各种组合进行测试。例如，相机应用程序可能会在其清单中列出CAMERA，READ_CONTACTS和ACCESS_FINE_LOCATION。您应该测试应用程序的每个权限打开和关闭，以确保应用程序可以正常处理所有权限配置。请注意，从Android 6.0开始，用户可以为任何应用打开或关闭权限，即使是目标为API级别22或更低的应用。
- 使用adb工具从命令行管理权限：

按组列出权限和状态：

```
$ adb shell pm list permissions -d -g
```

授予或撤销一个或多个权限：

```
$ adb shell pm [grant | revoke] <permission-name> ...
```

- 分析您的应用程序使用权限的服务。

分享简单的数据

应用间可以互相通信是 Android 程序中最棒的功能之一。当一个功能已存在于其他应用中，且并不是本程序的核心功能时，完全没有必要重新对其进行编写。

本章节会讲述一些在不同应用之间通过使用 Intent APIs与 ActionProvider 对象来发送与接受简单数据的常用方法。

课程

向其他应用发送简单的数据

学习如何使用 intent 向其他应用发送文本和二进制数据。

接收从其他应用返回的数据

学习如何通过 Intent 在我们的应用中接收来自其他应用的文本和二进制数据。

增加简单的分享功能

学习如何在 Acitonbar 上添加一个分享功能。

向其他应用发送简单的数据

当构建一个 Intent 时，必须指定这个 Intent 需要触发的 Action。Android 定义了一些 Action，比如ACTION_SEND，该 Action 会指示 Intent 用于从一个 Activity 发送数据到另外一个 Activity，甚至可以是跨进程之间的数据发送。为了发送数据到另外一个 Activity，我们只需要说明数据及其类型，系统会自动识别出能够兼容接受这些数据的 Activity。如果这些选择有多个，则把这些 Activity 显示给用户进行选择；如果只有一个，则立即启动该 Activity。同样的，我们可以在 manifest 文件的 Activity 描述中添加接受的数据类型。

在不同的程序之间使用 Intent 收发数据是在社交分享内容时最常用的方法。Intent 使用户能够通过最喜欢的程序进行快速简单的信息分享。

注意:为 ActionBar 添加分享功能的最佳方法是使用 ShareActionProvider，API level 14 以上的系统才支持。ShareActionProvider 将在关于添加简易分享 Action的课程中进行详细介绍。

分享文本内容

ACTION_SEND 最直接常用的地方是从一个 Activity 发送文本内容到另外一个 Activity。例如，Android 内置的浏览器可以将当前显示页面的 URL 作为文本内容分享到其他程序。这一功能对于通过邮件或者社交网络来分享文章或者网址给好友而言是非常有用的。下面是一段展示分享功能的代码:

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(sendIntent);
```

如果设备上安装有某个能够匹配 ACTION_SEND 且 MIME 类型为 text/plain 的程序，则 Android 系统会立即运行它。若有多个匹配的程序，则系统会把他们都给筛选出来，并呈现一个选择提示框给用户进行选择。

然而，如果你调用了 Intent.createChooser())，传入 Intent 对象，那么 Android 总是会显示选择器。这样做有一些好处：

- 即使用户之前为这个 Intent 设置了默认的 Action，选择界面还是会显示。
- 如果没有匹配的程序，Android会 显示系统信息。
- 我们可以指定选择器界面的标题。

下面是更新后的代码：

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(Intent.createChooser(sendIntent, getResources().getTe
```

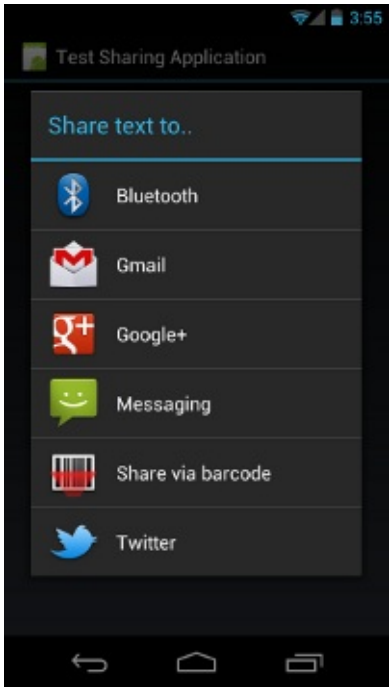


图**1**： 手机端 ACTION_SEND 意图选择器屏幕截图。

图 1 显示的弹出框结果。

另外，我们可以为 Intent 设置一些标准的附加值，例如：EXTRA_EMAIL, EXTRA_CC, EXTRA_BCC, EXTRA_SUBJECT 等。如果接收程序没有针对这些做特殊的处理，则会忽略不处理。

> 注： 一些 e-mail 程序，例如 Gmail,对应接收的是 EXTRA_EMAIL, EXTRA_CC，他们都是 String[] 类型的，可以使用 putExtra(string,string[])) 方法来添加至 Intent 中。

分享二进制内容

分享二进制的数据需要使用 ACTION_SEND 设置特定的 MIME 类型，需要在 EXTRA_STREAM 里面放置数据的 URI。 下面有个分享图片的例子，该例子也可以修改用于分享任何类型的二进制数据：

```
Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);
shareIntent.putExtra(Intent.EXTRA_STREAM, uriToImage);
shareIntent.setType("image/jpeg");
startActivity(Intent.createChooser(shareIntent, getResources().get
```

请注意以下内容：

- 我们可以使用 */* 这样的方式来指定 MIME 类型，但是这仅仅会匹配到那些能够处理一般数据类型的 Activity (即一般的 Activity 无法详尽所有的 MIME 类型)。
- 接收的程序需要有访问 Uri 资源的权限。推荐以下方法来处理这个问题：
    - 将数据存储在 ContentProvider 中，确保其他程序有访问 ContentProvider 的权限。较好的提供访问权限的方法是使用 per-URI permissions，其对接收程序而言是只是暂时拥有该许可权限。类似于这样创建 ContentProvider 的一种简单的方法是使用 FileProvider 帮助类。
    - 使用 MediaStore 系统。MediaStore 系统主要用于音视频及图片的 MIME 类型。但在 Android 3.0 之后，它也可以用于存储非多媒体类型的数据（更多信息请参阅 MediaStore.Files）。

Files can be inserted into the MediaStore using scanFile() after which a content:// style Uri suitable for sharing is passed to the provided onScanCompleted() callback.

在有适合分享的 `content://` 类型的 Uri 传递到 onScanCompleted()) 回调方法之后，可以通过调用 scanFile()) 方法来向 MediaStore 插入文件。

发送多个内容

为了同时分享多种不同类型的内容，需要 ACTION_SEND_MULTIPLE Action 与指定内容 URI 列表一起配合使用。MIME 类型也会根据分享的内容结构的不同而有差异。例如，如果你分享 3 张 JPEG 格式的图片，那么 MIME 类型仍然是 "image/jpeg"。如果是不同图片格式的话，应该是用 "image/*" 来匹配那些可以接收任何图片类型的 Activity。如果需要分享多种不同类型的数据，可以使用 */* 来表示 MIME。像前面描述的那样，这取决于接收你数据的应用如何解析和处理你的数据。下面是一个例子：

```
ArrayList<Uri> imageUris = new ArrayList<Uri>();
imageUris.add(imageUri1); // Add your image URIs here
imageUris.add(imageUri2);
I
Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND_MULTIPLE);
shareIntent.putParcelableArrayListExtra(Intent.EXTRA_STREAM, imageU
shareIntent.setType("image/*");
startActivity(Intent.createChooser(shareIntent, "Share images to..'
```

当然，请确保指定到数据的 URI 能够被接收程序所访问。

> 翻译：@misparking
> 原始文档：https://developer.android.google.cn/training/sharing/send.html

从其他应用接收简单数据

如同你的应用可以向其他应用发送数据一样，你的应用也可以很容易的从其他应用接收数据。思考下用户如何和你的应用进行交互，以及你想要从其他应用获取数据的类型。例如，社交网络类的应用更倾向于接收文本内容，从另外一个应用获取诸如网页 URL 的数据。Google+ Android 应用既接收文本信息，又可以接收单张或多张图片。通过这个应用，用户可以很容易的在 Google+ 中发布从 Android 图片应用中获取的图片。

更新你的 **Manifest** 文件

意图过滤器提醒系统一个应用组件想要接收什么意图。和在向其他应用发送简单数据课程里，你如何通过 ACTION_SEND 的 Action 来构建一个意图类似，你创建意图过滤器是为了通过这个 Action 来接收意图。你在 Manifest 文件中定义一个意图过滤器，使用 `<intent-filter>` 元素。例如，如果你的应用处理文本内容接收，一张或者多张任何类型的图片，你的 Manifest 文件就会如下所示：

```
<activity android:name=".ui.MyActivity" >
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND_MULTIPLE"
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
</activity>
```

> 备注： 更多关于意图过滤器和意图解决方案，请参阅意图和意图过滤器。

当其他应用通过构建一个意图并将它传给 startActivity()) 方法来尝试分享此种内容的时候，你的应用将会在意图选择器中作为一个可选项被列出。相符合 Activity（以上样例中的 `.ui.MyActivity`） 将会被启动。然后就是取决于你在你的代码和 UI 中如何正确的处理这个内容了。

处理接收的内容

要处理 Intent 传送的内容，需要先通过调用 getIntent()) 方法来获取 Intent 对象。一旦你拥有了这个对象，你就可以判断它的内容来决定下一步做什么。要注意的就是如果这个 Activity 可以被系统的其他部分启动的话，例如一个 launcher，需要在检查 intent 的时候考虑这种情况。 Keep in mind that if this activity can be started from other parts of the system, such as the launcher, then you will need to take this into consideration when examining the intent.

```
void onCreate (Bundle savedInstanceState) {
    ...
    // Get intent, action and MIME type
    Intent intent = getIntent();
    String action = intent.getAction();
    String type = intent.getType();

    if (Intent.ACTION_SEND.equals(action) && type != null) {
        if ("text/plain".equals(type)) {
            handleSendText(intent); // Handle text being sent
        } else if (type.startsWith("image/")) {
            handleSendImage(intent); // Handle single image being s
        }
    } else if (Intent.ACTION_SEND_MULTIPLE.equals(action) && type
        if (type.startsWith("image/")) {
            handleSendMultipleImages(intent); // Handle multiple im
        }
    } else {
        // Handle other intents, such as being started from the hom
    }
    ...
}
```

```
void handleSendText(Intent intent) {
    String sharedText = intent.getStringExtra(Intent.EXTRA_TEXT);
    if (sharedText != null) {
        // Update UI to reflect text being shared
    }
}

void handleSendImage(Intent intent) {
    Uri imageUri = (Uri) intent.getParcelableExtra(Intent.EXTRA_STR
    if (imageUri != null) {
        // Update UI to reflect image being shared
    }
}

void handleSendMultipleImages(Intent intent) {
    ArrayList<Uri> imageUris = intent.getParcelableArrayListExtra(
    if (imageUris != null) {
        // Update UI to reflect multiple images being shared
    }
}
```

> 注意：一定要格外检查接收的数据，你无法知道其他应用可能给你发送的数据类型。比如，可能会设置错误的 MIME 类型，或者过大的图片。因此我们应避免在 UI 线程里面去处理那些获取到的数据。

更新 UI 可以像填充 EditText 一样简单，也可以是更加复杂一点的操作，例如过滤出感兴趣的图片。这个完全取决于应用接下来要做些什么。

> 翻译：@ifeegoo
> 校对：@misparking
> 原始文档：https://developer.android.google.cn/training/sharing/receive.html

添加一个简单的分享动作

在 Android 4.0(API Level 14) 引入的 ActionProvider 是你更容易的在 AcionBar 实现一个有效并且友好的分享动作。ActionProvider 一但附加到 action bar 的菜单选项中，就会去处理该选项的外观和行为。在使用 ShareActionProvider 的情况下，你只需要提供意图他会做其他的。

> **Note:** ShareActionProvider 是在API Level 14 以及更高的版本提供的。

更新菜单声明

开始使用 ShareActionProvider，在你的 menu resource 文件中为相应的 `<item>` 定义 `android:actionProviderClass` 属性:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
            android:id="@+id/menu_item_share"
            android:showAsAction="ifRoom"
            android:title="Share"
            android:actionProviderClass=
                "android.widget.ShareActionProvider" />
    ...
</menu>
```

项目的外观和行为将委托给 ShareActionProvider。但是你需要告诉提供者你想要分享。

设置分享的意图

在 ShareActionProvider 的顺序方法中，你必须提供一个分享意图。这个分享意图的描述应该是相同的与 Sending Simple Data to Other Apps 课程中，有 ACTION_SEND 的动作和添加 EXTRA_TEXT 和 EXTRA_STREAM 附加数据。分配分享意图，首先发现相应的 MenuItem 当你的菜单资源文件填充了你的 Activity

或者 Fragment 。然后，调用 MenuItem.getActionProvider()) 来检索 ShareActionProvider 的实例。使用 setShareIntent()) 去更新与该操作关联的分享意图。这里是个例子：

```
private ShareActionProvider mShareActionProvider;
...

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate menu resource file.
    getMenuInflater().inflate(R.menu.share_menu, menu);

    // Locate MenuItem with ShareActionProvider
    MenuItem item = menu.findItem(R.id.menu_item_share);

    // Fetch and store ShareActionProvider
    mShareActionProvider = (ShareActionProvider) item.getActionProv

    // Return true to display menu
    return true;
}

// Call to update the share intent
private void setShareIntent(Intent shareIntent) {
    if (mShareActionProvider != null) {
        mShareActionProvider.setShareIntent(shareIntent);
    }
}
```

你可能需要在创建菜单只设置分享意图一次，或者你可能希望设置的时候然后去更新 UI 的改变。例如，当你在图库应用中全屏查看照片时， 当浏览的图片改变时分享意图也需要改变。

有关 ShareActionProvider 对象的进一步讨论，请参阅 Action Views and Action Providers。

> 翻译：@edtj
> 原始文档：https://developer.android.google.cn/training/basics/firstapp/creating-project.html

设置文件共享

当向其它应用安全的提供一个文件时，需要在应用中为提供的文件配置安全的方式，通过内容 URI。Android FileProvider 组件会为文件生成内容 URI，依赖于你在 XML 中提供的配置。本节将说明如何在应用实现 FileProvider ，以及如何指定你要提供给另外一个应用的文件。

> 注：FileProvider 是在 V4（v4 Support Library）包中，关于导入这个库到应用中，请参见 Support Library Setup。

指定 FileProvider

在清单文件中需要配置一个入口来定义 FileProvider。这个入口指向一个生成的内容 URIs，同样的这个 XML 文件名指向一个你分享的目录。

下面的代码片段表述了如何在清单文件中添加 <provider> 元素，来指向 FileProvider 对象，如下：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/androi
    package="com.example.myapp">
    <application
        ...>
        <provider
            android:name="android.support.v4.content.FileProvider"
            android:authorities="com.example.myapp.fileprovider"
            android:grantUriPermissions="true"
            android:exported="false">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/filepaths" />
        </provider>
        ...
    </application>
</manifest>
```

上例中，android:authorities 属性指向一个 URI，这个 URI 是 FileProvider 根据内容生成的。本例中，认证码是：*com.example.myapp.fileprovider*。为自己的应用指向的认证码是 android:package 加 "fileprovider" 的后缀。学习更多的认证值，见主题 Content URIs 和 android:authorities 属性的文档。

在 XML 文件中的 <provider> 节点下，<meta-data> 子元素是指向你想分享的目录。*android:resource* 的值是文件的路径和名称，排除了 *.xml* 的后缀。下一节将介绍描述文件的内容。

指定可分享的目录

一旦你添加 FileProvider 到清单文件中，就需要去指定包括你想分享的文件目录。创建 *filepaths.xml* 文件来指定目录，在项目的 *res/xml/* 子目录下。在这个文件中，为每一个目录添加一个 XML 标签。下面的片段是 *res/xml/filepaths.xml* 例子的内容。这个片段也展示了如何如分享子目录，在内存卡区域的 *file/* 目录中。

```
<paths>
    <files-path path="images/" name="myimages" />
</paths>
```

在本例中，*<files-path>* 标签分享的目录是在内存卡的 files/ 目录下。path 属性表示分享 images/ 的子目录来自 files/ 目录下。name 属性告诉 FileProvider 添加路径 files/images/ 到内容 URIs 中。

<paths> 元素可以拥有多个子元素，每个元素指向不同分享的目录。在添加的 *<files-path>* 元素中，使用 *<external-path>* 标签分享内存卡目录，使用 *<cache-path>* 标签分享内部缓存目录。学习更多分享目录的子标签，参见 FileProvider 相关文档。

> 注：XML 文件是分享文件仅有的方式，不能直接通过程序分享目录。

现在学习了如何使用 FileProvider 的完整流程，生成的内容 URIs 对应的是应用的内存卡或 files/ 的子目录下的文件。当为文件生成一个内容 URI 时，它包含 *<provider>* 元素 *(com.example.myapp.fileprovider)*，路径 *images/* 和 文件名。

例如，当按照本节去定义一个 FileProvider 时，你请求 *default_image.jpg* 文件的 URI，FileProvider 会返回如下路径：

```
content://com.example.myapp.fileprovider/myimages/default_image.jpg
```

翻译：@iOnesmile
原始文档：https://developer.android.google.cn/training/secure-file-sharing/setup-sharing.html#DefineProvider

共享文件

将应用设置为使用内容 URI 共享文件后，你可以响应其他应用对这些文件的请求。响应这些请求的一种方式是从服务器应用提供其他应用可以调用的文件选择接口。这种方法允许客户端应用程序让用户从服务器应用程序中选择一个文件，然后接收所选文件的内容 URI。

本课将向你展示如何在应用程序中创建响应文件请求的文件选择的 Activity

接收文件请求

要从客户端应用程序接收文件请求并使用内容 URI 进行响应，你的应用程序应提供文件选择 Activity。 客户端应用程序通过使用包含操作 ACTION_PICK 的 Intent 调用 startActivityForResult（）来启动此 Activity。 当客户端应用调用 startActivityForResult（） 时，你的应用会返回用户选择文件的 URI 地址。

要了解如何在客户端应用程序中实现文件请求，请参阅请求共享文件中的课程。

创建文件选择的 **Activity**

要创建文件选择的 Activity，请首先在清单中指定 Activity，以及与操作 ACTION_PICK 和类别 CATEGORY_DEFAULT 和 CATEGORY_OPENABLE 匹配的意图过滤器。 还要为你的应用为其他应用提供的文件添加 MIME 类型过滤器。以下代码段显示如何指定新的 Activity 和意向过滤器：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/androic
...
    <application>
    ...
        <activity
            android:name=".FileSelectActivity"
            android:label="@File Selector" >
            <intent-filter>
                <action
                    android:name="android.intent.action.PICK"/>
                <category
                    android:name="android.intent.category.DEFAULT"/
                <category
                    android:name="android.intent.category.OPENABLE"
                <data android:mimeType="text/plain"/>
                <data android:mimeType="image/*"/>
            </intent-filter>
        </activity>
```

在代码中定义文件选择的 Activity

接下来，定义一个 Activity 子类，显示从应用程序的 files/images/ 目录在内部存储中可用的文件，并允许用户选择所需的文件。 以下代码段演示了如何定义此 Activity 并响应用户的选择：

```
public class MainActivity extends Activity {
    // The path to the root of this app's internal storage
    private File mPrivateRootDir;
    // The path to the "images" subdirectory
    private File mImagesDir;
    // Array of files in the images subdirectory
    File[] mImageFiles;
    // Array of filenames corresponding to mImageFiles
    String[] mImageFilenames;
    // Initialize the Activity
    @Override
    protected void onCreate(Bundle savedInstanceState) {
    ...
    // Set up an Intent to send back to apps that request a file
    mResultIntent =
            new Intent("com.example.myapp.ACTION_RETURN_FILE");
    // Get the files/ subdirectory of internal storage
    mPrivateRootDir = getFilesDir();
    // Get the files/images subdirectory;
    mImagesDir = new File(mPrivateRootDir, "images");
    // Get the files in the images subdirectory
    mImageFiles = mImagesDir.listFiles();
    // Set the Activity's result to null to begin with
    setResult(Activity.RESULT_CANCELED, null);
    /*
     * Display the file names in the ListView mFileListView.
     * Back the ListView with the array mImageFilenames, which
     * you can create by iterating through mImageFiles and
     * calling File.getAbsolutePath() for each File
     */
     ...
       }
...
}
```

响应文件选择

一旦用户选择了共享文件，你的应用程序必须确定选择了哪个文件，然后为该文件生成内容 URI。 由于 Activity 显示 ListView 中的可用文件列表，当用户单击文件名时，系统调用方法 onItemClick（），在其中可以获取所选文件。

当使用意图将文件的 URI 从一个应用程序发送到另一个应用程序时，你必须小心获取其他应用程序可以读取的 URI。 在运行 Android 6.0（API级别23）及更高版本的设备上执行此操作时需要特别注意，因为该版本 Android 中的权限模型发生了更改，特别是 READ_EXTERNAL_STORAGE 已成为危险权限，接收应用可能缺少此权限。

考虑到这些因素，我们建议你避免使用 Uri.fromFile（）)，这存在一些缺点。 这种方法：

- 不允许在配置文件之间共享文件。
- 要求你的应用拥有 WRITE_EXTERNAL_STORAGE 权限，当执行在 Android 4.4（API等级19）或更低版本的系统上。
- 要求接收应用具有 READ_EXTERNAL_STORAGE 权限，在没有这项权限的重要应用（例如 Gmail ）上，请求将会失败。

替代 Uri.fromFile（）)，你可以使用 URI 权限授予其他应用程序访问特定的 URI。尽管 URI 权限对由 Uri.fromFile（）) 生成的 file：// URI 不起作用，但它们在与内容提供者相关联的 URI 上工作。 FileProvider API 可以帮助你创建此类 URI。此方法也适用于不在外部存储中的文件，但在发送意图的应用程序的本地存储中。

在 onItemClick（）中，为所选文件的文件名获取 File 对象，并将其作为参数传递给 getUriForFile（），以及在 FileProvider 的 <provider> 元素中指定的权限。结果内容 URI 包含权限，对应于文件目录（如 XML 元数据中指定的）的路径段以及包括其扩展名的文件的名称。 FileProvider 如何将目录映射到基于 XML 元数据的路径段，请参阅指定可分享目录一节。

以下代码段显示了如何检测所选文件并获取其内容 URI：

```
    protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks on a file in the l
    mFileListView.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
        @Override
        /*
```

```
            * When a filename in the ListView is clicked, get its
            * content URI and send it to the requesting app
            */
           public void onItemClick(AdapterView<?> adapterView,
                   View view,
                   int position,
                   long rowId) {
               /*
                * Get a File for the selected file name.
                * Assume that the file names are in the
                * mImageFilename array.
                */
               File requestFile = new File(mImageFilename[position]);
               /*
                * Most file-related method calls need to be in
                * try-catch blocks.
                */
               // Use the FileProvider to get a content URI
               try {
                   fileUri = FileProvider.getUriForFile(
                           MainActivity.this,
                           "com.example.myapp.fileprovider",
                           requestFile);
               } catch (IllegalArgumentException e) {
                   Log.e("File Selector",
                         "The selected file can't be shared: " +
                         clickedFilename);
               }
               ...
           }
       });
       ...
   }
```

请记住，您只能为驻留在包含 < path > 元素的元数据文件中指定的目录中的文件生成内容 URI，如指定可分享目录一节中所述。 如果对未指定的路径中的文件调用 getUriForFile（），则会收到 IllegalArgumentException。

授予文件权限

现在你有要分享到另外一个应用的文件 URI 地址，你需要允许客户端应用能访问这个文件。 要允许访问，通过将内容 URI 添加到 Intent，然后在 Intent 上设置权限标志，向客户端应用程序授予权限。 您授予的权限是临时的，并在接收应用程序的任务堆栈完成时自动过期。

以下代码段显示如何为文件设置读取权限：

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks in the ListView
    mFileListView.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> adapterView,
                View view,
                int position,
                long rowId) {
            ...
            if (fileUri != null) {
                // Grant temporary read permission to the content U
                mResultIntent.addFlags(
                    Intent.FLAG_GRANT_READ_URI_PERMISSION);
            }
            ...
         }
         ...
    });
...
}
```

> 警告：调用 setFlags（）是使用临时访问权限安全授予对文件的访问权限的唯一方法。 避免为文件的内容 URI 调用 Context.grantUriPermission（）方法，因为此方法授予你只能通过调用 Context.revokeUriPermission（）撤消的访问。

不要使用 Uri.fromFile（）。 它强制接收应用程序具有 READ_EXTERNAL_STORAGE 权限，如果你尝试在所有用户之间共享，则不会工作;在低于 4.4（API级别19）的 Android 版本中，你的应用程序需要拥有 WRITE_EXTERNAL_STORAGE。 而且真正重要的共享目标（例如 Gmail 应用）没有 READ_EXTERNAL_STORAGE，导致此调用失败。 相反，你可以使用 URI 权限授予其他应用程序对特定 URI 的访问权限。 虽然 URI 权限对由 Uri.fromFile（）生成的 file：// URIs 不起作用，但它们在与内容提供者相关联的 Uris 上工作。 而不是实现自己的，为此，你可以和应该使用 FileProvider ，如文件共享中所述。

与请求应用程序共享文件

要与请求它的应用程序共享文件，请将包含内容 URI 和权限的 Intent 传递给 setResult（）。 当刚刚定义的 Activity 完成后，系统将包含内容 URI 的 Intent 发送到客户端应用程序。 以下代码段显示了如何执行此操作：

```
    protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks on a file in the l
    mFileListView.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> adapterView,
                View view,
                int position,
                long rowId) {
            ...
            if (fileUri != null) {
                ...
                // Put the Uri and MIME type in the result Intent
                mResultIntent.setDataAndType(
                        fileUri,
                        getContentResolver().getType(fileUri));
                // Set the result
                MainActivity.this.setResult(Activity.RESULT_OK,
                        mResultIntent);
                } else {
                    mResultIntent.setDataAndType(null, "");
                    MainActivity.this.setResult(RESULT_CANCELED,
                            mResultIntent);
                }
            }
    });
```

为用户提供一种在选择文件后立即返回到客户端应用程序的方法。一种方法是提供复选标记或完成按钮。使用按钮的 android：onClick 属性将方法与按钮相关联。在方法中，调用 finish（）。例如：

```
 public void onDoneClick(View v) {
    // Associate a method with the Done button
    finish();
}
```

> 备注
> 翻译：@jarylan
> 校对：@iOnesmile
> 原始文档: https://developer.android.com/training/secure-file-sharing/share-file.html

请求共享文件

当应用程序想要访问由其他应用程序共享的文件时，请求应用程序（客户端）通常会向共享文件的应用程序（服务器）发送请求。在大多数情况下，请求在服务器应用程序中启动一个 Activity ，显示它可以共享的文件。 用户选择一个文件，之后服务器应用程序将文件的内容URI返回给客户端应用程序。

本课将向您展示客户端应用程序如何从服务器应用程序请求文件，从服务器应用程序接收文件的内容URI，以及如何使用内容URI打开文件。

发送文件请求

要从服务器应用程序请求文件，客户端应用程序使用包含操作（如 ACTION_PICK ）和客户端应用程序可以处理的 MIME 类型的 Intent 调用 startActivityForResult 。

例如，以下代码段演示了如何向服务器应用程序发送 Intent 以启动共享文件中描述的Activity ：

```
    public class MainActivity extends Activity {
    private Intent mRequestFileIntent;
    private ParcelFileDescriptor mInputPFD;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mRequestFileIntent = new Intent(Intent.ACTION_PICK);
        mRequestFileIntent.setType("image/jpg");
        ...
    }
    ...
    protected void requestFile() {
        /**
         * When the user requests a file, send an Intent to the
         * server app.
         * files.
         */
            startActivityForResult(mRequestFileIntent, 0);
        ...
    }
    ...
}
```

访问请求的文件

服务器应用程序在 Intent 中将文件的内容URI发送回客户端应用程序。 这个 Intent 传递给客户端应用程序的覆盖 startActivityForResult 。 一旦客户端应用程序具有文件的内容URI，它就可以通过获取其 FileDescriptor 来访问文件。

在此过程中保留文件安全性，因为内容URI是客户端应用程序接收的唯一数据。 由于此URI不包含目录路径，因此客户端应用程序无法发现和打开服务器应用程序中的任何其他文件。 只有客户端应用程序可以访问该文件，并且只有服务器应用程序授予的权限。 权限是临时的，因此一旦客户端应用程序的任务堆栈完成，该文件就不再在服务器应用程序之外访问。

下一个片段演示了客户端应用程序如何处理从服务器应用程序发送的 Intent ，以及客户端应用程序如何使用内容URI获取 FileDescriptor ：

```
/*
 * When the Activity of the app that hosts files sets a result and
 * finish(), this method is invoked. The returned Intent contains t
 * content URI of a selected file. The result code indicates if the
 * selection worked or not.
 */
@Override
public void onActivityResult(int requestCode, int resultCode,
        Intent returnIntent) {
    // If the selection didn't work
    if (resultCode != RESULT_OK) {
        // Exit without doing anything else
        return;
    } else {
        // Get the file's content URI from the incoming Intent
        Uri returnUri = returnIntent.getData();
        /*
         * Try to open the file for "read" access using the
         * returned URI. If the file isn't found, write to the
         * error log and return.
         */
        try {
            /*
             * Get the content resolver instance for this context,
             * to get a ParcelFileDescriptor for the file.
             */
            mInputPFD = getContentResolver().openFileDescriptor(ret
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            Log.e("MainActivity", "File not found.");
            return;
        }
        // Get a regular file descriptor for the file
        FileDescriptor fd = mInputPFD.getFileDescriptor();
        ...
    }
}
```

方法 openFileDescriptor（）为文件返回一个 ParcelFileDescriptor 。 从此对象，客户端应用程序获取一个 FileDescriptor 对象，然后可以使用它来读取文件。

翻译人：JackWaiting

原文地址：https://developer.android.com/training/secure-file-sharing/request-file.html#OpenFile

获取文件信息

在一个客户端应用试图获取一个文件的内容 URI 之前，应该先从服务器请求获取这个文件的信息，包括文件的数据类型和文件大小。数据类型可以帮助客户端应用确认是否可以操作此文件，文件大小可以有助于应用建立建立缓冲和缓存。

本课将展示如何通过查询服务端应用程序的 FileProvider 来获取文件的 MIME (*MIME*：全称 *Multipurpose Internet Mail Extensions*，多功能 *Internet* 邮件扩充服务。它是一种多用途网际邮件扩充协议) 类型和文件大小。

获取文件的 MIME 类型

客户端应用要根据一个文件的数据类型去判断如何处理文件的内容。客户端应用可以通过 ContentResolver.getType()) 去获取一个包含 URI 的分享文件的数据类型。该方法返回文件的 MIME 类型。默认情况下，一个 FileProvider 通过文件的后缀名来确定其 MIME 类型。

接下来的代码片段介绍了服务端把 Content URI 反馈给客户端应用后，客户端如何获取文件的 MIME 类型：

```
    ...
    /*
     * Get the file's content URI from the incoming Intent, then
     * get the file's MIME type
     */
    Uri returnUri = returnIntent.getData();
    String mimeType = getContentResolver().getType(returnUri);
    ...
```

获取文件名和文件大小

FileProvider 类有一个 query()) 方法的默认实现，它返回一个 Cursor 对象，该 Cursor 对象包含了 Content URI 所关联的文件的名称和大小。默认的实现返回下面两列信息：

DISPLAY_NAME

文件名，String 类型。这个值和 File.getName()) 所返回的值一样。

SIZE

文件大小，以字节为单位，`long` 类型。这个值和 File.length()) 所返回的值一样。

客户端应用可以通过将 query()) 的除了 Content URI 之外的其他参数都设置为 `null`，来同时获取文件的 DISPLAY_NAME 和 SIZE。例如，下面的代码片段获取一个文件的 DISPLAY_NAME 和 SIZE，然后在两个 TextView 中将他们显示出来：

```
/*
     * Get the file's content URI from the incoming Intent,
     * then query the server app to get the file's display name
     * and size.
     */
    Uri returnUri = returnIntent.getData();
    Cursor returnCursor =
            getContentResolver().query(returnUri, null, null, null,
    /*
     * Get the column indexes of the data in the Cursor,
     * move to the first row in the Cursor, get the data,
     * and display it.
     */
    int nameIndex = returnCursor.getColumnIndex(OpenableColumns.DIS
    int sizeIndex = returnCursor.getColumnIndex(OpenableColumns.SIZ
    returnCursor.moveToFirst();
    TextView nameView = (TextView) findViewById(R.id.filename_text)
    TextView sizeView = (TextView) findViewById(R.id.filesize_text)
    nameView.setText(returnCursor.getString(nameIndex));
    sizeView.setText(Long.toString(returnCursor.getLong(sizeIndex))
    ...
```

> 翻译：@misparking
> 校对：@ifeegoo
> 原始文档：https://developer.android.google.cn/training/secure-file-sharing/retrieve-info.html

向其他设备发送文件

这节课教你如何在你的应用中使用 Android Beam 文件传输来向另外一个设备发送大文件。发送文件之前，你需要请求使用 NFC 和内部存储的权限，测试确保你的设备支持 NFC，给 Android Beam 文件传输提供 URI。

Android Beam 文件传输功能有以下要求：

1.Android Beam 大文件传输功能在 Android 4.1（API Level 16）以及更高版本上才能使用。

2.你要传输的文件必须位于外部存储。更多关于外部存储的使用，请参阅外部存储的使用。

3.每一个你想要传输的文件必须是全局可读的。你可以通过调用 `File.setReadable(true,false)` 方法来设置这个权限。

4.你必须为你需要传输的文件提供一个文件 URI。Android Beam 文件传输无法处理通过 `FileProvider.getUriForFile` 生成的内容 URI。

Manifest 文件中声明特性

首先，编辑应用中的 Manifest 文件来声明应用需要的权限和特性。

请求权限

为了能够让你的应用通过 NFC 使用 Android Beam 文件传输来发送外部存储的文件，你应当在应用 Manifest 文件中请求以下权限：

NFC

允许你的应用通过 NFC 发送数据。要指定此权限，需要添加以下元素作为 `<manifest>` 的子元素：

```
<uses-permission android:name="android.permission.NFC" />
```

READ_EXTERNAL_STORAGE

允许你的应用读取外部存储。要指定此权限，需要添加以下元素作为 `<manifest>` 的子元素：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STO
```

> 备注：从 Android 4.2.2 (API Level 17）开始，这个权限不再强制了。针对想要读取外部存储的应用，将来的平台版本可能会有此权限要求。为了确保向前兼容性，在强制之前，我们现在就请求此权限。

指定 NFC 特性

可以通过追加一个 `<uses-feature>` 的 `<manifest>` 的子元素，来声明你的应用需要使用 NFC。将 `android:required` 属性设置为 `true` 的话，就表明只有当前 NFC 连接打开时应用才能使用。

以下代码片段教你如何指定 `<uses-feature>` 元素：

```
<uses-feature
    android:name="android.hardware.nfc"
    android:required="true" />
```

备注：如果你的应用对 NFC 的使用只是一个可选项，但是如果当前 NFC 功能不能使用的话，仍然可以工作，你应当将 `android:required` 属性设置成 `false`，在代码中判断是否支持 NFC。

指定 Android Beam 文件传输

因为 Android Beam 文件传输在 Android 4.1 （API level 16）及以上版本才支持，如果你的应用依赖 Android Beam 文件传输作为功能的关键部分，那么你用 `android:minSdkVersion="16"` 属性来指定 `<uses-sdk>` 元素。

测试 Android Beam 文件传输的支持

如果在应用 manifest 文件中指定 NFC 是可选的，你需要使用以下元素：

```
<uses-feature android:name="android.hardware.nfc" android:required=
```

如果你设置 `android:required="false"` 属性的话，你必须在代码中判断是否有 NFC 功能和支持 Android Beam 文件传输功能。

可以通过调用 `PackageManager.hasSystemFeature()` 方法传入 `FEATURE_NFC` 参数来判断设备是否支持 NFC 功能。然后通过判断 `SDK_INT` 的值来确定 Android 系统是否支持 Android Beam 文件传输功能。如果 Android Beam 文件传输功能支持的话，可以通过获取一个 NFC 控制器的实例，这个实例可以让你和 NFC 硬件进行通信。例如：

```
public class MainActivity extends Activity {
    ...
    NfcAdapter mNfcAdapter;
    // Flag to indicate that Android Beam is available
    boolean mAndroidBeamAvailable  = false;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // NFC isn't available on the device
        if (!PackageManager.hasSystemFeature(PackageManager.FEATURE
            /*
             * Disable NFC features here.
             * For example, disable menu items or buttons that acti
             * NFC-related features
             */
            ...
        // Android Beam file transfer isn't supported
        } else if (Build.VERSION.SDK_INT <
                Build.VERSION_CODES.JELLY_BEAN_MR1) {
            // If Android Beam isn't available, don't continue.
            mAndroidBeamAvailable = false;
            /*
             * Disable Android Beam file transfer features here.
             */
            ...
        // Android Beam file transfer is available, continue
        } else {
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        ...
        }
    }
    ...
}
```

创建提供文件的回调方法

一旦你确认设备支持 Android Beam 文件传输，添加一个系统回调方法，这个方法会在 Android Beam 文件传输检测到用户想要向另外一个启动 NFC 设备发送文件的动作。在这个回调方法中，会返回一个 Uri 对象的数组。Android Beam 文件传输会复制这些用 URI 表示的文件到接收的设备。

要想添加这个回调方法，需要实现 `NfcAdapter.CreateBeamUrisCallback` 接口以及它的 `createBeamUris()`)方法。以下代码片段告诉你如何做：

```
public class MainActivity extends Activity {
    ...
    // List of URIs to provide to Android Beam
    private Uri[] mFileUris = new Uri[10];
    ...
    /**
     * Callback that Android Beam file transfer calls to get
     * files to share
     */
    private class FileUriCallback implements
            NfcAdapter.CreateBeamUrisCallback {
        public FileUriCallback() {
        }
        /**
         * Create content URIs as needed to share with another devi
         */
        @Override
        public Uri[] createBeamUris(NfcEvent event) {
            return mFileUris;
        }
    }
    ...
}
```

一旦你实现了这个接口，你需要通过调用 `setBeamPushUrisCallback()`) 方法来传递回调给 Android Beam 文件传输。以下代码片段告诉你如何做：

```
public class MainActivity extends Activity {
    ...
    // Instance that returns available files from this app
    private FileUriCallback mFileUriCallback;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Android Beam file transfer is available, continue
        ...
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        /*
         * Instantiate a new FileUriCallback to handle requests for
         * URIs
         */
        mFileUriCallback = new FileUriCallback();
        // Set the dynamic callback for URI requests.
        mNfcAdapter.setBeamPushUrisCallback(mFileUriCallback,this);
        ...
    }
    ...
}
```

> 备注：你也可以通过你应用的 `NfcAdapter` 的实例直接给 NFC 框架提供 Uri 对象数组。如果你可以在 NFC 触摸时间发生之前就可以定义你需要传输的 URI 的话，可以选择这种方法。更多关于此方法，参见 `NfcAdapter.setBeamPushUris()`。

指定传输文件

要想向其他支持 NFC 功能的设备传输一个或多个文件的话，需要获取每一个文件的 URI（带有文件 scheme 的 URI），然后将这个 URI 添加到 Uri 对象数组中。传输一个文件，你需要拥有这个文件的完全读写权限。例如，以下代码片段会告知你如何从一个文件名中获取文件 URI，然后添加 URI 到数组中：

```
        /*
         * Create a list of URIs, get a File,
         * and set its permissions
         */
        private Uri[] mFileUris = new Uri[10];
        String transferFile = "transferimage.jpg";
        File extDir = getExternalFilesDir(null);
        File requestFile = new File(extDir, transferFile);
        requestFile.setReadable(true, false);
        // Get a URI for the File and add it to the list of URIs
        fileUri = Uri.fromFile(requestFile);
        if (fileUri != null) {
            mFileUris[0] = fileUri;
        } else {
            Log.e("My Activity", "No File URI available for file."
        }
```

翻译：@ifeegoo
审核：@misparking
原始文档：https://developer.android.google.cn/training/beam-files/send-files.html

从其他的设备接收文件

Android Beam 文件传输文件到接收设备的特殊目录下。它并且使用 Android Media Scanner 扫描复制的文件和添加媒体文件到 MediaStore 提供者。本片课程告诉你如何在文件复制完成后做出响应，并且在接收的设备上找到复制的文件。

响应请求去显示数据

当 Android Beam 文件传输到接收设备完成后， 它会发送一个通知，包含一个 ACTION_VIEW 动作，传输的第一个文件的 MIME 类型和指向都一个文件的 URI 的 Intent。当用户点击这个通知，这个意图将被发送到系统。要想你的应用响应这个意图，为 应该响应的Activity 的 元素添加一个 元素。这个 元素， 添加下面的子元素:

匹配发送 ACTION_VIEW 意图的通知。

匹配一个没有明确种类的意图。

匹配 MIME 类型。仅指定您的应用程序可以处理的 MIME 类型。

```
    <activity
        android:name="com.example.android.nfctransfer.ViewActivity"
        android:label="Android Beam Viewer" >
        ...
        <intent-filter>
            <action android:name="android.intent.action.VIEW"/>
            <category android:name="android.intent.category.DEFAULT
            ...
        </intent-filter>
    </activity>
```

> **Note:** Android Beam 文件传输不是 ACTION_VIEW 意图的唯一来源。接收设备上的其他应用也可能发送具有此动作的意图。处理这种情况将从 Get the directory from a content URI 章节讨论。

请求文件权限

> 读取 Android Beam 文件传输复制到设备的文件,请求权限 READ_EXTERNAL_STORAGE。如例子

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STO
```

如果你想复制传输文件到你应用独有的存储区域，替换成请求权限 WRITE_EXTERNAL_STORAGE。 WRITE_EXTERNAL_STORAGE 包含 READ_EXTERNAL_STORAGE.

> **Note:** 在 Android 4.2.2(API level 17)，这个权限 READ_EXTERNAL_STORAGE 仅当用户选择使才施行。未来版本的平台可能所有的权限都需要这样。为了确保向前兼容，请在需求之前请求许可。

因为你的应用可以控制内部存储区域，当你复制传输的文件到内部存储区域时你不一定需要请求写入权限

获取已复制文件的目录

在一次传输中Android Beam 文件传输复制的所有文件在接收设备的一个文件夹中。Android Beam文件传输通知发送的内容意图的 URI 指向第一个传输的文件。但是，您的应用程序也可能从 Android Beam 文件传输之外的来源接收 ACTION_VIEW 意图。要确定如何处理传入Intent，您需要检查其方案和权限。要获取 URI 的方案，请调用Uri.getScheme（）。 以下代码段显示了如何确定方案并相应地处理URI：

```
public class MainActivity extends Activity {
    ...
    // A File object containing the path to the transferred files
    private File mParentPath;
    // Incoming Intent
```

```
    private Intent mIntent;
    ...
    /*
     * Called from onNewIntent() for a SINGLE_TOP Activity
     * or onCreate() for a new Activity. For onNewIntent(),
     * remember to call setIntent() to store the most
     * current Intent
     *
     */
    private void handleViewIntent() {
        ...
        // Get the Intent action
        mIntent = getIntent();
        String action = mIntent.getAction();
        /*
         * For ACTION_VIEW, the Activity is being asked to display
         * Get the URI.
         */
        if (TextUtils.equals(action, Intent.ACTION_VIEW)) {
            // Get the URI from the Intent
            Uri beamUri = mIntent.getData();
            /*
             * Test for the type of URI, by getting its scheme valu
             */
            if (TextUtils.equals(beamUri.getScheme(), "file")) {
                mParentPath = handleFileUri(beamUri);
            } else if (TextUtils.equals(
                    beamUri.getScheme(), "content")) {
                mParentPath = handleContentUri(beamUri);
            }
        }
        ...
    }
    ...
}
```

从文件的 URI 获取目录

如果传入的 Intent 包含文件 URI ，则 URI 包含文件的绝对文件名，包括完整的目录路径和文件名。对于 Android Beam 文件传输，目录路径指向其他传输文件的位置（如果有）。 要获取目录路径，请获取 URI 的路径部分，其中包含除文件：前缀之外的所有 URI。 从路径部分创建一个文件，然后获取文件的父路径：

```
...
    public String handleFileUri(Uri beamUri) {
        // Get the path part of the URI
        String fileName = beamUri.getPath();
        // Create a File object for this filename
        File copiedFile = new File(fileName);
        // Get a string containing the file's parent directory
        return copiedFile.getParent();
    }
    ...
```

从内容的 **URI** 获取目录

如果传入 Intent 包含内容 URI ，则 URI 可以指向存储在 MediaStore 内容提供者中的目录和文件名。 您可以通过测试 URI 的权限值来检测 MediaStore的内容URI。 MediaStore 的内容 URI 可能来自 Android Beam 文件传输或来自其他应用程序，但在这两种情况下，您都可以检索内容URI的目录和文件名。

您还可以接收传入的 ACTION_VIEW 意图，其中包含除 MediaStore 之外的内容提供者的内容 URI。 在这种情况下，内容 URI 不包含 MediaStore 权限值，并且内容 URI 通常不指向目录。

> **Note:** 对 于Android Beam 文件传输，如果第一个传入文件的MIME类型为“audio / ”，*“image / ”*或“video / *”，则会在 ACTION_VIEW 意图中接收到一个内容 URI， - 相关。Android Beam 文件传输通过在存储传输文件的目录上运行 Media Scanner 来对其传输的媒体文件进行索引。Media Scanner 将其结果写入 MediaStore 内容提供程序，然后将第一个文件的内容 URI 传递回 Android Beam 文件传输。 此内容URI是您在通知 Intent 中收到的 URI。 要获取第一个文件的目录，可以使用内容 URI 从 MediaStore 中检索它。

确定内容提供者

要确定是否可以从内容 URI 检索文件目录，请通过调用 Uri.getAuthority（）来确定与 URI 关联的内容提供者以获取 URI 的权限。 结果有两个可能的值：

MediaStore.AUTHORITY

> 该URI用于由MediaStore跟踪的一个或多个文件。 从MediaStore检索完整的文件名，并

Any other authority value

> 来自另一内容提供商的内容URI。 显示与内容URI相关联的数据，但不获取文件目录。

要获取 MediaStore 内容 URI 的目录，请运行一个查询，指定 Uri 参数的传入内容 URI 和投影的列 MediaColumns.DATA。 返回的 Cursor 包含由 URI 表示的文件的完整路径和名称。 此路径还包含 Android Beam 文件传输刚刚复制到设备的所有其他文件。

以下代码段显示了如何测试内容 URI 的权限，并检索传输文件的路径和文件名：

```
...
    public String handleContentUri(Uri beamUri) {
        // Position of the filename in the query Cursor
        int filenameIndex;
        // File object for the filename
        File copiedFile;
        // The filename stored in MediaStore
        String fileName;
        // Test the authority of the URI
        if (!TextUtils.equals(beamUri.getAuthority(), MediaStore.AU
            /*
             * Handle content URIs for other content providers
             */
        // For a MediaStore content URI
        } else {
            // Get the column that contains the file name
            String[] projection = { MediaStore.MediaColumns.DATA };
            Cursor pathCursor =
                    getContentResolver().query(beamUri, projection,
                    null, null, null);
```

```
            // Check for a valid cursor
            if (pathCursor != null &&
                    pathCursor.moveToFirst()) {
                // Get the column index in the Cursor
                filenameIndex = pathCursor.getColumnIndex(
                        MediaStore.MediaColumns.DATA);
                // Get the full file name including path
                fileName = pathCursor.getString(filenameIndex);
                // Create a File object for the filename
                copiedFile = new File(fileName);
                // Return the parent directory of the file
                return new File(copiedFile.getParent());
             } else {
                // The query didn't work; return null
                return null;
             }
        }
    }
    ...
```

要了解有关从内容提供程序检索数据的详细信息，请参阅从提供程序检索数据一节。

拍照

这一课将讲解如何使用应用相机拿取照片。

假设你正在做一个全球天气地图的服务，需要整合从客户端应用拿取到的天空照片。整合照片仅仅是应用的一个小功能。你想最方便的掉用拍照，而不是重新写一个相机模块。幸运的是大多数 Android 系统设备都最少安装有一个相机应用。这一课，将学习如何调用它去拍照。

请求相机权限

如果你的应用需要拍照功能，在 Google Play 上显示需要相机，去通知应用需要相机。添加 标签到清单文件中：

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera"
                  android:required="true" />
    ...
</manifest>
```

如果你的应用有使用，但是没有请求相机对应的功能，替换设置 android:required 为 false。Google Play 将允许设备没有相机去下载应用。应用有必要在运行时检查相机是否可用，通过调用 hasSystemFeature(PackageManager.FEATURE_CAMERA)) 方法，当不可用时应该让相机功能不可用。

获取照片通过相机应用

在 Android 系统上，通过调用 Intent 描述所要做的事，将动作委托到另外一个应用。设置过程包括三点：Intent本身，调用开启一个 Activity，和当 Activity 返回时去处理图片和数据的代码。

下面的代码功能是调用 Intent 开启拍照。

```
static final int REQUEST_IMAGE_CAPTURE = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_C
    if (takePictureIntent.resolveActivity(getPackageManager()) != n
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAP
    }
}
```

注意这里的 startActivityForResult() 方法是受到 IF 条件保护的，resolveActivity()) 方法会返回能够处理 Intent 的第一个 Activity 组件。做这个检查是非常必要的，因为如果调用 startActivityForResult() 执行的 Intent 没有一个应用能处理，应用将会闪退。所以只有当结果不为空时才执行才是安全的方式。

获取缩略图

打开相机拍照或许并不是你的目的，你可能想拿到从相机中拍到的照片并做一些处理。

系统相机应用会将照片编译成一个小的 Bitmap，放在 Intent 中最后通过 onActivityResult() 返回，通过键 "data" 获取。下面的恢复照片并显示到 ImageView 上。

```
@Override
protected void onActivityResult(int requestCode, int resultCode, In
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESUL
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        mImageView.setImageBitmap(imageBitmap);
    }
}
```

注意：从 *"data"* 中取到的缩略图可能适合用来做图标，并不适合做其它用途，其它用途请采用全尺寸图片。

保存全尺寸照片

当你设置要保存到一个文件时，系统相机会设置一个全尺寸相片。当相机保存图片时，你设置的必须是有效的文件名。

通常，用户拍摄的所有照片都应该是保存在公共的外部存储卡，因为它可以被所有应用访问。适当的分享照片目录通过 getExternalStoragePublicDirectory()) 方法的 DIRECTORY_PICTURES 属性获取，因为方法提供的这个目录是所有应用共享的，读写这个目录需要分别请求 READ_EXTERNAL_STORAGE 和 WRITE_EXTERNAL_STORAGE 权限。写入权限中是包含读取权限的，当写存储卡时需要去请求这一个权限：

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNA
    ...
</manifest>
```

当然如果仅想对自己的应用有效，可用替换使用 getExternalFilesDir()) 提供的目录。在 Android 4.3 以及更低的版本上，写这个目录也是请求 WRITE_EXTERNAL_STORAGE 权限。从 Android 4.4 开始，不用在请求这个权限，因为这个目录其它应用不可访问。所以请求这个权限只需要在比较底的版本上，通过添加 maxSdkVersion 属性设置：

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNA
                     android:maxSdkVersion="18" />
    ...
</manifest>
```

注：通过 getExternalFilesDir()) 和 getFilesDir()) 获取的文件目录里保存的文件，在应用卸载的时候会被删除。

当确定了文件目录后，你需要创建独特的文件名，避免产生同名等冲突。你也可以保存上次使用的文件路径到一个成员变量中。如下例子是通过时间戳为一张新照片返回唯一名字的方法。

```
String mCurrentPhotoPath;

private File createImageFile() throws IOException {
    // Create an image file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").form
    String imageFileName = "JPEG_" + timeStamp + "_";
    File storageDir = getExternalFilesDir(Environment.DIRECTORY_PIC
    File image = File.createTempFile(
        imageFileName,  /* prefix */
        ".jpg",         /* suffix */
        storageDir      /* directory */
    );

    // Save a file: path for use with ACTION_VIEW intents
    mCurrentPhotoPath = image.getAbsolutePath();
    return image;
}
```

使用这个方法可以为照片创建一个有效的文件，如下方式可以创建和调用 Intent：

```
static final int REQUEST_TAKE_PHOTO = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_C
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != n
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {
            // Error occurred while creating the File
            ...
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            Uri photoURI = FileProvider.getUriForFile(this,
                                                    "com.example.andr
                                                    photoFile);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, pho
            startActivityForResult(takePictureIntent, REQUEST_TAKE_
        }
    }
}
```

注：我们使用 getUriForFile(Context, String, File)) 会返回 content:// URI。在 Android 7.0 （API 24）以及更高的版本上，返回的 file:// URI 导致产生 FileUriExposedException。因此，现在更多的通用方式是使用 FileProvider 来储存。

现在，你需要去配置 FileProvider，在应用的清单文件中，添加一个文件提供者到应用中：

```
<application>
   ...
   <provider
        android:name="android.support.v4.content.FileProvider"
        android:authorities="com.example.android.fileprovider"
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/file_paths"></meta-data>
    </provider>
    ...
</application>
```

确定认证字符串是否匹配第二个参数使用 getUriForFile(Context, String, File)。在文件提供者定义的 meta-data 块中，你能看到在专用的资源文件 res/xml/file_paths.xml 中配置的合法路径。下面是请求内容代表的例子：

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="my_images" path="Android/data/com.example.
</paths>
```

路径组件与通过 getExternalFilesDir()) 方法的 Environment.DIRECTORY_PICTURES 拿取的路径是一致的。确定替换的 com.example.package.name 是你应用的真实包名。另外，检查和 external-path 有关的文件与对应的 FileProvider 扩展描述。

添加照片到相册：

通过意图创建一张照片，你可以知道图片是本地的，因为你在第一个地方去保存它。至于其它，推测最简单的方式是从系统媒体提供者中拿取照片。

注：如果你保存照片的路径是通过 getExternalFilesDir() 获取的，那么系统媒体检查器不能访问到这些文件，因为这个目录是私有的。

下面的例子是展示如何去调用系统媒体扫描器，并添加照片到媒体提供者数据库，使它在系统相册和其它应用生效。

```
private void galleryAddPic() {
    Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER
    File f = new File(mCurrentPhotoPath);
    Uri contentUri = Uri.fromFile(f);
    mediaScanIntent.setData(contentUri);
    this.sendBroadcast(mediaScanIntent);
}
```

解析扫描到的图片

管理全尺寸的图片要当心受限的内存。你会发现显示仅仅一点的图片就会出现内存泄露，你可以通过扩展的 JPEG 导入到内存中，可以显著的减少动态堆的使用，它会缩放图片得到一个与显示一致的尺寸。下面的例子展示这个技术。

```
private void setPic() {
    // Get the dimensions of the View
    int targetW = mImageView.getWidth();
    int targetH = mImageView.getHeight();

    // Get the dimensions of the bitmap
    BitmapFactory.Options bmOptions = new BitmapFactory.Options();
    bmOptions.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(mCurrentPhotoPath, bmOptions);
    int photoW = bmOptions.outWidth;
    int photoH = bmOptions.outHeight;

    // Determine how much to scale down the image
    int scaleFactor = Math.min(photoW/targetW, photoH/targetH);

    // Decode the image file into a Bitmap sized to fill the View
    bmOptions.inJustDecodeBounds = false;
    bmOptions.inSampleSize = scaleFactor;
    bmOptions.inPurgeable = true;

    Bitmap bitmap = BitmapFactory.decodeFile(mCurrentPhotoPath, bmO
    mImageView.setImageBitmap(bitmap);
}
```

> 翻译：@iOnesmile
> 原始文档：https://developer.android.google.cn/training/camera/photobasics.html

控制相机

在本课中，我们将讨论如何使用框架 API 直接控制摄像机硬件。

直接控制设备相机需要比从现有相机应用程序中请求图片或视频更多的代码。但是，如果您要构建专门的相机应用程序或者完全集成到应用程序UI中的东西，本课程将向您展示。

打开相机对象

获取 Camera 对象的实例是直接控制相机过程的第一步。作为 Android 自己的相机应用程序，推荐的访问相机的方法是在从 onCreate()) 启动的单独线程上打开相机。这种方法是一个好主意，因为它可能需要一段时间，并可能会杀死UI线程。在更基本的实现中，打开相机可以推迟到 onResume()) 方法，以方便代码重用，并保持控制流程简单。

如果相机已被另一个应用程序使用，调用 Camera.open()) 会抛出异常，因此我们将其包装在一个 try 块中。

```
private boolean safeCameraOpen(int id) {
    boolean qOpened = false;

    try {
        releaseCameraAndPreview();
        mCamera = Camera.open(id);
        qOpened = (mCamera != null);
    } catch (Exception e) {
        Log.e(getString(R.string.app_name), "failed to open Camera'
        e.printStackTrace();
    }

    return qOpened;
}

private void releaseCameraAndPreview() {
    mPreview.setCamera(null);
    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
       }
}
```

自 API 版本 9 以来，相机框架支持多台相机。 如果您使用旧版 API 并调用 open()) 而不带参数，则会获得第一个后置摄像头。

创建相机预览

拍摄照片通常需要您的用户在点击快门之前看到他们的主题的预览。 为此，你可以使用 SurfaceView 绘制相机传感器拾取的预览。

预览课程

要开始显示预览，您需要预览类。 该预览需要实现 android.view.SurfaceHolder.Callback 接口，用于将图像数据从摄像机硬件传递到应用程序。

```
class Preview extends ViewGroup implements SurfaceHolder.Callback {

    SurfaceView mSurfaceView;
    SurfaceHolder mHolder;

    Preview(Context context) {
        super(context);

        mSurfaceView = new SurfaceView(context);
        addView(mSurfaceView);

        // Install a SurfaceHolder.Callback so we get notified when
        // underlying surface is created and destroyed.
        mHolder = mSurfaceView.getHolder();
        mHolder.addCallback(this);
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
...
}
```

在开始实时图像预览之前，必须将预览类传递给 Camera 对象，如下一节所示。

设置并启动预览

相机实例及其相关预览必须以特定顺序创建，相机对象为第一个。 在下面的代码片段中，初始化相机的过程将被封装，以便每当用户执行某些操作才能更改相机时，将通过 setCamera() 方法调用 Camera.startPreview()。 预览也必须在预览类 surfaceChanged() 回调方法中重新启动。

```
public void setCamera(Camera camera) {
    if (mCamera == camera) { return; }

    stopPreviewAndFreeCamera();

    mCamera = camera;

    if (mCamera != null) {
        List<Size> localSizes = mCamera.getParameters().getSupporte
        mSupportedPreviewSizes = localSizes;
        requestLayout();

        try {
            mCamera.setPreviewDisplay(mHolder);
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Important: Call startPreview() to start updating the pre
        // surface. Preview must be started before you can take a p
        mCamera.startPreview();
    }
}
```

修改相机设置

相机设置会更改相机拍摄的方式，从缩放级别到曝光补偿。 此示例仅更改预览大小; 请参阅相机应用程序的源代码更多。

```
public void surfaceChanged(SurfaceHolder holder, int format, int w,
    // Now that the size is known, set up the camera parameters and
    // the preview.
    Camera.Parameters parameters = mCamera.getParameters();
    parameters.setPreviewSize(mPreviewSize.width, mPreviewSize.heig
    requestLayout();
    mCamera.setParameters(parameters);

    // Important: Call startPreview() to start updating the preview
    // Preview must be started before you can take a picture.
    mCamera.startPreview();
}
```

设置预览方向

大多数相机应用程序将显示器锁定到横向模式，因为这是相机传感器的自然方向。此设置不会阻止您拍摄肖像模式照片，因为设备的方向记录在EXIF标题中。setCameraDisplayOrientation()) 方法可以改变预览的显示方式，而不影响图像的记录方式。但是，在 API 级别为 14 之前的 Android 中，您必须先停止预览，然后再更改方向，然后重新启动。

拍照

预览开始后，使用 Camera.takePicture() 方法拍摄照片。您可以创建 Camera.PictureCallback 和 Camera.ShutterCallback 对象，并将它们传递到 Camera.takePicture() 中。

如果要持续抓取图像，可以创建一个实现 onPreviewFrame() 的 Camera.PictureCallback。对于两者之间的内容，您只能捕获所选的预览帧，或设置一个延迟动作来调用 takePicture()。

重新启动预览

拍摄照片后，您必须重新启动预览，然后用户才能拍摄照片。 在此示例中，重新启动是通过重新启动快门按钮完成的。

```
@Override
public void onClick(View v) {
    switch(mPreviewState) {
    case K_STATE_FROZEN:
        mCamera.startPreview();
        mPreviewState = K_STATE_PREVIEW;
        break;

    default:
        mCamera.takePicture( null, rawCallback, null);
        mPreviewState = K_STATE_BUSY;
    } // switch
    shutterBtnConfig();
}
```

停止预览并释放相机

一旦您的应用程序完成使用相机，它是时候清理。 特别是，您必须释放 Camera 对象，否则可能会导致其他应用程序崩溃，包括您自己应用程序的新实例。

什么时候应该停止预览并释放相机？ 那么你的预览表面被破坏是一个非常好的提示，现在是停止预览和释放相机的时间，如预览类中的这些方法所示。

```
public void surfaceDestroyed(SurfaceHolder holder) {
    // Surface will be destroyed when we return, so stop the previe
    if (mCamera != null) {
        // Call stopPreview() to stop updating the preview surface.
        mCamera.stopPreview();
    }
}

/**
 * When this function returns, mCamera will be null.
 */
private void stopPreviewAndFreeCamera() {

    if (mCamera != null) {
        // Call stopPreview() to stop updating the preview surface.
        mCamera.stopPreview();

        // Important: Call release() to release the camera for use
        // applications. Applications should release the camera imm
        // during onPause() and re-open() it during onResume()).
        mCamera.release();

        mCamera = null;
    }
}
```

在课程之前，此过程也是 setCamera() 方法的一部分，因此初始化摄像机始终以停止预览开始。

备注：
翻译：@JackWaiting
原始文档：https://developer.android.com/training/camera/cameradirect.html#TaskOrientation

照片打印

拍照和分享照片是移动设备最常用的功能之一。如果在我们的应用中需要拍照并且展示照片，或者允许用户共享图片，那么就需要考虑在应用中可以打印这些图片。The Android Support Library 提供了一个方便的功能，只需要使用少量的代码和简单的打印布局选择设置就可以进行图片打印。

这节课就是来讲述如何使用 V4 支持库中的 PrintHelper 类来打印一张图片。

打印图片

PrintHelper 是 Android 支持的库类，它提供了一种简单的打印图片的方式。这个类有一个单一的布局选项：setScaleMode)，它支持下面二种打印选项的其中一个：

- SCALE_MODE_FIT-这个选项会调整图片大小，以便整个图片可以显示在打印区域内。
- SCALE_MODE_FILL-这个选项会调整图片比例，可以全部填充打印范围。选择这个选项意味着顶部和底部，或者左侧和右侧的边缘是不能打印的。如果不设置图片的布局选项，这个选项就是默认的。

`setScaleMode()`) 的二个选项都可以保证图片原始比例的完整性。接下来的代码样例将展示如何创建 PrintHelper 类的实例，设置布局选项并开始打印进程：

```
private void doPhotoPrint() {
    PrintHelper photoPrinter = new PrintHelper(getActivity());
    photoPrinter.setScaleMode(PrintHelper.SCALE_MODE_FIT);
    Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
            R.drawable.droids);
    photoPrinter.printBitmap("droids.jpg - test print", bitmap);
}
```

该方法可以被看作一个菜单项的操作。注意那些设备不一定支持的菜单项（如打印）应该放在菜单中的“更多”中。有关更新多信息，请参阅 Action Bar 设计指南。

在调用 printBitmap()) 后，应用不需要再进行其他操作了。Android 打印用户界面会随后出现，允许用户选择打印机和打印选项。用户可以选择打印图片或者取消打印。如果用户选择了打印图片，则会创建打印任务并在通知栏中显示打印通知。

如果你想要打印的不仅仅是图片，你必须构建一个打印文档。有关打印文档的信息，请参阅 Printing an HTML Document 或 Printing Custom Documents课程。

> 翻译：@misparking
> 校对：@ifeegoo
> 原始文档：https://developer.android.google.cn/training/printing/photos.html#image

打印HTML文档

在 Android 中打印内容超越一个简单的照片需要组合打印文本和图形的文档。Android 框架提供一种用 HTML 方式使用最少的代码去打印文档。

在Android 4.4(API level 19), WebVie 类已更新开启打印 HTML 内容。这个类允许你去加载一个本地的 HTML 资源或者从网络下载的页面。创建一个打印任务,并把它传递给 Android 的打印服务。

本课程将向您展示如何快速构建包含文本和图形的HTML文档，并使用 WebView 进行打印。

加载HTML文档

使用 WebView 打印 HTML文档包含加载 HTML 资源或使用 HTML 文档的字符串。本节介绍如何构建 HTML 字符串并将其加载到 WebView 进行打印。

通常使用这个视图对象作为一个活动布局的一部分。然而,如果您的应用程序不使用 WebView,您可以创建类的实例专门为印刷的目的。创建自定义打印视图的主要步骤是:

1. 创建一个在加载 HTML 资源之后开始打印任务的 WebViewClient
2. 将 HTML 资源加载 到 ebView 对象中

以下代码示例演示了如何创建一个简单 的 WebViewClient 并加载一个即时创建的 HTML 文档：

```
private WebView mWebView;

private void doWebViewPrint() {
    // Create a WebView object specifically for printing
    WebView webView = new WebView(getActivity());
    webView.setWebViewClient(new WebViewClient() {

            public boolean shouldOverrideUrlLoading(WebView view, S
                return false;
            }

            @Override
            public void onPageFinished(WebView view, String url) {
                Log.i(TAG, "page finished loading " + url);
                createWebPrintJob(view);
                mWebView = null;
            }
    });

    // Generate an HTML document on the fly:
    String htmlDocument = "<html><body><h1>Test Content</h1><p>Test
            "testing, testing...</p></body></html>";
    webView.loadDataWithBaseURL(null, htmlDocument, "text/HTML", "U

    // Keep a reference to WebView object until you pass the PrintD
    // to the PrintManager
    mWebView = webView;
}
```

> **NOTE:** 确保您在生成打印作业的调用发生在你在上一节中创建的 WebViewClient 的 onPageFinished（） 方法中。 如果你不等待页面加载完成，打印输出可能不完整或空白，否则可能会完全失败。
>
> **NOTE:** 上面的示例代码包含 WebView 对象的实例，因此在创建打印作业之前不会垃圾回收。 确保你在自己的实现中执行相同操作，否则打印过程可能会失败。

如果要在页面中包含图形，请将图形文件放在项目的 assets /目录中，并在 loadDataWithBaseURL（） 方法的第一个参数中指定一个基本 URL，如下面的代码示例所示：

```
webView.loadDataWithBaseURL("file:///android_asset/images/", htmlBo
        "text/HTML", "UTF-8", null);
```

你还可以通过用 loadUrl（） 替换loadDataWithBaseURL（）方法来加载要打印的网页，如下所示。

```
// Print an existing web page (remember to request INTERNET permiss
webView.loadUrl("http://developer.android.com/about/index.html");
```

当使用 WebView 创建打印文档时，你应该注意以下限制：

- 你不能向文档添加页眉或页脚，包括页码。
- HTML 文档的打印选项不包括打印页面范围的功能，例如：不支持打印10页 HTML 文档中的第2页到第4页。
- WebView 的一个实例一次只能处理一个打印作业。不支持包含CSS打印属性（如风景属性）的 HTML 文档。
- 你不能在 HTML 文档中使用 JavaScript 来触发打印。

> **NOTE** 布局中包含的 WebView 对象的内容也可以在加载文档时打印。

创建打印作业

在创建 WebView 并加载 HTML 内容之后，您的应用程序几乎完成了其部分打印过程。 接下来的步骤是访问 PrintManager，创建打印适配器，最后创建打印作业。

```
private void createWebPrintJob(WebView webView) {

    // Get a PrintManager instance
    PrintManager printManager = (PrintManager) getActivity()
            .getSystemService(Context.PRINT_SERVICE);

    // Get a print adapter instance
    PrintDocumentAdapter printAdapter = webView.createPrintDocument

    // Create a print job with name and adapter instance
    String jobName = getString(R.string.app_name) + " Document";
    PrintJob printJob = printManager.print(jobName, printAdapter,
            new PrintAttributes.Builder().build());

    // Save the job object for later status checking
    mPrintJobs.add(printJob);
}
```

此示例保存 PrintJob 对象的实例以供应用程序使用，这不是必需的。您的应用程序可能会使用此对象来跟踪正在处理的打印作业的进度。当您想要在应用程序中监视打印作业的状态以完成，失败或用户取消时，此方法很有用。不需要创建应用内通知，因为打印框架会自动为打印作业创建系统通知。

打印自定义文档

对于某些应用程序，如绘图应用程序，页面布局应用程序和其他专注于图形输出的应用程序，创建漂亮的打印页面是一个关键功能。 在这种情况下，打印图像或 HTML 文档是不够的。 这些类型应用程序的打印输出需要精确控制进入页面的所有内容，包括字体，文本流，分页符，页眉，页脚和图形元素。

为你的应用创建完全定制的打印输出需要比以前讨论的方法更多的编程投入。 您必须构建与打印框架通信的组件，适应打印机设置，绘制页面元素并管理多页打印。

本课程将向您展示如何连接打印管理器，创建打印适配器并构建打印内容。

连接到打印管理器

当你的应用程序直接管理打印过程时，从用户收到打印请求之后的第一步是连接到 Android 打印框架并获取 PrintManager 类的实例。 该类允许你初始化打印作业并开始打印生命周期。 以下代码示例显示如何获取打印管理器并开始打印过程。

```
private void doPrint() {
    // Get a PrintManager instance
    PrintManager printManager = (PrintManager) getActivity()
        .getSystemService(Context.PRINT_SERVICE);

    // Set job name, which will be displayed in the print queue
    String jobName = getActivity().getString(R.string.app_name) + "

    // Start a print job, passing in a PrintDocumentAdapter impleme
       // to handle the generation of a print document
    printManager.print(jobName, new MyPrintDocumentAdapter(getActiv
        null); //
}
```

以上示例代码演示如何命名打印作业并设置 PrintDocumentAdapter 类的实例，该实例处理打印生命周期的步骤。下一节将讨论打印适配器类的实现。

> 注意：print（）方法中的最后一个参数会使用一个PrintAttributes对象。您可以使用此参数为打印框架提供提示，并根据先前的打印周期预设选项，从而改善用户体验。您也可以使用此参数来设置更适合于正在打印的内容的选项，例如在打印方向的照片时将方向设置为横向。

创建打印适配器

打印适配器与Android打印框架进行交互，并处理打印过程的步骤。此过程需要用户在创建要打印的文档之前选择打印机和打印选项。当用户选择具有不同输出功能，不同页面大小或不同页面方向的打印机时，这些选择可以影响最终输出。当进行这些选择时，打印框架要求您的适配器布局并生成打印文档，以准备最终输出。一旦用户点击打印按钮，框架将使用最终的打印文档并将其传递给打印提供者进行输出。在打印过程中，用户可以选择取消打印动作，因此您的打印适配器还必须收听取消请求并作出反应。

PrintDocumentAdapter抽象类旨在处理打印生命周期，它具有四个主要的回调方法。您必须在打印适配器中实现这些方法，才能与打印框架正常交互：

- onStart（） 在打印过程开始时调用一次。如果您的应用程序可以执行任何一次性的准备任务，例如获取要打印的数据的快照，请在此处执行。不需要在适配器中实现此方法。
- onLayout（） - 每次用户更改影响输出的打印设置（例如不同的页面大小或页面方向）时，调用它们，使您的应用程序有机会计算要打印的页面的布局。至少，此方法必须返回打印文档中预期的页数。
- onWrite（） - 调用将打印页面复制到要打印的文件中。在每次onLayout（）调用之后，此方法可能会被调用一次或多次。
- onFinish（） - 在打印过程结束时调用一次。如果您的应用程序有任何一次执行任务，请在此处执行。不需要在适配器中实现此方法。

以下部分介绍如何实现布局和写入方法，这对打印适配器的功能至关重要。

> 注意：这些适配器方法在应用程序的主线程上调用。如果您期望在执行中执行这些方法花费大量时间，请执行它们在单独的线程中执行。例如，您可以将布局或打印文档编写工作封装在单独的AsyncTask对象中。

计算打印文档信息

在PrintDocumentAdapter类的实现中，您的应用程序必须能够指定正在创建的文档的类型，并计算打印作业的总页数（给定有关打印页面大小的信息）。 在适配器中实现onLayout（）方法进行这些计算，并提供有关PrintDocumentInfo类中打印作业的预期输出的信息，包括页数和内容类型。 以下代码示例显示了PrintDocumentAdapter的onLayout（）方法的基本实现：

```
@Override
public void onLayout(PrintAttributes oldAttributes,
                     PrintAttributes newAttributes,
                     CancellationSignal cancellationSignal,
                     LayoutResultCallback callback,
                     Bundle metadata) {
    // Create a new PdfDocument with the requested page attributes
    mPdfDocument = new PrintedPdfDocument(getActivity(), newAttribu

    // Respond to cancellation request
    if (cancellationSignal.isCancelled() ) {
        callback.onLayoutCancelled();
        return;
    }

    // Compute the expected number of printed pages
    int pages = computePageCount(newAttributes);

    if (pages > 0) {
        // Return print information to print framework
        PrintDocumentInfo info = new PrintDocumentInfo
                .Builder("print_output.pdf")
                .setContentType(PrintDocumentInfo.CONTENT_TYPE_DOCU
                .setPageCount(pages);
                .build();
        // Content layout reflow is complete
        callback.onLayoutFinished(info, true);
    } else {
        // Otherwise report an error to the print framework
        callback.onLayoutFailed("Page count calculation failed.");
    }
}
```

在不能完成布局计算的情况下，onLayout（）方法的执行可以有三个结果：完成，取消或失败。 您必须通过调用PrintDocumentAdapter.LayoutResultCallback对象的相应方法来指示其中一个结果。

> 注意：onLayoutFinished（）方法的布尔参数指示布局内容自上次请求以来是否实际上已更改。 正确设置此参数允许打印框架避免不必要地调用onWrite（）方法，基本上缓存先前写入的打印文档并提高性能。

onLayout（）的主要工作是计算给定打印机属性的输出页数。 如何计算这个数字很大程度上取决于您的应用程序如何布局打印页面。 以下代码示例显示了一个实现，其中页数由打印方向决定：

```
private int computePageCount(PrintAttributes printAttributes) {
    int itemsPerPage = 4; // default item count for portrait mode

    MediaSize pageSize = printAttributes.getMediaSize();
    if (!pageSize.isPortrait()) {
        // Six items per page in landscape orientation
        itemsPerPage = 6;
    }

    // Determine number of print items
    int printItemCount = getPrintItemCount();

    return (int) Math.ceil(printItemCount / itemsPerPage);
}
```

写一个打印文档文件

当将打印输出写入文件的时候，Android打印框架调用应用程序的PrintDocumentAdapter类的onWrite（）方法。该方法的参数指定要写入的页面和要使用的输出文件。您的这种方法的实现必须将每个请求的内容页面呈现到多页PDF文档文件。当此过程完成时，您调用回调对象的onWriteFinished（）方法。

> 注意：Android打印框架可以每次调用onLayout（）一次或多次调用onWrite（）方法。因此，当打印内容布局未更改时，将onLayoutFinished（）方法的布尔参数设置为false是很重要的，以避免不必要的重新打印打印文档。
>
> 注意：onLayoutFinished（）方法的布尔参数指示布局内容自上次请求以来是否实际上已更改。正确设置此参数允许打印框架避免不必要地调用onLayout（）方法，基本上缓存以前写入的打印文档并提高性能。

以下示例演示了使用PrintedPdfDocument类创建PDF文件的此过程的基本机制：

```
@Override
public void onWrite(final PageRange[] pageRanges,
                    final ParcelFileDescriptor destination,
                    final CancellationSignal cancellationSignal,
                    final WriteResultCallback callback) {
    // Iterate over each page of the document,
    // check if it's in the output range.
    for (int i = 0; i < totalPages; i++) {
        // Check to see if this page is in the output range.
        if (containsPage(pageRanges, i)) {
            // If so, add it to writtenPagesArray. writtenPagesArra
            // is used to compute the next output page index.
            writtenPagesArray.append(writtenPagesArray.size(), i);
            PdfDocument.Page page = mPdfDocument.startPage(i);

            // check for cancellation
            if (cancellationSignal.isCancelled()) {
                callback.onWriteCancelled();
                mPdfDocument.close();
                mPdfDocument = null;
                return;
            }

            // Draw page content for printing
            drawPage(page);

            // Rendering is complete, so page can be finalized.
            mPdfDocument.finishPage(page);
        }
    }

    // Write PDF document to file
    try {
        mPdfDocument.writeTo(new FileOutputStream(
                destination.getFileDescriptor()));
    } catch (IOException e) {
        callback.onWriteFailed(e.toString());
        return;
```

```
    } finally {
        mPdfDocument.close();
        mPdfDocument = null;
    }
    PageRange[] writtenPages = computeWrittenPages();
    // Signal the print framework the document is complete
    callback.onWriteFinished(writtenPages);

    ...
}
```

该示例将PDF页面内容的呈现委托给drawPage（）方法，这将在下一节中讨论。

与布局一样，在无法写入内容的情况下，onWrite（）方法的执行可以有三个结果：完成，取消或失败。您必须通过调用PrintDocumentAdapter.WriteResultCallback对象的相应方法来指示其中一个结果。

> 注意：渲染用于打印的文档可能是资源密集型的操作。为了避免阻塞应用程序的主用户界面线程，您应该考虑在单独的线程上执行页面呈现和写入操作，例如在AsyncTask中执行。有关处理执行线程（如异步任务）的更多信息，请参阅进程和线程。

绘制PDF页面内容

当您的应用程序打印时，您的应用程序必须生成PDF文档并将其传递到Android打印框架进行打印。您可以使用任何PDF生成库来实现此目的。本课程将介绍如何使用PrintedPdfDocument类从您的内容生成PDF页面。

PrintedPdfDocument类使用Canvas对象在PDF页面上绘制元素，类似于活动布局上的绘图。您可以使用Canvas绘制方法在打印页面上绘制元素。以下示例代码演示了如何使用以下方法在PDF文档页面上绘制一些简单元素：

```
private void drawPage(PdfDocument.Page page) {
    Canvas canvas = page.getCanvas();

    // units are in points (1/72 of an inch)
    int titleBaseLine = 72;
    int leftMargin = 54;

    Paint paint = new Paint();
    paint.setColor(Color.BLACK);
    paint.setTextSize(36);
    canvas.drawText("Test Title", leftMargin, titleBaseLine, paint)

    paint.setTextSize(11);
    canvas.drawText("Test paragraph", leftMargin, titleBaseLine + 2

    paint.setColor(Color.BLUE);
    canvas.drawRect(100, 100, 172, 172, paint);
}
```

当使用Canvas绘制PDF页面时，元素以点为单位指定，为1/72英寸。 请确保使用此度量单位来指定页面上元素的大小。 对于绘制元素的定位，坐标系从页面左上角的0,0开始。

> 提示：虽然Canvas对象允许您将打印元素放置在PDF文档的边缘，但是许多打印机无法打印到实体纸张的边缘。 当您使用此类构建打印文档时，请确保您解决页面的不可打印边缘。

在 Android 应用中用 OpenGL ES 绘制图形，必须先创建一个 View 的容器。最直接的一种方式是实现 GLSurfaceView 和 GLSurfaceView.Renderer。GLSurfaceView 是 OpenGL 绘制图形的 View 容器，GLSurfaceView.Renderer 是对 View 的控制。更多关于这一方面的信息，请见 OpenGL ES 开发指南。

GLSurfaceView 是集成 OpenGL ES 到应用中合适的一种方式。创建一个全屏或进全屏的图形界面，它是很合理的选择。开发者想要集成 OpenGL ES 到 layout 布局中，可以使用 TextureView。事实上，在自己开发时，使用 SurfaceView 去构建一个 OpenGL ES 也是很好的选择，但是这个选择需要编写大量的代码。

这一课将讲述在 Activity 中用最少的实现 GLSurfaceView 和 GLSurfaceView.Renderer 去完成一个简单的应用。

在清单文件中声明 OpenGL ES 的使用

按照下面的方式，在应用中使用 OpenGL ES 2.0 API。你必须添加如下的声明到清单文件中：

```
<uses-feature android:glEsVersion="0x00020000" android:required="tr
```

如果在你应用中使用了压缩方式，你需要声明应用支持的压缩格式，以便于应用只安装在支持的设备上。

```
<supports-gl-texture android:name="GL_OES_compressed_ETC1_RGB8_text
<supports-gl-texture android:name="GL_OES_compressed_paletted_textu
```

更多关于结构压缩格式的信息，参见 OpenGL 开发者指南。

创建 OpenGL ES 图像的 Activity

使用 OpenGL ES 的 Android 应用和其他应用一样有一个用户界面。主要的不用点是其它应用给 Activity 设置的是 layout 布局。很多应用可能有使用 TextView, Button 和 ListView，当要使用 OpenGL ES 时也能添加 GLSurfaceView。

下面的代码是使用 GLSurfaceView 最少实现的示例：

```
public class OpenGLES20Activity extends Activity {

    private GLSurfaceView mGLView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Create a GLSurfaceView instance and set it
        // as the ContentView for this Activity.
        mGLView = new MyGLSurfaceView(this);
        setContentView(mGLView);
    }
}
```

> 注：OpenGL ES 2.0 请求的最小版本是 Android 2.2（API 8）或更高，因此请调整项目的 API。

构建 GLSurfaceView 对象

GLSurfaceView 是针对 OpenGL ES 图形绘制的 View。但是它本身并没有做太多，通常需要设置 GLSurfaceView.Renderer 来绘制对象。事实上，这个对象的代码非常的少，你可能有兴趣不继承它而自己创建一个 GLSurfaceView 对象，但是不会这么做。你需要继承这个类，并重写触摸事件，涉及到 响应触摸事件 这一课。

下面是快速的实现 GLSurfaceView 必备的最少代码，通常会在 Activity 里面创建一个内部内：

```
class MyGLSurfaceView extends GLSurfaceView {

    private final MyGLRenderer mRenderer;

    public MyGLSurfaceView(Context context){
        super(context);

        // Create an OpenGL ES 2.0 context
        setEGLContextClientVersion(2);

        mRenderer = new MyGLRenderer();

        // Set the Renderer for drawing on the GLSurfaceView
        setRenderer(mRenderer);
    }
}
```

其它的操作是给 GLSurfaceView 设置渲染模式，只有当数据改变时才去绘制的模式是 GLSurfaceView.RENDERMODE_WHEN_DIRTY：

```
// Render the view only when there is a change in the drawing data
setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
```

这个设置防止 GLSurfaceView 从开始到你调用 requestRender()) 反复绘制的问题，这个例子更有效。

构建和渲染类

使用 OpenGL ES 时需要实现 GLSurfaceView.Renderer 去做一些有趣的事情，这个类与 GLSurfaceView 绑定去控制它的绘制。下面是三个在 Android 系统中按序在 GLSurfaceView 绘制时被回调的方法：

- onSurfaceCreated()) - 在 view 的 OpenGL ES 环境创建时被调用。
- onDrawFrame()) - 在 View 绘制或重新绘制时的回调。
- onSurfaceChanged()) - 在 View 的结构改变时回调，如设备屏幕方向改变。

这是 OpenGL ES 非常基础渲染的实现，在 GLSurfaceView 中绘制一个黑色背景没有更多：

```
public class MyGLRenderer implements GLSurfaceView.Renderer {

    public void onSurfaceCreated(GL10 unused, EGLConfig config) {
        // Set the background frame color
        GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    }

    public void onDrawFrame(GL10 unused) {
        // Redraw background color
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);
    }

    public void onSurfaceChanged(GL10 unused, int width, int height
        GLES20.glViewport(0, 0, width, height);
    }
}
```

上面这些是实现它的所有步骤！上面的代码是使用 OpenGL 创建了简单的显示黑屏的应用。目前实现的这些代码还没有做任何有趣的事情，但是是使用 OpenGL 绘制最基本的需求。

注：你可能想知道在使用 OpengGL ES 2.0 API 时，为什么这些方法有 GL10 参数。这些方法标记是为了在 2.0 API 上简单的重复使用，并保持 Android 框架代码更简洁。

如果你熟悉了 OpenGL ES API，那么现在你应该可以搭建 OpenGL ES 环境和开始绘制图像。然而，如果需要更多的 OpenGL 帮助信息，继续下一课获取更多信息。

> 翻译：@iOnesmile
> 原始文档：https://developer.android.google.cn/training/graphics/opengl/environment.html#manifest

定义 Shapes

能够在 OpenGL ES 视图的上下文中定义要绘制的形状是创建高端图形杰作的第一步。如果没有一些 OpenGL ES 定义图形对象的基础知识，绘制图形可能会有一点困难。

这节课讲解了相对于 Android 设备屏幕 OpenGL ES 的坐标系，定义形状和形状表面的基本知识，如定义一个三角形和一个下文形。

定义一个三角形

OpenGL ES 允许在三维坐标里定义图像。所以，在画三角形前，必须定义坐标。在OpenGL ES里，要做到这一点的典型方法是定义一个浮点型的顶点数组。为获得做大效率，将坐标写进 ByteBuffer 里，然后传给 OpenGL ES 的图形处理流程中。

```
public class Triangle {

    private FloatBuffer vertexBuffer;

    // number of coordinates per vertex in this array
    static final int COORDS_PER_VERTEX = 3;
    static float triangleCoords[] = {   // in counterclockwise orde
             0.0f,  0.622008459f, 0.0f, // top
            -0.5f, -0.311004243f, 0.0f, // bottom left
             0.5f, -0.311004243f, 0.0f  // bottom right
    };

    // Set color with red, green, blue and alpha (opacity) values
    float color[] = { 0.63671875f, 0.76953125f, 0.22265625f, 1.0f }

    public Triangle() {
        // initialize vertex byte buffer for shape coordinates
        ByteBuffer bb = ByteBuffer.allocateDirect(
                // (number of coordinate values * 4 bytes per float
                triangleCoords.length * 4);
        // use the device hardware's native byte order
        bb.order(ByteOrder.nativeOrder());

        // create a floating point buffer from the ByteBuffer
        vertexBuffer = bb.asFloatBuffer();
        // add the coordinates to the FloatBuffer
        vertexBuffer.put(triangleCoords);
        // set the buffer to read the first coordinate
        vertexBuffer.position(0);
    }
}
```

默认情况下，OpenGL ES 假定一个坐标系，其中 [0,0,0]*(X,Y,Z)* 为 GLSurfaceView 的中心点，[1,1,0] 是指右上角，[-1,-1,0] 即左下角。想要了解更多的坐标系的图解说明，可以参考 OpenGL ES 开发指南。

注意，这种形状的坐标以逆时针顺序被定义。绘制顺序很重要，因为它定义了你要想绘制的哪一面是形状的正面，以及可以选择使用 OpenGL ES 不绘制背面。有关更多信息，请参阅 OpengGL ES 的开发者指南。

定义正方形

在 OpenGL ES 里定义三角形是比较简单的，但如果你想绘制点更复杂的形状呢？比如，一个正方形？有许多方法可以做到这一点，但以在 OpenGL ES 绘制的形状的典型路径是使用两个三角形绘制：

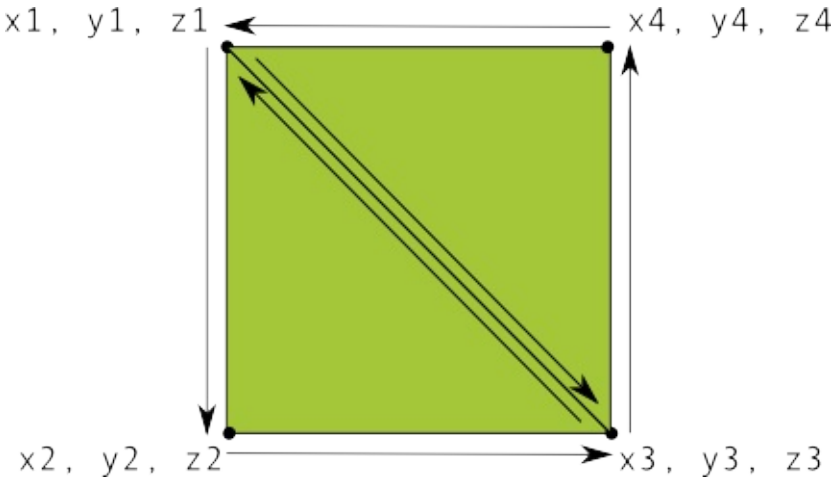


图 1. 使用两个三角形绘制正方形

再次，你应该定义一个逆时针顺序的顶点表示这种形状的三角形，并把值存到 ByteBuffer。为了避免定义由每个三角形使用两次共同坐标，使用绘图列表告诉 OpenGL ES 的图形管道如何绘制这些顶点。下面是这个形状的代码：

```
    public class Square {

    private FloatBuffer vertexBuffer;
    private ShortBuffer drawListBuffer;

    // number of coordinates per vertex in this array
    static final int COORDS_PER_VERTEX = 3;
    static float squareCoords[] = {
            -0.5f,  0.5f, 0.0f,   // top left
            -0.5f, -0.5f, 0.0f,   // bottom left
             0.5f, -0.5f, 0.0f,   // bottom right
             0.5f,  0.5f, 0.0f }; // top right

    private short drawOrder[] = { 0, 1, 2, 0, 2, 3 }; // order to d

    public Square() {
        // initialize vertex byte buffer for shape coordinates
        ByteBuffer bb = ByteBuffer.allocateDirect(
        // (# of coordinate values * 4 bytes per float)
                squareCoords.length * 4);
        bb.order(ByteOrder.nativeOrder());
        vertexBuffer = bb.asFloatBuffer();
        vertexBuffer.put(squareCoords);
        vertexBuffer.position(0);

        // initialize byte buffer for the draw list
        ByteBuffer dlb = ByteBuffer.allocateDirect(
        // (# of coordinate values * 2 bytes per short)
                drawOrder.length * 2);
        dlb.order(ByteOrder.nativeOrder());
        drawListBuffer = dlb.asShortBuffer();
        drawListBuffer.put(drawOrder);
        drawListBuffer.position(0);
    }
}
```

这个例子告诉你用 OpenGL 创建更复杂的形状。一般情况下，用三角形集合绘制对象。在下一课中，你将学习如何在屏幕上绘制这些形状。

> 翻译：@misparking
> 原始文档：https://developer.android.google.cn/training/graphics/opengl/shapes.html

绘制Shapes

在你用OpenGL定义绘制图像之后，你可能想要把它们画出来。它提供的api非常好的处理了控制图形和渲染，所以使用OpenGL ES 2.0绘制图形比你想象中还要多一些代码。

本课说明了如何使用OpenGL ES 2.0 API绘制上一堂课定义的图像。

初始化图形

在进行任何绘图之前，你必须初始化并加载你计划要绘制的图形。除非程序中使用的形状结构（原始坐标）在执行过程中发生变化，你应该在您的渲染器的 onSurfaceCreated() 方法初始化它们，以用于内存和处理效率。

```
public class MyGLRenderer implements GLSurfaceView.Renderer {
    ...
    private Triangle mTriangle;
    private Square   mSquare;

    public void onSurfaceCreated(GL10 unused, EGLConfig config) {
        ...

        // initialize a triangle
        mTriangle = new Triangle();
        // initialize a square
        mSquare = new Square();
    }
    ...
}
```

绘制图形

使用OpenGL ES 2.0 绘制一个定义好的图形需要大量的代码，因此你必须提供非常多图形渲染上的细节，具体来说，你必须定义以下内容：

- 顶点着色器- OpenGL ES图形代码渲染顶点的形状。
- 片段着色器- OpenGL ES代码去渲染图形表面上的颜色和纹路。
- 程序 - 一个OpenGL ES对象，它包含了绘制一个或者多个图形的着色器。

你需要至少一个顶点着色器去绘制形状和一个片段着色器来着色给图形，这些着色器必须遵守，然后增加到OpenGL ES的程序中，然后来绘制形状，下面是一个实例，说明如何定义基本着色器，用于Triangle类的绘制形状。

```
public class Triangle {

    private final String vertexShaderCode =
        "attribute vec4 vPosition;" +
        "void main() {" +
        "  gl_Position = vPosition;" +
        "}";

    private final String fragmentShaderCode =
        "precision mediump float;" +
        "uniform vec4 vColor;" +
        "void main() {" +
        "  gl_FragColor = vColor;" +
        "}";

    ...
}
```

着色器包含OpenGL的着色语言（GLSL）代码，它必须被编译使用在OpenGL ES的环境之前。在编译该代码时，创建渲染器类的实用方法。

```
public static int loadShader(int type, String shaderCode){

    // create a vertex shader type (GLES20.GL_VERTEX_SHADER)
    // or a fragment shader type (GLES20.GL_FRAGMENT_SHADER)
    int shader = GLES20.glCreateShader(type);

    // add the source code to the shader and compile it
    GLES20.glShaderSource(shader, shaderCode);
    GLES20.glCompileShader(shader);

    return shader;
}
```

为了绘制你要的形状，你必须编译该着色器代码，在你的绘制类完成构造函数时，添加它们到OpenGL ES程序类和程序链接，它只会完成了一次。

> 注意: 编译OpenGL ES着色器和链接程序在CPU周期和处理时间上是很昂贵的。因此你应该避免多次操作。如果运行时不知道着色器内容，你应该构建你的代码，这样他们只需创建一次，然后缓存以后使用。

```
public class Triangle() {
    ...

    private final int mProgram;

    public Triangle() {
        ...

        int vertexShader = MyGLRenderer.loadShader(GLES20.GL_VERTEX
                                        vertexShaderCode);
        int fragmentShader = MyGLRenderer.loadShader(GLES20.GL_FRAG
                                        fragmentShaderCode);

        // create empty OpenGL ES Program
        mProgram = GLES20.glCreateProgram();

        // add the vertex shader to program
        GLES20.glAttachShader(mProgram, vertexShader);

        // add the fragment shader to program
        GLES20.glAttachShader(mProgram, fragmentShader);

        // creates OpenGL ES program executables
        GLES20.glLinkProgram(mProgram);
    }
}
```

此时此刻，你该准备添加绘制图形的实际调用了，使用OpenGL ES的绘制图形需要指定几个参数。告诉渲染渠道你想要绘制什么以及如何绘制它。由于绘制选择能根据形状而变化，所以让你的形状类包含自己的绘制逻辑是一个好主意。

创建绘制图形的draw()方法，这代码将位置和颜色值设置为图形的顶点着色器以及片段着色器，然后执行绘制功能。

```
private int mPositionHandle;
private int mColorHandle;

private final int vertexCount = triangleCoords.length / COORDS_PER_
```

```
private final int vertexStride = COORDS_PER_VERTEX * 4; // 4 bytes

public void draw() {
    // Add program to OpenGL ES environment
    GLES20.glUseProgram(mProgram);

    // get handle to vertex shader's vPosition member
    mPositionHandle = GLES20.glGetAttribLocation(mProgram, "vPositi

    // Enable a handle to the triangle vertices
    GLES20.glEnableVertexAttribArray(mPositionHandle);

    // Prepare the triangle coordinate data
    GLES20.glVertexAttribPointer(mPositionHandle, COORDS_PER_VERTEX
                                 GLES20.GL_FLOAT, false,
                                 vertexStride, vertexBuffer);

    // get handle to fragment shader's vColor member
    mColorHandle = GLES20.glGetUniformLocation(mProgram, "vColor");

    // Set color for drawing the triangle
    GLES20.glUniform4fv(mColorHandle, 1, color, 0);

    // Draw the triangle
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);

    // Disable vertex array
    GLES20.glDisableVertexAttribArray(mPositionHandle);
}
Once you have all this code in place, drawing this object just requ

public void onDrawFrame(GL10 unused) {
    ...

    mTriangle.draw();
}
```

一旦你拥有了这些代码，绘制该对象仅仅需要从你渲染器的onDrawFrame()方法内调用draw方法。

```
public void onDrawFrame(GL10 unused) {
    ...

    mTriangle.draw();
}
```

在你跑应用程序的时候，应该看起来像这样:

图**1** 没有投射或照相视图绘制的三角形。

这个代码实例有一些问题，首先，它不会给你的朋友带来深刻的印象，其次，在你改设备的屏幕方向的时候，这个三角形会被压扁，改变形状。图形倾斜的实际原因是由于类对象顶点没有得到屏幕区域修正到正常比例显示GLSurfaceView。在下一堂课中，可以使用投射和相机视图修复他们。

> 翻译：@northJjL
> 审核：@JackWaiting
> 原始文档：https://developer.android.google.cn/training/graphics/opengl/draw.html#draw

应用投影和相机视图

在OpenGL ES环境中，投影和摄像机视图允许你的眼睛看到物体对象更接近的方式显示绘制的对象。物理观察的这种模拟是通过绘制对象坐标的数学变换完成的：

- 投影 - 此变换根据显示 GLSurfaceView 的宽度和高度来调整绘制对象的坐标。没有这个计算， OpenGL ES 绘制的对象被视图窗口的不等比例偏移。在渲染器的 onSurfaceChanged() 方法中建立或更改 OpenGL 视图的比例时，通常只需计算投影变换。有关 OpenGL ES 投影和坐标映射的更多信息，请参阅绘制对象的坐标坐标。
- 相机视图 - 此转换根据虚拟相机位置调整绘制对象的坐标。请注意，OpenGL ES 不会定义实际的摄像机对象，而是提供了通过转换绘制对象的显示来模拟摄像机的实用方法。建立 GLSurfaceView 时，相机视图转换可能只计算一次，或者可能会根据用户操作或应用程序的功能动态更改。

本课介绍如何创建投影和相机视图，并将其应用于 GLSurfaceView 中绘制的形状。

定义投影

投影变换的数据在 GLSurfaceView.Renderer 类的 onSurfaceChanged() 方法中计算。 以下示例代码使用 GLSurfaceView 的高度和宽度，并使用它使用 Matrix.frustumM()) 方法填充投影变换矩阵：

```
// mMVPMatrix is an abbreviation for "Model View Projection Matrix"
private final float[] mMVPMatrix = new float[16];
private final float[] mProjectionMatrix = new float[16];
private final float[] mViewMatrix = new float[16];

@Override
public void onSurfaceChanged(GL10 unused, int width, int height) {
    GLES20.glViewport(0, 0, width, height);

    float ratio = (float) width / height;

    // this projection matrix is applied to object coordinates
    // in the onDrawFrame() method
    Matrix.frustumM(mProjectionMatrix, 0, -ratio, ratio, -1, 1, 3,
}
```

该代码填充一个投影矩阵 mProjectionMatrix，然后您可以在 onDrawFrame() 方法中与相机视图转换相结合，下一节将显示。

> 注意：将投影变换应用于绘图对象通常会导致非常空的显示。 一般来说，您还必须应用相机视图转换才能在屏幕上显示任何内容。

定义相机视图

通过在渲染器中添加一个相机视图转换作为绘图过程的一部分，完成转换绘制对象的过程。 在以下示例代码中，使用 Matrix.setLookAtM() 方法计算相机视图转换，然后与先前计算的投影矩阵组合。 然后将组合的变换矩阵传递到绘制的形状。

```
@Override
public void onDrawFrame(GL10 unused) {
    ...
    // Set the camera position (View matrix)
    Matrix.setLookAtM(mViewMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f, 1.0

    // Calculate the projection and view transformation
    Matrix.multiplyMM(mMVPMatrix, 0, mProjectionMatrix, 0, mViewMat

    // Draw shape
    mTriangle.draw(mMVPMatrix);
}
```

应用投影和相机转换

为了使用预览部分中显示的组合投影和相机视图转换矩阵，首先将矩阵变量添加到先前在三角形类中定义的顶点着色器：

```
public class Triangle {

    private final String vertexShaderCode =
        // This matrix member variable provides a hook to manipulat
        // the coordinates of the objects that use this vertex shad
        "uniform mat4 uMVPMatrix;" +
        "attribute vec4 vPosition;" +
        "void main() {" +
        // the matrix must be included as a modifier of gl_Position
        // Note that the uMVPMatrix factor *must be first* in order
        // for the matrix multiplication product to be correct.
        "  gl_Position = uMVPMatrix * vPosition;" +
        "}";

    // Use to access and set the view transformation
    private int mMVPMatrixHandle;

    ...
}
```

接下来，修改图形对象的draw（）方法以接受组合的转换矩阵并将其应用于形状：

```
public void draw(float[] mvpMatrix) { // pass in the calculated tra
    ...

    // get handle to shape's transformation matrix
    mMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram, "uMVPM

    // Pass the projection and view transformation to the shader
    GLES20.glUniformMatrix4fv(mMVPMatrixHandle, 1, false, mvpMatrix

    // Draw the triangle
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);

    // Disable vertex array
    GLES20.glDisableVertexAttribArray(mPositionHandle);
}
```

一旦你正确地计算并应用了投影和相机视图转换，你的图形对象就按照正确的比例绘制，应该是这样的：

图**1.**应用投影和相机视图绘制的三角形。

现在，您有一个应用程序以正确的比例显示您的形状，现在是添加运动到您的形状的时间。

> 翻译：JackWaiting
> 审核：@northJjL
> 原始文档：https://developer.android.com/training/graphics/opengl/projection.html#projection

添加移动

在屏幕上绘制图像只是 OpenGL 比较基础的特性，除此之外还可以使用 Android 图像框架相关的对象，包括 Canvas、Drawable 等对象。OpenGL ES 提供了三维空间的图形移动和转换，或其它独特的功能，去创建非凡的用户体验。

> 这一课，将进一步学习如何使用 OpenGL ES 对一个图形添加旋转动画。

旋转模型

在 OpenGL ES 2.0 中旋转绘图对象是相当简单的。在渲染器中创建一个新的转换矩阵（旋转矩阵），并将它合并到项目中，转换矩阵如下：

```
    private float[] mRotationMatrix = new float[16];
    public void onDrawFrame(GL10 gl) {
        float[] scratch = new float[16];

        ...

        // Create a rotation transformation for the triangle
        long time = SystemClock.uptimeMillis() % 4000L;
        float angle = 0.090f * ((int) time);
        Matrix.setRotateM(mRotationMatrix, 0, angle, 0, 0, -1.0f);

        // Combine the rotation matrix with the projection and came
        // Note that the mMVPMatrix factor *must be first* in orde
        // for the matrix multiplication product to be correct.
        Matrix.multiplyMM(scratch, 0, mMVPMatrix, 0, mRotationMatr

        // Draw triangle
        mTriangle.draw(scratch);
    }
```

如果做了如上操作后三角图形并没有旋转，请检查是否有添加 GLSurfaceView.RENDERMODE_WHEN_DIRTY 设置，在下一节将会介绍该设置。

设置连续渲染

如果你认真的跟随本课示例代码实现到了这里，确认一下是否把设置渲染模式为 *GLSurfaceView.RENDERMODE_WHEN_DIRTY* 这一行注释掉了，否则 OpenGL 对此图形仅仅有一次旋转，然后就等待 GLSurfaceView 容器的 requestRender()) 方法来触发再次执行：

```
    public MyGLSurfaceView(Context context) {
        ...
        // Render the view only when there is a change in the drawi
        // To allow the triangle to rotate automatically, this line
        //setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
    }
```

一般是将这个配置打开的，除非你所要操作的对象变化和用户交互无关。接下来这行代码不会注释，下一课将会再次使用。

> 翻译：@iOnesmile
> 审核：@misparking 原始文档：https://developer.android.google.cn/training/graphics/opengl/motion.html#rotate

响应触摸事件

让一个对象根据预设的程序进行移动，比如让三角形旋转，看起来很吸引眼球。但是怎么才能让用户和OpenGL ES 图形进行交互？关键就在于扩展 GLSurfaceView 的实现，重写 onTouchEvent()) 来监听处理用户触摸事件。

这节课讲解如何根据监听触摸事件来旋转 OpenGL ES 图形对象。

设置触摸监听器

为了让 OpenGL ES 应用响应触摸事件，需要实现在 GLSurfaceView 类中的 onTouchEvent()) 方法。下面简单的实现展示了怎样去监听 MotionEvent.ACTION_MOVE 事件并将之转换为形状旋转的角度。

```
private final float TOUCH_SCALE_FACTOR = 180.0f / 320;
private float mPreviousX;
private float mPreviousY;

@Override
public boolean onTouchEvent(MotionEvent e) {
    // MotionEvent reports input details from the touch screen
    // and other input controls. In this case, you are only
    // interested in events where the touch position changed.

    float x = e.getX();
    float y = e.getY();

    switch (e.getAction()) {
        case MotionEvent.ACTION_MOVE:

            float dx = x - mPreviousX;
            float dy = y - mPreviousY;

            // reverse direction of rotation above the mid-line
            if (y > getHeight() / 2) {
              dx = dx * -1 ;
            }
```

```
            // reverse direction of rotation to left of the mid-lin
            if (x < getWidth() / 2) {
              dy = dy * -1 ;
            }

            mRenderer.setAngle(
                    mRenderer.getAngle() +
                    ((dx + dy) * TOUCH_SCALE_FACTOR));
            requestRender();
    }

    mPreviousX = x;
    mPreviousY = y;
    return true;
}
```

需要注意的是，在计算一个旋转的角度后，都将调用一次 requestRender()) 方法来告诉渲染器需要进行绘制了。这是最高效的方法，因为如果没有发生旋转改变，是不需要重绘的。当然，当数据发生改变时需要使用 setRenderMode()) 方法，以保证渲染器在重绘时的效率不受影响，所以请确保没有注释这行代码：

```
public MyGLSurfaceView(Context context) {
    ...
    // Render the view only when there is a change in the drawing c
    setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
}
```

传递旋转角度

上面的代码还需要我们在渲染的过程中添加一个 public 成员变量，用来展示旋转角度。因为渲染器代码运行在一个独立线程中，而不是 UI 线程。所以需要声明这个公共变量为 *volatile*。下面是代码示例，包括了此变量的声明以及对应的set和get方法。

```
public class MyGLRenderer implements GLSurfaceView.Renderer {
    ...

    public volatile float mAngle;

    public float getAngle() {
        return mAngle;
    }

    public void setAngle(float angle) {
        mAngle = angle;
    }
}
```

旋转

触摸产生的旋转，添加以下代码 为了使触摸输入产生的旋转生效，需要注释掉创建图形旋转角度的代码并添加*mAngle* 变量，该变量为触摸输入所产生的角度：

```
public void onDrawFrame(GL10 gl) {
    ...
    float[] scratch = new float[16];

    // Create a rotation for the triangle
    // long time = SystemClock.uptimeMillis() % 4000L;
    // float angle = 0.090f * ((int) time);
    Matrix.setRotateM(mRotationMatrix, 0, mAngle, 0, 0, -1.0f);

    // Combine the rotation matrix with the projection and camera v
    // Note that the mMVPMatrix factor *must be first* in order
    // for the matrix multiplication product to be correct.
    Matrix.multiplyMM(scratch, 0, mMVPMatrix, 0, mRotationMatrix, 0

    // Draw triangle
    mTriangle.draw(scratch);
}
```

完成以上步骤之后，运行程序，然后可以通过手指滑动屏幕来改变三角形的角度。

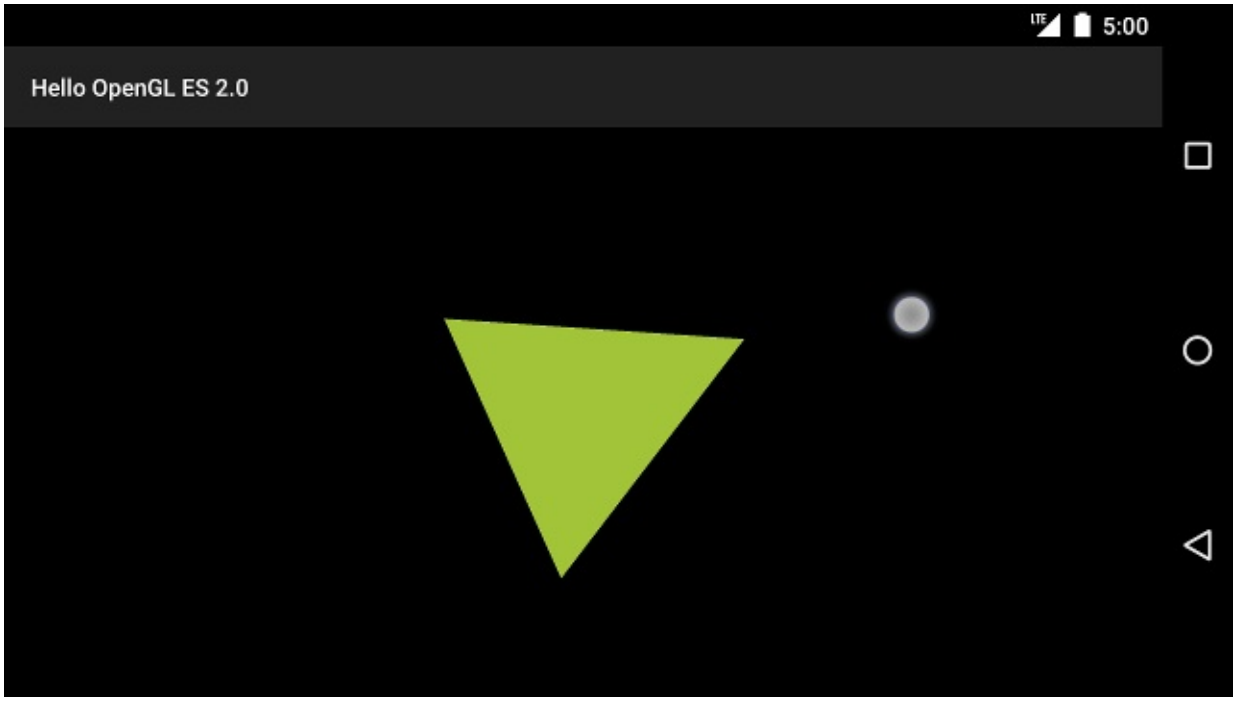


图 1. 三角形随着触摸屏幕的手指滑动而旋转(圆点为当前触摸位置).

> 翻译：@misparking
> 审核：@iOnesmile
> 原始文档：https://developer.android.google.cn/training/graphics/opengl/touch.html

过渡动画框架

应用程序的用户界面动画提供更多的不仅仅是视觉上的吸引力。动画明显的变化和提供的视觉提示，都帮助用户了解应用程序是如何工作的。

安卓提供了过渡动画框架 ，帮助你在一个视图层级和另一个视图层级之间完成动画变化。这框架适用于一个或者多个动画在层级结构上的所有视图，框架随它们变化而变化。

这个框架有跟随特性：

组-等级动画

适合一个或者多个动画效果在层级结构上的所有视图。

过渡-基础动画

运行动画基于开始和结束视图属性值之间的变化。

内置动画

包括普通效果的预定义动画，例如淡出和移动。

资源文件支持

从布局资源文件加载视图层结构和内置动画。

生命周期回调

定义回调以便更好地控制动画在层级的变化过程。

概述

这个例子在图1显示出动画是怎么通过提供视觉线索去帮助用户的。过渡动画框架随着应用程序从搜索条目页面到结果页面，它将淡出不再使用的视图，并淡入新的视图。

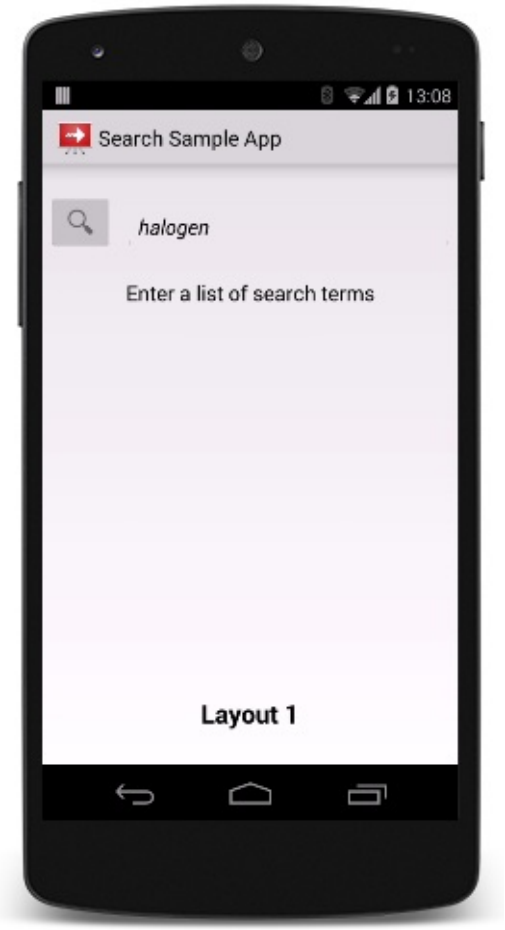


图**1.** 用户界面动画使用视觉线索

这一个使用过渡动画框架动画的例子，这个框架动画变化在两个层级结构的所有视图。一个层级结构能够更简单的选出视图和更复杂的包含一个详细视图树的视图组，在初始化开始视图层级和最后结束视图层级之间，这个框架赋予每个视图改变一次或者多次属性值。

在视图层级和动画，这个过渡动画框架工作方式是相同的，以储存视图层级状态的目的，更改这些层次结构来修改设备屏幕的外观，且通过存储和应用动画定义来激活更改。

这个图标图2阐明视图层级、框架类和动画之间的关系。

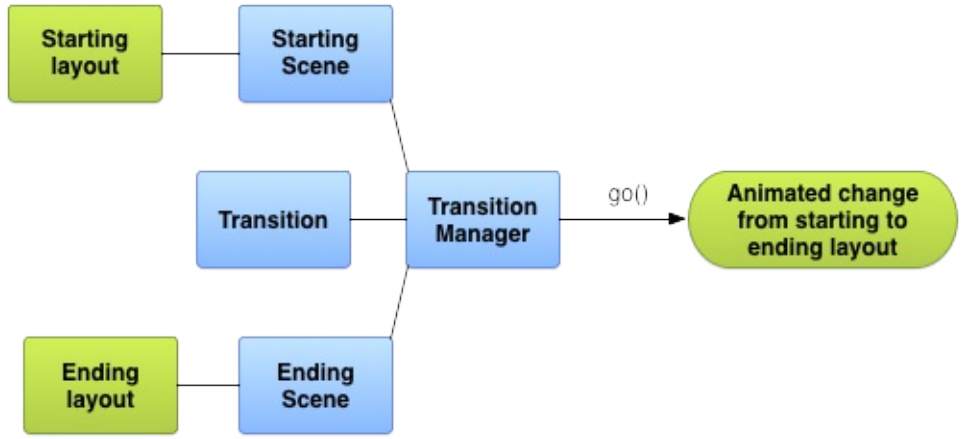


图**2.** 过渡动画框架的关系图

这个过渡动画框架提供抽象的场景、过渡和过渡管理，以下几节描述了这些内容。使用这个框架，你需要在应用程序的视图层级创建场景，以便两者之间进行更改。接下来，你在每个要使用的动画创建一个过渡。在两个视图层级之间开始，你使用一个过渡管理指定这个过渡去使用和结束场景，这个程序描述的细节在这堂课剩下的课程。

场景

场景储存了视图层级的状态，包括所有视图和它们的属性值。视图层级能够简化视图，可以复杂视图树和子布局。在场景中存储视图层次结构状态可以使你从另一场景过渡到该状态。这个框架提供的 场景 可去代表这个场景。

这个过渡动画框架让你通过代码在布局资源文件 和视图组 类创建场景，如果你动态生成一个视图层次结构，或者在运行时修改视图层次结构，那么在代码中创建一个场景是非常有用的。

在多半的情况中，如果你使用过渡，你不能明确的创建一个启动场景。使用框架提供的结束场景作为任何后续过渡的开始场景，如果你不应用过渡，框架会收集有关当前场景状态的视图信息。

在你做出场景改变的时候运行场景,能够同样定义它拥有的活动。举个例子，此功能对过渡到场景后的清理视图设置非常有用。

在添加视图层级和它的属性值，场景还将一个引用存储到视图层次结构的父级。影响场景的场景和动画的更改发生在场景根中。影响场景的场景和动画的更改发生在场景根中。此根视图称为场景根。

去学习怎么去创建场景，看 创建场景 。

过渡

在这个过渡动画框架，动画创建一系列的帧数，描述开始和结束场景之间的视图层次结构的变化。有关动画的信息存储在一个 过渡 对象中。要运行动画，你可以使用 过渡管理器 实例应用过渡。框架可以在两个不同的场景之间过渡，或者过渡到当前场景的不同状态。

该框架包括一组用于常用动画效果的内置过渡，如衰减和调整视图大小。那你还可以定义自己的自定义过渡，以使用动画框架中的API创建动画效果。因为过渡框架使你能够将动画效果组合在包含一组单独的内置或自定义过渡动画中。

这个过渡生命周期类似于活动生命周期，它表示框架在动画开始和完成之间监视的过渡状态。在重要的生命周期状态中，框架调用回调方法，可以在不同阶段的过渡中实现对用户界面的调整。

想学习更多关于过渡，看 Applying a Transiton 和 Create Custom Transitions .

局限性

- 动画适用于 SurfaceView 图形，可能不能正常显示出来。SurfaceView 的实例是从非UI线程更新，所以更新可能会失去与其他视图的动画同步。
- 一些特殊的过渡类型可能不会产生所需的动画效果，在适用于 TextureView 的时候。
- 扩展的 AdapterView 类，比如 Listview ,管理他们的孩子视图的方式来与过渡动画框架不符，如果你尝试基于 AdapterView 的动画视图,设备可能会挂掉。
- 如果你尝试在动画时测量大小 TextView ，在类完全测量大小后，文本将弹一个新位置。若要避免此问题，请不要调整包含文本视图的大小。

> 翻译：@northJjL
> 审核：@JackWaiting
> 原始文档：https://developer.android.google.cn/training/transitions/overview.html

创建场景

场景存储视图层次结构的状态，包括其所有视图及其属性值。转换框架可以在开始和结束场景之间运行动画。起始场景通常根据用户界面的当前状态自动确定。对于结束场景，该框架能够帮助你从布局资源文件或代码中的一组视图来创建场景。

本课程将向你展示如何在应用程序中创建场景以及如何定义场景动作。下一课将向你展示如何在两个场景之间进行转换。

> 注意：该框架可以在不使用场景的情况下对单个视图层次结构中的更改进行动画处理，如"应用无场景转换"中所述。然而，理解这一课是处理转换的关键。

从布局资源创建场景

你可以直接从布局资源文件创建一个 Scene 实例。当文件中的视图层次结构大部分是静态时，可以使用此技术。创建的场景表示视图层次结构的状态。如果你更改视图层次结构，则必须重新创建场景。因为框架从文件的整个视图层次结构中创建场景，所以你不能拿布局文件中的一部分去创建场景。

要从布局资源文件创建 Scene 实例，请从布局中 ViewGroup 实例检索，然后调用 Scene.getSceneForLayout（） 方法，其中包含场景以及视图层次结构的布局文件和资源ID。

定义场景布局

本节其余部分的代码段将向你展示如何使用具有相同根元素的场景创建两个不同的场景。这些代码段还表明你可以加载多个不相关的 Scene 对象，而不意味着它们彼此相关。

该示例由以下布局定义组成：

- 具有文本标签和子版面的活动的主要布局。
- 具有两个文本字段的第一个场景的相对布局。
- 具有相同两个文本字段却不同顺序的第二个场景的相对布局。
- 该示例被设计为，所有动画都发生在该活动的主布局的子布局中。主布局中的文本标签保持静态。

活动的主要布局定义如下：

res/layout/ activity_main.xml

```
LinearLayout xmlns:android="http://schemas.android.com/apk/res/andr
    android:id="@+id/master_layout">
    <TextView
        android:id="@+id/title"
        ...
        android:text="Title"/>
    <FrameLayout
        android:id="@+id/scene_root">
        <include layout="@layout/a_scene" />
    </FrameLayout>
</LinearLayout>
```

此布局定义包含场景的文本字段和子版式。 第一个场景的布局包含在主布局文件中。 因为框架只能将整个布局文件加载到场景中，所以应用程序将其显示为初始化用户界面的一部分，并将其加载到场景中。

第一个场景的布局定义如下：

res/layout/a_scene.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/a
    android:id="@+id/scene_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/text_view1
        android:text="Text Line 1" />
    <TextView
        android:id="@+id/text_view2
        android:text="Text Line 2" />
</RelativeLayout>
```

第二个场景的布局包含相同的两个文本字段（具有相同的ID）以不同的顺序放置，并定义如下：

res/layout/another_scene.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/a
    android:id="@+id/scene_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/text_view2
        android:text="Text Line 2" />
    <TextView
        android:id="@+id/text_view1
        android:text="Text Line 1" />
</RelativeLayout>
```

从布局生成场景

为两个相对布局创建定义后，你可以为它们每一个获取一个场景。 这使你可以稍后在两个UI配置之间进行转换。要获得场景，你需要参考场景根布局和布局资源ID。

以下代码片段显示了如何获取对场景根的引用，并从布局文件创建两个Scene对象：

```
Scene mAScene;
Scene mAnotherScene;

// Create the scene root for the scenes in this app
mSceneRoot = (ViewGroup) findViewById(R.id.scene_root);

// Create the scenes
mAScene = Scene.getSceneForLayout(mSceneRoot, R.layout.a_scene, thi
mAnotherScene =
    Scene.getSceneForLayout(mSceneRoot, R.layout.another_scene, thi
```

在应用程序中，现在有两个基于视图层次结构的 Scene 对象。 两个场景都使用 res / layout / activity_main.xml 中的 FrameLayout 元素定义的场景根布局。

在你的代码中创建一个场景

你也可以在 ViewGroup 对象的代码中创建一个 Scene 实例。 在代码中直接修改视图层次结构或动态生成视图层次结构时，请使用此技术。

要从代码中的视图层次结构创建场景，请使用 Scene（sceneRoot，viewHierarchy）构造函数。 当你已经填充了布局，调用此构造函数等同于在调用 Scene.getSceneForLayout（） 方法。

以下代码片段演示了如何从代码中的场景根元素和场景的视图层次结构创建一个 Scene 实例：

```
Scene mScene;

// Obtain the scene root element
mSceneRoot = (ViewGroup) mSomeLayoutElement;

// Obtain the view hierarchy to add as a child of
// the scene root when this scene is entered
mViewHierarchy = (ViewGroup) someOtherLayoutElement;

// Create a scene
mScene = new Scene(mSceneRoot, mViewHierarchy);
```

创建场景动作

该框架使你能够定义系统在进入或退出场景时运行自定义场景动作。在许多情况下，定义自定义场景动作是不必要的，因为框架会自动动画场景之间的变化。

场景动作对于处理这些情况很有用：

- 动画不在同一层次结构中的视图。你可以使用退出和输入场景动作作为起始和结束场景的动画。
- 动画视图，转换框架不能自动动画，例如 ListView 对象。有关详细信息，请参阅限制。

要提供自定义场景动作，请将你的操作定义为可运行对象，并将其传递给 Scene.setExitAction（）或 Scene.setExitAction（）方法。在运行转换动画之后，框架在运行转换动画之前调用 Scene.setExitAction（）方法，然后在结束场景上运行 Scene.setExitAction（）方法。

> 注意：不要在开始和结束场景视图之间传递数据时使用场景动作。有关更多信息，请参阅定义转换生命周期回调。
>
> 翻译：JackWaiting
> 审核：@northJjL
> 原始文档：https://developer.android.com/training/transitions/scenes.html

引用过渡动画

在过渡框架中，动画创建一系列框架，描绘起始和结束场景中的视图层次结构之间的变化。框架将这些动画描绘成包含动画信息的过渡对象。 要运行动画，你可以向过渡管理器提供使用过渡和结束场景。

本课程教你如何使用内置的过渡框架来在两个场景之间运行一个动画来移动，调整大小和淡化视图。 下一课向你展示如何自定义过渡动画。

创建过渡动画

在上一课中，学习了如何创建表示不同视图层次结构状态的场景。 一旦定义了要在其间切换的起始场景和结束场景，则需要创建一个定义动画的 Transition 对象。 该框架使你能够在资源文件中指定内置的过渡，并在代码中加载它，或直接在代码中创建内置过渡类的实例。

表 1，内置过渡类型。

Class	Tag	Attributes	Effect
AutoTransition	\	-	Default transition. Fade out, move and resize, and fade in views, in that order.
Fade	\	android:fadingMode="[*fade_in* | *fade_out* | *nfade_in_out*]"	*fade_in* fades in views *fade_out* fades out views *fade_in_out* (default) does a fade_out followed by a fade_in.
ChangeBounds	\	-	Moves and resizes views.

从资源文件创建过渡实例

这种技术使你能够修改过渡效果的定义，而无需更改 Activity 中的代码。 这种技术也可用于将复杂的过渡效果定义与应用程序代码分开，如 “Specify Multiple Transitions” 所示。

要在资源文件中指定内置的过渡，请按照下列步骤操作：

1. 将 res/transition/ 目录添加到项目中。
2. 在此目录中创建一个新的 XML 资源文件。
3. 添加一个内置过渡效果的 XML 节点。

例如，以下资源文件指定了 Fade 过渡：

res/transition/fade_transition.xml

```
<fade xmlns:android="http://schemas.android.com/apk/res/android" />
```

以下代码片段显示了如何从资源文件中充实活动中的Transition实例：

```
Transition mFadeTransition =
        TransitionInflater.from(this).
        inflateTransition(R.transition.fade_transition);
```

在代码中创建一个过渡实例

如果你在代码中的修改用户界面，并且创建简单的内置过渡实例（很少或没有参数），则此技术对于动态创建过渡对象很有用。

要创建内置过渡的实例，请调用 Transition 类的子类中的一个公共构造函数。 例如，以下代码片段创建了一个 Fade 过渡效果的实例：

```
Transition mFadeTransition = new Fade();
```

引用过渡动画

响应不同视图层次结构之间的事件触发（例如用户操作），通常可以通过引用过渡效果来实现。

通常使用过渡效果的场景是在不同的视图层次结构之间响应事件操作 你通常应用过渡以在响应于事件（例如用户操作）的不同视图层次结构之间进行更改。 例如，考虑一个搜索应用程序：当用户输入搜索字词并点击搜索按钮时，应用程序将界面展

示更改为搜索结果的布局场景，同时搜索按钮渐变淡出，且搜索结果渐变淡入的过渡效果。

要对 Activity 中的某些事件应用过渡效果并产生场景变化时，请调用 TransitionManager.go()) 静态方法，并使用结尾场景和过渡实例来使用动画，如以下代码片段所示：

```
TransitionManager.go(mEndingScene, mFadeTransition);
```

在运行由过渡实例指定的动画的情况下，框架将从场景根目录中的视图层次结构更改为结束场景中的视图层次结构。 起始场景是上一次过渡动效的结束场景。 如果没有以前的过渡动效，则从用户界面的当前状态自动确定起始场景。

如果不指定过渡实例，过渡管理器可以应用自动过渡，这对大多数情况都是合理的。 有关更多信息，请参阅 TransitionManager 类的 API 参考。

选择具体目标视图

默认情况下，该框架将过渡应用于起始和结束场景中的所有视图。 在某些情况下，你可能只想将动画应用于场景中的一部分视图。 例如，框架不支持 ListView 对象的动画变化，因此就不用再尝试在过渡动画期间为它们设置动画。该框架使你能够选择需要进行动画的特定视图。

过渡动画的每个视图称为目标。你只能选择与场景相关联的视图层次结构的一部分做为目标。

要从目标列表中删除一个或多个视图，请在开始过渡之前调用 removeTarget() 方法。 仅将你指定的视图添加到目标列表，请调用 addTarget()) 方法。 有关更多信息，请参阅 Transition 类的API参考。

指定多个过渡

如果想达到比较好的动画效果，应该将各个场景之间发生的动画变化类型进行匹配。例如，如果要删除某些视图并在场景之间添加其他视图，则动画中的淡出/淡入可以表示某些视图不再可用。如果你将视图移动到屏幕上的不同点，更好的选择是为其设置动画，以便用户注意到视图的新位置。

你不必仅仅选择一个动画，因为过渡框架使你能够将动画效果组合在包含一组单独的内置或自定义过渡动画中。

要从XML中的过渡集合中定义过渡集，请在 *res / transitions /* 目录中创建资源文件，并列出 *transitionSet* 元素下的过渡效果。例如，以下代码段显示了如何指定与 AutoTransition 类具有相同行为的过渡集：

```
<transitionSet xmlns:android="http://schemas.android.com/apk/res/ar
    android:transitionOrdering="sequential">
    <fade android:fadingMode="fade_out" />
    <changeBounds />
    <fade android:fadingMode="fade_in" />
</transitionSet>
```

要将代码中的 TransitionSet 对象充满过渡集，请调用活动中的 TransitionInflater.from（）方法。 TransitionSet 类从 Transition 类扩展，因此你可以像过渡实例一样使用过渡管理器。

应用没有场景的过渡

更改视图层次结构不是修改用户界面的唯一方法。你还可以通过在当前层次结构中添加，修改和删除子视图进行更改。例如，你可以仅使用单个布局实现搜索交互。从显示搜索条目栏和搜索图标的布局开始。要更改用户界面以显示结果，当用户通过调用 ViewGroup.removeView（）)方法单击它时，删除搜索按钮，并通过调用 ViewGroup.addView（）方法添加搜索结果。

如果替代方案有两个几乎相同的层次结构，则可能需要使用这种方法。不必在用户界面中创建和维护两个单独的布局文件，而是可以在代码中修改一个包含视图层次结构的布局文件。

如果你以此方式在当前视图层次结构中进行更改，则无需创建场景。相反，你可以使用延迟过渡创建并应用视图层次结构的两个状态之间的过渡。过渡框架的此功能以当前视图层次结构状态开始，记录对其视图所做的更改，并在系统重新绘制用户界面时应用一个动画变化的过渡。

要在单个视图层次结构中创建延迟过渡，请按照下列步骤操作：

1. 当触发过渡的事件发生时，调用 TransitionManager.beginDelayedTransition（）方法，提供要更改的所有视图的父视图以及要使用的过渡。框架存储子视图及其属性值的当前状态。
2. 根据用例的要求更改子视图。框架记录你对子视图及其属性所做的更改。
3. 当系统根据你的更改重新绘制用户界面时，框架将动画化原始状态和新状态之间的变化。

以下示例显示了如何使用延迟过渡将文本视图添加到视图层次结构中。第一个片段显示布局定义文件：

res/layout/activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/a
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <EditText
        android:id="@+id/inputText"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    ...
</RelativeLayout>
```

下一个代码段显示了添加文本视图的代码：

MainActivity.java

```
private TextView mLabelText;
private Fade mFade;
private ViewGroup mRootView;
...

// Load the layout
this.setContentView(R.layout.activity_main);
...

// Create a new TextView and set some View properties
mLabelText = new TextView();
mLabelText.setText("Label").setId("1");

// Get the root view and create a transition
mRootView = (ViewGroup) findViewById(R.id.mainLayout);
mFade = new Fade(IN);

// Start recording changes to the view hierarchy
TransitionManager.beginDelayedTransition(mRootView, mFade);

// Add the new TextView to the view hierarchy
mRootView.addView(mLabelText);

// When the system redraws the screen to show this update,
// the framework will animate the addition as a fade in
```

定义过渡动画生命周期回调

过渡动画生命周期类似于 activity 生命周期。它表示框架在调用 TransitionManager.go（）方法和完成动画之间的时间期间监视的过渡状态。在重要的生命周期状态下，框架调用由 TransitionListener 接口定义的回调。

过渡生命周期回调是有用的，例如，在场景更改期间将视图属性值从起始视图层次结构复制到结束视图层次结构。你不能简单地将值从起始视图复制到结束视图层次结构中的视图，因为在过渡动画完成之前结束视图层次结构不能增大。相反，你需要将值存储在变量中，然后在框架完成过渡后将其复制到结束视图层次结构中。要在过渡完成时收到通知，可以在你的 Activity 中实现 TransitionListener.onTransitionEnd（）方法。

有关更多信息，请参阅 TransitionListener 类的 API 参考。

> 翻译：@iOnesmile 审核：@misparking 原始文档：https://developer.android.google.cn/training/transitions/transitions.html#Callbacks

创建自定义过渡动画

用自定义的形式创建的过渡动画是其它任何内置的过渡类都不能直接实现的。比如你可以定义一个改变文字的前景颜色和输入字段的区域为灰色，表示在此界面中这块区域不可用。这种改变的效果可以帮助用户清楚哪些区域不可用。

自定义的过渡就像内置的过渡类型一样，将动画应用于起始场景和结束场景的子视图中。但是，与内置的过渡类型不同的是：自定义过渡动画必须提供可以获取属性值并生成动画的代码。你也可以为某些控件的子集定义动画。

这一节课将帮助你通过获取属性值和生成动画去创建自定义过渡动画。

扩展过渡类

创建一个自定义过渡动画，需要先在项目工程中添加一个类，用来扩展 Transition 类以及重写它的一些方法。详见下面的代码段：

```
public class CustomTransition extends Transition {

    @Override
    public void captureStartValues(TransitionValues values) {}

    @Override
    public void captureEndValues(TransitionValues values) {}

    @Override
    public Animator createAnimator(ViewGroup sceneRoot,
                                   TransitionValues startValues,
                                   TransitionValues endValues) {}
}
```

下一节将会详细讲解如何重写这些方法。

获取控件的属性值

过渡动画使用 Property Animation 描述的属性动画系统。属性动画会在指定的时间内更改起始值和结束值之间的控件属性，所以框架需要同时具有开始和结束时的属性值去构建这个动画。

然后，属性动画通常仅仅需要所有的控件属性值的一小部分。比如，颜色动画需要颜色属性值，而移动动画需要位置属性值。 由于这里动画所需要的属性值是针对于过渡动画的，所以这个框架不再提供每个属性值。而是提供一些回调方法以供过渡动画去获取一些需要的属性值并传递给框架进行保存。

获取开始时的值

通过开始时的框架控件值，实现 capturestartvalues（transitionvalues）) 方法。框架中每一个在开始场景中控件都会调用此方法，方法的参数是一个包含对控件引用的 transitionvalues 对象，可以使用 Map 实例保存你需要的控件值。在这个实现里，检索这些属性值并使用 Map 保存，再反馈到框架中。

为了确保一个属性值的键值不与其他 TransitionValues 键值冲突，需要使用下面的命名方式：

```
package_name:transition_name:property_name
```

以下片段显示了 captureStartValues（）方法的实现：

```
public class CustomTransition extends Transition {

    // Define a key for storing a property value in
    // TransitionValues.values with the syntax
    // package_name:transition_class:property_name to avoid collisi
    private static final String PROPNAME_BACKGROUND =
            "com.example.android.customtransition:CustomTransition:

    @Override
    public void captureStartValues(TransitionValues transitionValue
        // Call the convenience method captureValues
        captureValues(transitionValues);
    }


    // For the view in transitionValues.view, get the values you
    // want and put them in transitionValues.values
    private void captureValues(TransitionValues transitionValues) {
        // Get a reference to the view
        View view = transitionValues.view;
        // Store its background property in the values map
        transitionValues.values.put(PROPNAME_BACKGROUND, view.getBa
    }
    ...
}
```

获取结束时的值

针对每一个目标控件，框架需要在结束场景时调用一次 captureEndValues(TransitionValues)) 方法。在任意情况下，captureendvalues()) 的工作原理与 capturestartvalues()) 都是一致的。

下面的代码段展示了 captureendvalues()) 方法的实现：

```
@Override
public void captureEndValues(TransitionValues transitionValues) {
    captureValues(transitionValues);
}
```

在这个例子中，capturestartvalues()) 和 captureendvalues()) 方法都调用了 *capturevalues()* 去检索和保存数据。每个控件属性都有相同的 *capturevalues()* 检索，但是有不同的开始场景和结束场景。框架为控件的开始和结束时的状态维护了独立的映射。

创建自定义动画对象

对一个控件进行过渡在它的起始场景和结束场景之间，通过动画的形式对控件从开始场景到结束场景之间进行改变，需要使用一个动画对象并重写 createAnimator()) 方法。当框架调用此方法时，将会把你获取到的开始时的值和结束时的值传递到根控件和 TransitionValues 对象中。

这个系统框架调用 createAnimator()) 的次数取决于控件在开始和结束时的场景过程中的变化次数。例如，通过自定义过渡动画实现淡入淡出动画。如果在开始时的场景已知有五个目标动画效果，在结束的场景中移除了其中两个，同时另外三个新的动画效果会添加在新的目标任务。也就是说框架会调用 createAnimator()) 六次：三次淡出和淡入；二次淡出以及一次在新的目标结束场景中的淡入效果。

在开始和结束的场景中，目标控件是一直存在的，框架提供了一个包含开始值参数和结束值参数的 TransitionValues 对象。针对目标控件只存在于开始场景或者只存在于结束场景的情况，框架也提供了含有对应参数的 TransitionValues 对象。 当你通过实现 createAnimator(ViewGroup, TransitionValues, TransitionValues)) 方法去创建自定义过渡动画时，需要使用获取到的控件属性值创建动画管理对象并把属性值返回给框架。有一个简单的样例，请参阅 CustomTransition 示例中的 ChangeColor 类。有关属性动画的更多信息，请参见 Property Animation。

实现 createanimator（ViewGroup，TransitionValues，TransitionValues）)

使用自定义动画

自定义过渡动画和内置过渡类的实现原理类似。你可以参考 Applying a Transition 描述的那样，去使用过渡动画管理器应用自定义过渡动画。

> 翻译：@misparking
> 审核：@iOnesmile
> 原始文档：https://developer.android.google.cn/training/transitions/custom-transitions.html

交叉渐变的两个视图

交叉渐变动画（也称之为溶解）指的是一个UI控件渐渐淡出的同时，另一个控件也淡化。这个动画在你切换应用中的内容或者视图的时候非常有用。它是非常微妙且短暂的提供了一个页面到下个页面的流畅过渡。在你不使用它的时候，过渡提供的感觉生硬且仓促。

这是一个从进度指示器到一些文本内容交叉渐变的例子 。

交叉渐变动画

点击设备屏幕来重新播放电影

如果你想跳到前面，并查看完整的工作示例，下载并运行示例应用，然后选择交叉渐变的例子，请参阅下面代码实现的文件:

- `src/CrossfadeActivity.java`
- `layout/activity_crossfade.xml`
- `menu/activity_crossfade.xml`

创建视图

创建两个你想交叉渐变的视图，下面是创建进度指示器和滑动的文本视图的例子。

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/andr
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ScrollView xmlns:android="http://schemas.android.com/apk/res/a
        android:id="@+id/content"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView style="?android:textAppearanceMedium"
            android:lineSpacingMultiplier="1.2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/lorem_ipsum"
            android:padding="16dp" />

    </ScrollView>

    <ProgressBar android:id="@+id/loading_spinner"
        style="?android:progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />

</FrameLayout>
```

设置动画

如何设置动画：

1.创建一个你想交叉渐变视图的成员变量，稍后在动画修改视图时，你需要这些引用。

2.将要淡入的视图，设置它的可见度为GONE,这样可以阻止视图占用布局的空间且布局计算中可忽略它，加快处理。

3.用一个成员变量缓存系统属性 config_shortAnimTime,这个属性在动画中定义了短暂持续时间的标准，这种持续时间频繁出现在理想的精巧动画和动画。如果你想使用它们， config_longAnimTime和 config_mediumAnimTime 同样有效。

下面的示例是之前使用那段代码的布局作为Activity文本视图：

```
public class CrossfadeActivity extends Activity {

private View mContentView;
private View mLoadingView;
private int mShortAnimationDuration;

...

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_crossfade);

    mContentView = findViewById(R.id.content);
    mLoadingView = findViewById(R.id.loading_spinner);

    // Initially hide the content view.
    mContentView.setVisibility(View.GONE);

    // Retrieve and cache the system's default "short" animation ti
    mShortAnimationDuration = getResources().getInteger(
            android.R.integer.config_shortAnimTime);
}
```

交叉渐变视图

现在视图正确的设置好了，通过下面的步骤来让他们逐渐消失吧:

1.正要淡入的视图，设置透明值为0，并且可见度为VISIBILE,（记得初始化它的时候设置为 GONE)这是让视图看得见但却完全透明。

2.正要淡入的视图，动画的透明值从0到1，同时，正要淡出的视图，动画的透明值从1到0。

3.使用 Animator.AnimatorListener() 中的 onAnimationEnd())，设置淡出的视图可见度为GONE，阻止视图占用布局的空间且布局计算中可忽略它，加快处理。

下面展示了如何执行操作的示例：

```
private View mContentView;
private View mLoadingView;
private int mShortAnimationDuration;

...

private void crossfade() {

    // Set the content view to 0% opacity but visible, so that it i
    // (but fully transparent) during the animation.
    mContentView.setAlpha(0f);
    mContentView.setVisibility(View.VISIBLE);

    // Animate the content view to 100% opacity, and clear any anim
    // listener set on the view.
    mContentView.animate()
            .alpha(1f)
            .setDuration(mShortAnimationDuration)
            .setListener(null);

    // Animate the loading view to 0% opacity. After the animation
    // set its visibility to GONE as an optimization step (it won't
    // participate in layout passes, etc.)
    mLoadingView.animate()
            .alpha(0f)
            .setDuration(mShortAnimationDuration)
            .setListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    mLoadingView.setVisibility(View.GONE);
                }
            });
}
```

翻译：@northJjL

原始文档：https://developer.android.google.cn/training/animation/crossfade.html

如何使用 ViewPager 进行屏幕滑动

屏幕滑动时一个屏幕与另一个屏幕之间的过滤，这种UI的使用在我们的APP中也是非常常见的。本课程将向你展示如何使用支持库提供的 ViewPager 进行屏幕滑动。同时ViewPagers也可以自动添加动画。 下面这个就是从一个屏幕滑动到另一个屏幕：

如果你想跳过并查看完整的工作示例，请下载并运行示例应用程序并选择滑动示例。 有关代码实现，请参阅以下文件：

- src/ScreenSlidePageFragment.java
- src/ScreenSlideActivity.java
- layout/activity_screen_slide.xml
- layout/fragment_screen_slide_page.xml

创建Views

创建一个布局文件，稍后将用于 Fragment 的内容。 以下示例包含显示一些文本视图：

```
<!-- fragment_screen_slide_page.xml -->
<ScrollView xmlns:android="http://schemas.android.com/apk/res/andro
    android:id="@+id/content"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView style="?android:textAppearanceMedium"
        android:padding="16dp"
        android:lineSpacingMultiplier="1.2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/lorem_ipsum" />
</ScrollView>
```

还为该 Fragment 的内容定义一个字符串。

创建Fragment

创建一个 Fragment 类，返回刚刚在 onCreateView（） 方法中创建的布局。 然后，你可以在需要向用户显示的新页面时在父 Activity 中创建此 Fragment 的实例：

```
import android.support.v4.app.Fragment;
...
public class ScreenSlidePageFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup cor
            Bundle savedInstanceState) {
        ViewGroup rootView = (ViewGroup) inflater.inflate(
                R.layout.fragment_screen_slide_page, container, fal

        return rootView;
    }
}
```

添加一个ViewPager

ViewPagers 具有内置的滑动手势，可以通过界面进行转换，并且默认情况下显示滑动动画，因此你不需要创建任何内容。 ViewPager使用PagerAdapters作为要显示的适配器，因此 PagerAdapter 将使用你之前创建的 Fragment 类。

首先，创建一个包含ViewPager的布局：

```
<!-- activity_screen_slide.xml -->
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

创建一个执行以下操作的 Activity：

- 使用 ViewPager 将内容视图设置为布局。
- 创建一个扩展 FragmentStatePagerAdapter 的抽象类，并实现 getItem（）方法，将 ScreenSlidePageFragment 的实例作为 content。同时适配器还要求你实现getCount（）方法，该方法返回适配器将创建的页面数量（示例中为五个）。
- 将 PagerAdapter 设置到 ViewPager。
- 通过在虚拟堆栈中向后移动来处理设备的后退按钮。如果用户已经在第一页上，请返回到 Activity 堆栈。

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
...
public class ScreenSlidePagerActivity extends FragmentActivit
    /**
     * The number of pages (wizard steps) to show in this dem
     */
    private static final int NUM_PAGES = 5;

    /**
     * The pager widget, which handles animation and allows s
     * and next wizard steps.
     */
    private ViewPager mPager;

    /**
     * The pager adapter, which provides the pages to the vie
     */
    private PagerAdapter mPagerAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_screen_slide);

        // Instantiate a ViewPager and a PagerAdapter.
        mPager = (ViewPager) findViewById(R.id.pager);
        mPagerAdapter = new ScreenSlidePagerAdapter(getSuppor
```

```
            mPager.setAdapter(mPagerAdapter);
        }

        @Override
        public void onBackPressed() {
            if (mPager.getCurrentItem() == 0) {
                // If the user is currently looking at the first
                // Back button. This calls finish() on this activ
                super.onBackPressed();
            } else {
                // Otherwise, select the previous step.
                mPager.setCurrentItem(mPager.getCurrentItem() - 1
            }
        }

        /**
         * A simple pager adapter that represents 5 ScreenSlidePa
         * sequence.
         */
        private class ScreenSlidePagerAdapter extends FragmentSta
            public ScreenSlidePagerAdapter(FragmentManager fm) {
                super(fm);
            }

            @Override
            public Fragment getItem(int position) {
                return new ScreenSlidePageFragment();
            }

            @Override
            public int getCount() {
                return NUM_PAGES;
            }
        }
    }
```

使用PageTransformer自定义动画

要从默认滑动动画转换成显示不同的动画，请调用 ViewPager.PageTransformer 并将其提供给适配器。该界面暴露了一种方法 transformPage（）。每个可见页面（通常只有一个可见页面）和显示在屏幕上的两个界面调用此方法。例如，如果第三页是可见的，并且用户拖动到第四页，则在手势的每个步骤都会调用 transformPage（） 来调用页面2，3和4。

在实现 transformPage（） 时，你可以通过从页面的位置参数 transformPage（）方法获取的位置确定哪些页面需要进行变换来创建自定义的滑动动画。

位置参数指示给定页面相对于屏幕中心的位置。它是一种动态属性，随着用户滚动页面而改变。当页面填满屏幕时，其位置值为0.当页面刚刚从屏幕右侧绘制时，其位置值为1.如果用户在第一页和第二页之间滚动一半，则第一页的位置为-0.5和第二页的位置为0.5。根据屏幕上页面的位置，你可以通过使用setAlpha（），setTranslationX（） 或 setScaleY（） 等方法设置页面属性来创建自定义滑动动画。

当你实现一个 PageTransformer 时，请在实现中调用 setPageTransformer（） 来应用自定义动画。例如，如果你有一个名为 ZoomOutPageTransformer 的 PageTransformer，你可以设置自定义动画，如下所示：

```
ViewPager mPager = (ViewPager) findViewById(R.id.pager);
...
mPager.setPageTransformer(true, new ZoomOutPageTransformer());
```

有关PageTransformer的示例和视频，请参阅 Zoom-out page transformer 和 Depth page transformer 部分。

Zoom-out page transformer

当在相邻页面之间滚动时，此页面 transformer 收缩并淡化页面。 随着页面越来越接近中心，它逐渐恢复到正常的大小并且渐渐衰

```
public class ZoomOutPageTransformer implements ViewPager.PageTransf
    private static final float MIN_SCALE = 0.85f;
    private static final float MIN_ALPHA = 0.5f;

    public void transformPage(View view, float position) {
        int pageWidth = view.getWidth();
        int pageHeight = view.getHeight();

        if (position < -1) { // [-Infinity,-1)
            // This page is way off-screen to the left.
            view.setAlpha(0);

        } else if (position <= 1) { // [-1,1]
            // Modify the default slide transition to shrink the pa
            float scaleFactor = Math.max(MIN_SCALE, 1 - Math.abs(po
            float vertMargin = pageHeight * (1 - scaleFactor) / 2;
            float horzMargin = pageWidth * (1 - scaleFactor) / 2;
            if (position < 0) {
                view.setTranslationX(horzMargin - vertMargin / 2);
            } else {
                view.setTranslationX(-horzMargin + vertMargin / 2);
            }

            // Scale the page down (between MIN_SCALE and 1)
            view.setScaleX(scaleFactor);
            view.setScaleY(scaleFactor);

            // Fade the page relative to its size.
            view.setAlpha(MIN_ALPHA +
                    (scaleFactor - MIN_SCALE) /
                    (1 - MIN_SCALE) * (1 - MIN_ALPHA));

        } else { // (1,+Infinity]
            // This page is way off-screen to the right.
            view.setAlpha(0);
        }
    }
}
```

Depth page transformer

该页面 transformer 使用默认的滑动动画来滑动左侧的页面，同时使用“深度”动画将页面滑动到右侧。 这个深度动画使页面淡出，并将其线性缩放。

> 注意：在深度动画过程中，默认动画（滑动动画）仍然发生，因此你必须使用 -X 旋转来抵消滑动动画。 例如：

```
view.setTranslationX（-1 * view.getWidth（）* position）;
```

以下示例显示了如何抵消默认滑动动画：

```
public class DepthPageTransformer implements ViewPager.PageTransfor
    private static final float MIN_SCALE = 0.75f;

    public void transformPage(View view, float position) {
        int pageWidth = view.getWidth();

        if (position < -1) { // [-Infinity,-1)
            // This page is way off-screen to the left.
            view.setAlpha(0);

        } else if (position <= 0) { // [-1,0]
            // Use the default slide transition when moving to the
            view.setAlpha(1);
            view.setTranslationX(0);
            view.setScaleX(1);
            view.setScaleY(1);

        } else if (position <= 1) { // (0,1]
            // Fade the page out.
            view.setAlpha(1 - position);

            // Counteract the default slide transition
            view.setTranslationX(pageWidth * -position);

            // Scale the page down (between MIN_SCALE and 1)
            float scaleFactor = MIN_SCALE
                    + (1 - MIN_SCALE) * (1 - Math.abs(position));
            view.setScaleX(scaleFactor);
            view.setScaleY(scaleFactor);

        } else { // (1,+Infinity]
            // This page is way off-screen to the right.
            view.setAlpha(0);
        }
    }
}
```

卡片翻转动画

这一课讲述如何自定义卡片翻转动画。卡片翻转在视图之间，展示模拟的翻页动画。

> 这里是一个卡片翻转

卡片翻转动画

> 点击设备屏幕重播视频

如果你想抢先看到所有动画示例，下载 并运行示例应用，选择 Card Flip 示例。见如下文件的代码实现：

- src/CardFlipActivity.java
- animator/card_flip_right_in.xml
- animator/card_flip_right_out.xml
- animator/card_flip_left_in.xml
- animator/card_flip_left_out.xml
- layout/fragment_card_back.xml
- layout/fragment_card_front.xml

创建动画对象

创建卡片翻转动画。在卡片的正面需要两个动画，动画向左出和从左进。同样在卡片的反面也需要两个动画，动画从右进和向右出。

card_flip_left_in.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Before rotating, immediately set the alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:duration="0" />

    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="-180"
        android:valueTo="0"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_dece
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the a
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

card_flip_left_out.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="0"
        android:valueTo="180"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_dece
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the a
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

card_flip_right_in.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Before rotating, immediately set the alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:duration="0" />

    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="180"
        android:valueTo="0"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_dece
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the a
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

card_flip_right_out.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="0"
        android:valueTo="-180"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_dece
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the a
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

创建视图

卡片的每一面是单独的布局，包含着你想显示的任何内容。比如两屏文字，两张图片，或者任何视图的组合都可进行翻转。在即将显示的动画中，Fragment 需要使用两个布局。下面的布局是卡片的一面显示的文本：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/and
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#a6c"
    android:padding="16dp"
    android:gravity="bottom">

    <TextView android:id="@android:id/text1"
        style="?android:textAppearanceLarge"
        android:textStyle="bold"
        android:textColor="#fff"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/card_back_title" />

    <TextView style="?android:textAppearanceSmall"
        android:textAllCaps="true"
        android:textColor="#80ffffff"
        android:textStyle="bold"
        android:lineSpacingMultiplier="1.2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/card_back_description" />

</LinearLayout>
```

卡片的另外一面显示的 ImageView：

```
<ImageView xmlns:android="http://schemas.android.com/apk/res/androi
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:src="@drawable/image1"
    android:scaleType="centerCrop"
    android:contentDescription="@string/description_image_1" />
```

创建 Fragment

创建卡片正面和反面的 Fragment 类。在每个 Fragment 类的 onCreateView()) 方法中返回预先创建的布局。你可以在父类的 Activity 中创建要显示的卡片的 Fragment 实例。如下示例在父 Activity 内嵌的 Fragment 类：

```
public class CardFlipActivity extends Activity {
    ...
    /**
     * A fragment representing the front of the card.
     */
    public class CardFrontFragment extends Fragment {
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup
                Bundle savedInstanceState) {
            return inflater.inflate(R.layout.fragment_card_front, c
        }
    }

    /**
     * A fragment representing the back of the card.
     */
    public class CardBackFragment extends Fragment {
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup
                Bundle savedInstanceState) {
            return inflater.inflate(R.layout.fragment_card_back, co
        }
    }
}
```

卡片翻转动画

现在，你将需要在父 Activity 中显示 Fragment。为此，第一步先创建 Activity 的布局。下面的示例创建了 FrameLayout，可以在运行时添加 Fragment：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/andr
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

在 Activity 代码中，设置刚才创建的布局。在 Activity 创建的时候设置一个默认 Fragment，通常是个不错的方式，因此下面示例的 Activity 展示了如何默认显示卡片正面：

```
public class CardFlipActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_activity_card_flip);

        if (savedInstanceState == null) {
            getFragmentManager()
                    .beginTransaction()
                    .add(R.id.container, new CardFrontFragment())
                    .commit();
        }
    }
    ...
}
```

到目前为止卡片已经显示，在适当的时候通过动画翻转到卡片反面。创建方法显示卡片的另外一面，步骤如下：

- 给 Fragment 事务设置之前创建好的自定义动画。
- 用新的 Fragment 替换正在显示的 Fragment，伴随着已经创建的自定义动画。
- 添加准备显示的 Fragment 到返回栈，那么当用户按返回键时，卡片会翻回去。

```
private void flipCard() {
```

```
    if (mShowingBack) {
        getFragmentManager().popBackStack();
        return;
    }

    // Flip to the back.

    mShowingBack = true;

    // Create and commit a new fragment transaction that adds the f
    // the back of the card, uses custom animations, and is part of
    // manager's back stack.

    getFragmentManager()
            .beginTransaction()

            // Replace the default fragment animations with animato
            // representing rotations when switching to the back of
            // well as animator resources representing rotations wh
            // back to the front (e.g. when the system Back button
            .setCustomAnimations(
                    R.animator.card_flip_right_in,
                    R.animator.card_flip_right_out,
                    R.animator.card_flip_left_in,
                    R.animator.card_flip_left_out)

            // Replace any fragments currently in the container vie
            // fragment representing the next page (indicated by th
            // just-incremented currentPage variable).
            .replace(R.id.container, new CardBackFragment())

            // Add this transaction to the back stack, allowing use
            // Back to get to the front of the card.
            .addToBackStack(null)

            // Commit the transaction.
            .commit();
}
```

> 翻译：@iOnesmile
> 校对：@misparking
> 原始文档：https://developer.android.google.cn/training/animation/cardflip.html

缩放视图

这节课展示如何实现触摸缩放动画，它是比较实用的，例如在相册中用动画将缩略图扩展到全屏尺寸的图片。

下面是一个触摸缩放动画，内容是把缩略图展开到全屏幕显示。

> 缩放动画
> 点击设备屏幕来重新播放视频

如果你想提前查看完整的工作示例，请点击下载并运行示例应用程序，然后选择缩放示例。请参见下面文件的实现代码：

- `src/TouchHighlightImageButton.java` (一个简单的帮助类，当按下图像按钮时会显示蓝色的触摸高亮。)
- `src/ZoomActivity.java`
- `layout/activity_zoom.xml`

创建视图

为想要缩放的内容创建布局文件，包含小版本和大版本的内容。下面的示例创建了一个缩略图并可点击的 ImageButton 和一个展示图片大图的 ImageView。

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/andr
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="16dp">

        <ImageButton
            android:id="@+id/thumb_button_1"
            android:layout_width="100dp"
            android:layout_height="75dp"
            android:layout_marginRight="1dp"
            android:src="@drawable/thumb1"
            android:scaleType="centerCrop"
            android:contentDescription="@string/description_image_1

    </LinearLayout>

    <!-- This initially-hidden ImageView will hold the expanded/zoo
         the images above. Without transformations applied, it take
         screen. To achieve the "zoom" animation, this view's bound
         from the bounds of the thumbnail button above, to its fina
         bounds.
         -->

    <ImageView
        android:id="@+id/expanded_image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="invisible"
        android:contentDescription="@string/description_zoom_touch_

</FrameLayout>
```

设置缩放动画

当应用布局后，需要设置事件用来触发缩放动画。下面的例子为 ImageButton 添加了一个 View.OnClickListener，当用户点击按钮时执行缩放动画。

```
public class ZoomActivity extends FragmentActivity {
    // Hold a reference to the current animator,
    // so that it can be canceled mid-way.
    private Animator mCurrentAnimator;

    // The system "short" animation time duration, in milliseconds.
    // duration is ideal for subtle animations or animations that o
    // very frequently.
    private int mShortAnimationDuration;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_zoom);

        // Hook up clicks on the thumbnail views.

        final View thumb1View = findViewById(R.id.thumb_button_1);
        thumb1View.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                zoomImageFromThumb(thumb1View, R.drawable.image1);
            }
        });

        // Retrieve and cache the system's default "short" animatio
        mShortAnimationDuration = getResources().getInteger(
                android.R.integer.config_shortAnimTime);
    }
    ...
}
```

缩放视图

你需要在合适的时候进行缩放动画。一般来说，使用动画时需要按照边界来把正常尺寸的视图扩展到大尺寸的视图，下面的方法展示了如何实现缩放动画：

1. 把高清图像资源设置到已经被隐藏的“放大版”的 ImageView 中。下面的示例在 UI 线程上加载一个大的图像资源以简化操作。但是我们需要在一个单独的线程中来加载以免阻塞UI线程，然后再回到UI线程中设置。理想状况下，图片不要大过屏幕尺寸。
2. 计算 ImageView 开始和结束时的边界。
3. 从起始边到结束边同步地动态改变四个点的位置和大小属性 X，Y（SCALE_X 和 SCALE_Y）。这四个动画被加入到了 AnimatorSet，所以它们可以同一时间开始。
4. 缩小已经被放大的图片，当用户点击时只需要执行类似的相反动画。我们可以在 ImageView 中添加一个 View.OnClickListener 来实现它。当点击时，ImageView 缩回到原来缩略图的大小，然后设置它的 visibility 为 GONE 来隐藏。

```
private void zoomImageFromThumb(final View thumbView, int imageResI
    // If there's an animation in progress, cancel it
    // immediately and proceed with this one.
    if (mCurrentAnimator != null) {
        mCurrentAnimator.cancel();
    }

    // Load the high-resolution "zoomed-in" image.
    final ImageView expandedImageView = (ImageView) findViewById(
            R.id.expanded_image);
    expandedImageView.setImageResource(imageResId);

    // Calculate the starting and ending bounds for the zoomed-in i
    // This step involves lots of math. Yay, math.
    final Rect startBounds = new Rect();
    final Rect finalBounds = new Rect();
    final Point globalOffset = new Point();
```

```
    // The start bounds are the global visible rectangle of the thu
    // and the final bounds are the global visible rectangle of the
    // view. Also set the container view's offset as the origin for
    // bounds, since that's the origin for the positioning animatio
    // properties (X, Y).
    thumbView.getGlobalVisibleRect(startBounds);
    findViewById(R.id.container)
            .getGlobalVisibleRect(finalBounds, globalOffset);
    startBounds.offset(-globalOffset.x, -globalOffset.y);
    finalBounds.offset(-globalOffset.x, -globalOffset.y);

    // Adjust the start bounds to be the same aspect ratio as the f
    // bounds using the "center crop" technique. This prevents unde
    // stretching during the animation. Also calculate the start sc
    // factor (the end scaling factor is always 1.0).
    float startScale;
    if ((float) finalBounds.width() / finalBounds.height()
            > (float) startBounds.width() / startBounds.height()) {
        // Extend start bounds horizontally
        startScale = (float) startBounds.height() / finalBounds.hei
        float startWidth = startScale * finalBounds.width();
        float deltaWidth = (startWidth - startBounds.width()) / 2;
        startBounds.left -= deltaWidth;
        startBounds.right += deltaWidth;
    } else {
        // Extend start bounds vertically
        startScale = (float) startBounds.width() / finalBounds.widt
        float startHeight = startScale * finalBounds.height();
        float deltaHeight = (startHeight - startBounds.height()) /
        startBounds.top -= deltaHeight;
        startBounds.bottom += deltaHeight;
    }

    // Hide the thumbnail and show the zoomed-in view. When the ani
    // begins, it will position the zoomed-in view in the place of
    // thumbnail.
    thumbView.setAlpha(0f);
    expandedImageView.setVisibility(View.VISIBLE);

    // Set the pivot point for SCALE_X and SCALE_Y transformations
```

```
    // to the top-left corner of the zoomed-in view (the default
    // is the center of the view).
    expandedImageView.setPivotX(0f);
    expandedImageView.setPivotY(0f);

    // Construct and run the parallel animation of the four transla
    // scale properties (X, Y, SCALE_X, and SCALE_Y).
    AnimatorSet set = new AnimatorSet();
    set
            .play(ObjectAnimator.ofFloat(expandedImageView, View.X,
                    startBounds.left, finalBounds.left))
            .with(ObjectAnimator.ofFloat(expandedImageView, View.Y,
                    startBounds.top, finalBounds.top))
            .with(ObjectAnimator.ofFloat(expandedImageView, View.SC
            startScale, 1f)).with(ObjectAnimator.ofFloat(expandedIm
                    View.SCALE_Y, startScale, 1f));
    set.setDuration(mShortAnimationDuration);
    set.setInterpolator(new DecelerateInterpolator());
    set.addListener(new AnimatorListenerAdapter() {
        @Override
        public void onAnimationEnd(Animator animation) {
            mCurrentAnimator = null;
        }

        @Override
        public void onAnimationCancel(Animator animation) {
            mCurrentAnimator = null;
        }
    });
    set.start();
    mCurrentAnimator = set;

    // Upon clicking the zoomed-in image, it should zoom back down
    // to the original bounds and show the thumbnail instead of
    // the expanded image.
    final float startScaleFinal = startScale;
    expandedImageView.setOnClickListener(new View.OnClickListener(
        @Override
        public void onClick(View view) {
            if (mCurrentAnimator != null) {
```

```
                mCurrentAnimator.cancel();
            }

            // Animate the four positioning/sizing properties in pa
            // back to their original values.
            AnimatorSet set = new AnimatorSet();
            set.play(ObjectAnimator
                        .ofFloat(expandedImageView, View.X, startBo
                        .with(ObjectAnimator
                                .ofFloat(expandedImageView,
                                        View.Y,startBounds.top))
                        .with(ObjectAnimator
                                .ofFloat(expandedImageView,
                                        View.SCALE_X, startScaleFir
                        .with(ObjectAnimator
                                .ofFloat(expandedImageView,
                                        View.SCALE_Y, startScaleFir
            set.setDuration(mShortAnimationDuration);
            set.setInterpolator(new DecelerateInterpolator());
            set.addListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    thumbView.setAlpha(1f);
                    expandedImageView.setVisibility(View.GONE);
                    mCurrentAnimator = null;
                }

                @Override
                public void onAnimationCancel(Animator animation) {
                    thumbView.setAlpha(1f);
                    expandedImageView.setVisibility(View.GONE);
                    mCurrentAnimator = null;
                }
            });
            set.start();
            mCurrentAnimator = set;
        }
    });
}
```

> 翻译：@misparking
> 审核：@iOnesmile
> 原始文档：https://developer.android.google.cn/training/animation/zoom.html#animate

#

使用网络服务发现

网络服务发现（NSD）提供了应用可以在本地网络上访问其他设备的服务。支持 NSD 的设备包括打印机、网络摄像头、HTTPS 服务器和其他移动设备。

NSD 实现了基于 DNS 服务的发现机制（DNS-SD），允许应用程序通过指定类型的服务和指定名称的设备实例去请求服务，后者提供了必要的服务类型。DNS-SD 支持 Android 平台和其他移动平台。

在应用中添加 NSD，可以让用户在同一个网络中辨别出支持应用请求服务的其他设备。这对于各种点对点的应用非常有用，例如文件共享或者多人联机游戏。Android NSD 的 API 可以简化实现这些功能的难度。

这节课主要介绍如何构建一个应用，使其可以在局域网内广播名字和连接信息，并且扫描其它正在做同样事情的应用信息。最后，将介绍如何连接运行在另一台设备上的相同应用。

在网络中注册 NSD 服务

> 注: 这一步骤是可选的。如果我们并不关心在本地网络上广播 app 服务，那么我们可以跳过这一节内容，直接进入 Discover Services on the Network。

在本地网络注册服务，首先需要创建一个 NsdServiceInfo 对象。该对象提供了网络上其他设备在决定是否连接到我们的服务时所使用的信息。

```
public void registerService(int port) {
    // Create the NsdServiceInfo object, and populate it.
    NsdServiceInfo serviceInfo  = new NsdServiceInfo();

    // The name is subject to change based on conflicts
    // with other services advertised on the same network.
    serviceInfo.setServiceName("NsdChat");
    serviceInfo.setServiceType("nsdchat._tcp");
    serviceInfo.setPort(port);
    ....
}
```

这段代码设置服务名称为"NsdChat"。这个服务名称是一个实例名，对网络中的其他设备是可见的。同时该名称将对所有局域网络中使用 NSD 查找本地服务的设备可见。需要注意的是，在网络中该名称是唯一的且 Android 会自动处理名称冲突。在网络中，如果有二个设备同时拥有名称为 “NsdChat” 的应用，其中一个会自动改变服务名称，比如"NsdChat (1)"。

第二个参数设置了服务类型，指定应用使用的协议和传输层。语法是“< *protocol* >. < transportlayer >”。在这段代码中，服务使用了基于 TCP 的 HTTP 协议。如果应用需要提供打印服务（比如网络打印机），那么应该设置服务类型为"_ipp._tcp"。

> 注: 互联网数字分配机构（IANA）提供了用于服务发现协议（如 NSD 和 Bonjour）的官网服务类型列表。我们可以从 the IANA list of service names and port numbers 下载此列表。如果想要使用新的服务类型，我们应该在 IANA Ports and Service registration form 中填写申请表。

当设置服务端口时，需要避免硬编码，以防止与其他应用的冲突。例如，如果应用一直使用 1337 端口，就会与其他使用相同端口的应用存在潜在的冲突。因为信息是通过广播提供给其他应用的，在编译过程中是不需要让其他应用知道你的应用端口的。其他应用可以通过服务广播获取到信息，然后连接到我们的服务。

如果使用的是 socket，那么我们可以简单地将端口设置为 0，来初始化 socket 到任意可用的端口。

```
public void initializeServerSocket() {
    // Initialize a server socket on the next available port.
    mServerSocket = new ServerSocket(0);

    // Store the chosen port.
    mLocalPort =  mServerSocket.getLocalPort();
    ...
}
```

现在我们已经定义了 NsdServiceInfo 对象，我们还需要实现 RegistrationListener 接口。该接口包含了在 Android 中注册的回调函数，用于服务注册和注销的成功或者失败的提醒回调。

```
public void initializeRegistrationListener() {
    mRegistrationListener = new NsdManager.RegistrationListener() {

        @Override
        public void onServiceRegistered(NsdServiceInfo NsdServiceIn
            // Save the service name.  Android may have changed it
            // resolve a conflict, so update the name you initially
            // with the name Android actually used.
            mServiceName = NsdServiceInfo.getServiceName();
        }

        @Override
        public void onRegistrationFailed(NsdServiceInfo serviceInfo
            // Registration failed!  Put debugging code here to det
        }

        @Override
        public void onServiceUnregistered(NsdServiceInfo arg0) {
            // Service has been unregistered.  This only happens wh
            // NsdManager.unregisterService() and pass in this list
        }

        @Override
        public void onUnregistrationFailed(NsdServiceInfo serviceIn
            // Unregistration failed.  Put debugging code here to d
        }
    };
}
```

现在已经拥有注册服务的所有信息了。接下来调用方法 registerService())。

需要注意这个方法是异步的，所以后续的代码都需要在服务注册后的 onServiceRegistered()) 回调方法中去运行。

```
public void registerService(int port) {
    NsdServiceInfo serviceInfo  = new NsdServiceInfo();
    serviceInfo.setServiceName("NsdChat");
    serviceInfo.setServiceType("_http._tcp.");
    serviceInfo.setPort(port);

    mNsdManager = Context.getSystemService(Context.NSD_SERVICE);

    mNsdManager.registerService(
            serviceInfo, NsdManager.PROTOCOL_DNS_SD, mRegistrationL
}
```

在网络中发现服务

生活中网络无处不在，从嘈杂的网络打印机到安静的网络摄影机，以及附近激烈的井字棋游戏。网络服务发现是让我们的应用可以融入这些充满活力的生态系统功能的关键。应用需要在网络中监听服务广播以便获知哪些服务是可用的，并且过滤无效的信息。

服务发现，就像服务注册一样，有二个步骤：设置发现监听的相关回调，并且调用 discoverServices()) 这个异步 API 。

首先，实例化一个实现 NsdManager.DiscoveryListener 接口的匿名类。下面的代码段是一个简单的例子：

```
public void initializeDiscoveryListener() {

    // Instantiate a new DiscoveryListener
    mDiscoveryListener = new NsdManager.DiscoveryListener() {

        //  Called as soon as service discovery begins.
        @Override
        public void onDiscoveryStarted(String regType) {
            Log.d(TAG, "Service discovery started");
        }

        @Override
```

```
        public void onServiceFound(NsdServiceInfo service) {
            // A service was found!  Do something with it.
            Log.d(TAG, "Service discovery success" + service);
            if (!service.getServiceType().equals(SERVICE_TYPE)) {
                // Service type is the string containing the protoc
                // transport layer for this service.
                Log.d(TAG, "Unknown Service Type: " + service.getSe
            } else if (service.getServiceName().equals(mServiceName
                // The name of the service tells the user what they
                // connecting to. It could be "Bob's Chat App".
                Log.d(TAG, "Same machine: " + mServiceName);
            } else if (service.getServiceName().contains("NsdChat")
                mNsdManager.resolveService(service, mResolveListene
            }
        }

        @Override
        public void onServiceLost(NsdServiceInfo service) {
            // When the network service is no longer available.
            // Internal bookkeeping code goes here.
            Log.e(TAG, "service lost" + service);
        }

        @Override
        public void onDiscoveryStopped(String serviceType) {
            Log.i(TAG, "Discovery stopped: " + serviceType);
        }

        @Override
        public void onStartDiscoveryFailed(String serviceType, int
            Log.e(TAG, "Discovery failed: Error code:" + errorCode)
            mNsdManager.stopServiceDiscovery(this);
        }

        @Override
        public void onStopDiscoveryFailed(String serviceType, int e
            Log.e(TAG, "Discovery failed: Error code:" + errorCode)
            mNsdManager.stopServiceDiscovery(this);
        }
    };
```

```
}
```

NSD API 通过使用该接口中的方法通知应用程序搜索何时开始、何时失败以及何时找到可用服务和何时服务丢失（丢失意味着“不再可用”）。请注意当发现了可用服务时，上述代码程序做了数次校对。

1.对比搜索到的服务名称和本地服务的名称，验证设备是否获得自己的广播（是否合法）。
2.核对设备类型，确保服务类型是应用可以连接的。
3.核对服务名称，确保连接正确的应用。

有时候我们不必去核对服务名称，只需要在连接具体应用时进行检查。例如，应用也许只是想与运行在其他设备上的相同应用进行连接。然而，如果应用想要连接网络打印机，只需要确认服务类型为”_ipp._tcp"就可以了。

设置监听后，调用 discoverServices())，并传递相应的参数：应用需要的服务类型、使用的发现协议以及已经创建的监听器。

```
mNsdManager.discoverServices(
        SERVICE_TYPE, NsdManager.PROTOCOL_DNS_SD, mDiscoveryListene
```

在网络中连接服务

当应用在网络中发现可以连接的服务时，首先需要调用 resolveService()) 来确认服务的连接信息，实现NsdManager.ResolveListener 接口以传递这个方法，继而获取包含连接信息的 NsdServiceInfo。

```
public void initializeResolveListener() {
    mResolveListener = new NsdManager.ResolveListener() {

        @Override
        public void onResolveFailed(NsdServiceInfo serviceInfo, int
            // Called when the resolve fails.  Use the error code t
            Log.e(TAG, "Resolve failed" + errorCode);
        }

        @Override
        public void onServiceResolved(NsdServiceInfo serviceInfo) {
            Log.e(TAG, "Resolve Succeeded. " + serviceInfo);

            if (serviceInfo.getServiceName().equals(mServiceName))
                Log.d(TAG, "Same IP.");
                return;
            }
            mService = serviceInfo;
            int port = mService.getPort();
            InetAddress host = mService.getHost();
        }
    };
}
```

当服务解析完成后，应用会接收到包含 IP 地址和端口号的具体服务信息。这就是创建网络连接其他服务的所有信息了。

应用退出时注销服务

在应用的生命周期中适时的开启和关闭 NSD 功能是很重要的。在应用关闭时注销 NSD，可以防止其他应用认为服务还是一直存在而去尝试连接它。另外，服务发现是一个比较消耗性能的操作，当 Activity 暂停时应该停止，而当 Activity 恢复时再重新打开。重写主 Activity 生命周期的方法，然后在适当的时机添加开启、停止服务广播和发现的代码。

```
//In your application's Activity
@Override
    protected void onPause() {
        if (mNsdHelper != null) {
            mNsdHelper.tearDown();
        }
        super.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (mNsdHelper != null) {
            mNsdHelper.registerService(mConnection.getLocalPort());
            mNsdHelper.discoverServices();
        }
    }

    @Override
    protected void onDestroy() {
        mNsdHelper.tearDown();
        mConnection.tearDown();
        super.onDestroy();
    }

    // NsdHelper's tearDown method
        public void tearDown() {
        mNsdManager.unregisterService(mRegistrationListener);
        mNsdManager.stopServiceDiscovery(mDiscoveryListener);
    }
```

> 备注
> 翻译：@misparking
> 审核：@iOnesmile
> 原始文档: https://developer.android.google.cn/training/connect-devices-wirelessly/nsd.html

使用Wi-Fi创建P2P连接

Wi-Fi P2P允许你的应用程序在远远超出蓝牙功能的范围内快速找到并与附近的设备进行交互。

Wi-Fi peer-to-peer（P2P）API 允许应用程序连接到附近的设备，而无需连接到网络或热点（Android 的 Wi-Fi P2P 框架符合 Wi-Fi Direct™ 认证计划）。 如果你的应用被设计为安全的近距离网络的一部分，Wi-Fi Direct 比传统的 Wi-Fi ad-hoc 网络更为合适，因为以下原因：

- Wi-Fi Direct 支持 WPA2 加密。（某些 ad-hoc 网络仅支持WEP加密。）
- 设备可以广播他们提供的服务，这有助于其他设备更容易地发现合适的设备。
- 当确定哪个设备应该是网络的组所有者时，Wi-Fi Direct 将检查每个设备的电源管理，UI和服务功能，并使用此信息来选择最有效地处理服务器功能的设备。
- Android 不支持 Wi-Fi ad-hoc 模式。

本课程将向你展示如何使用 Wifi P2P 查找和连接附近的设备。

设置权限

为了使用 Wi-Fi P2P，请将 manifest 中的 ACCESS_COARSE_LOCATION ，CHANGE_WIFI_STATE，ACCESS_WIFI_STATE 和 INTERNET 权限添加到其中。 Wi-Fi P2P 不需要互联网连接，但它使用标准的 Java 套接字，这需要 INTERNET 权限。 所以你需要以下权限才能使用 Wi-Fi P2P：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/androic
    package="com.example.android.nsdchat"
    ...
    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.INTERNET"/>
    ...
```

设置广播接收器和 P2P 管理器

要使用 Wi-Fi P2P 时，你需要监听某些事件并告知应用这个广播 Intent。 在你的应用中，实例化 IntentFilter 并将其设置为监听以下内容：

```
WIFI_P2P_STATE_CHANGED_ACTION

    指示是否启用 Wi-Fi P2P

WIFI_P2P_PEERS_CHANGED_ACTION

    表示可用的 P2P 列表已更改。

WIFI_P2P_CONNECTION_CHANGED_ACTION

    表示 Wi-Fi P2P 的连接状态发生了变化。

WIFI_P2P_THIS_DEVICE_CHANGED_ACTION

    表示此设备的配置详细信息已更改。
```

-

```
private final IntentFilter intentFilter = new IntentFilter();
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    //  Indicates a change in the Wi-Fi P2P status.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_AC

    // Indicates a change in the list of available peers.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_AC

    // Indicates the state of Wi-Fi P2P connectivity has changed.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANC

    // Indicates this device's details have changed.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHAN

    ...
}
```

在 onCreate（） 方法的最后，获取一个 WifiP2pManager 的实例，并调用其 initialize（） 方法。 此方法返回一个 WifiP2pManager.Channel 对象，稍后你将使用该对象将应用连接到 Wi-Fi P2P 框架。

```
Override

Channel mChannel;

public void onCreate(Bundle savedInstanceState) {
    ....
    mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_S
    mChannel = mManager.initialize(this, getMainLooper(), null);
}
```

现在创建一个新的 BroadcastReceiver 类，你将用于监听系统的 Wi-Fi P2P 状态的更改。 在onReceive（）方法中，添加一个条件来处理上面列出的每个 P2P 状态变化。

```
@Override
public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(act
            // Determine if Wifi P2P mode is enabled or not, alert
            // the Activity.
            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIF
            if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
                activity.setIsWifiP2pEnabled(true);
            } else {
                activity.setIsWifiP2pEnabled(false);
            }
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equ

            // The peer list has changed!  We should probably do so
            // that.

        } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTIO

            // Connection state changed!  We should probably do som
            // that.

        } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTI
            DeviceListFragment fragment = (DeviceListFragment) acti
                    .findFragmentById(R.id.frag_list);
            fragment.updateThisDevice((WifiP2pDevice) intent.getPar
                    WifiP2pManager.EXTRA_WIFI_P2P_DEVICE));

        }
}
```

最后，当你的主要 Activiry 处于活动状态时，添加注册 intent filter （意向过滤器）和 broadcast (广播接收者)的代码，并在活动暂停时取消注册。 执行此操作的最佳方法是 onResume（）和 onPause（）方法。

```
    /** register the BroadcastReceiver with the intent values to be
    @Override
    public void onResume() {
        super.onResume();
        receiver = new WiFiDirectBroadcastReceiver(mManager, mChann
        registerReceiver(receiver, intentFilter);
    }

    @Override
    public void onPause() {
        super.onPause();
        unregisterReceiver(receiver);
    }
```

启动 P2P 搜索

要开始使用 Wi-Fi P2P 搜索附近的设备，请调用 discoverPeers（）。 此方法采用以下参数：

- 当你初始化 P2P 的 mManager 时，你收到的 WifiP2pManager.Channel
- 使用系统调用成功和不成功搜索的方法来实现 WifiP2pManager.ActionListener。

```
mManager.discoverPeers(mChannel, new WifiP2pManager.ActionLis

    @Override
    public void onSuccess() {
        // Code for when the discovery initiation is successf
        // No services have actually been discovered yet, so
        // can often be left blank.  Code for peer discovery
        // onReceive method, detailed below.
    }

    @Override
    public void onFailure(int reasonCode) {
        // Code for when the discovery initiation fails goes
        // Alert the user that something went wrong.
    }
});
```

请记住，这只会启动 P2P 的搜索功能。 discoverPeers（）方法启动搜索过程，搜索到然后立即返回。 如果回调了 onSuccess()，系统将通知你。 此外，搜索将保持活动状态，直到建立连接或形成P2P 组。

获取 P2P List集合

现在写代码来获取和处理 P2P 列表。 首先实现 WifiP2pManager.PeerListListener 接口，它提供有关 Wi-Fi P2P 监听到的 P2P 信息。 此信息还允许你的应用确定何时加入或离开网络。 以下代码片段说明了与 P2P 相关的这些操作：

```
private List<WifiP2pDevice> peers = new ArrayList<WifiP2pDevice>();
    ...

    private PeerListListener peerListListener = new PeerListListene
        @Override
        public void onPeersAvailable(WifiP2pDeviceList peerList) {

            List<WifiP2pDevice> refreshedPeers = peerList.getDevice
            if (!refreshedPeers.equals(peers)) {
                peers.clear();
                peers.addAll(refreshedPeers);

                // If an AdapterView is backed by this data, notify
                // of the change.  For instance, if you have a List
                // available peers, trigger an update.
                ((WiFiPeerListAdapter) getListAdapter()).notifyData

                // Perform any other updates needed based on the ne
                // peers connected to the Wi-Fi P2P network.
            }

            if (peers.size() == 0) {
                Log.d(WiFiDirectActivity.TAG, "No devices found");
                return;
            }
        }
    }
```

当接收到 WIFI_P2P_PEERS_CHANGED_ACTION 的 Intent 时，现在修改广播接收者的onReceive（）方法来调用requestPeers（）。你需要以这种方式把这个 requestPeers() 传给接收者。一种方法是将其作为参数发送给广播接收者的构造函数。

```
public void onReceive(Context context, Intent intent) {
    ...
    else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(ac

        // Request available peers from the wifi p2p manager. This
        // asynchronous call and the calling activity is notified w
        // callback on PeerListListener.onPeersAvailable()
        if (mManager != null) {
            mManager.requestPeers(mChannel, peerListListener);
        }
        Log.d(WiFiDirectActivity.TAG, "P2P peers changed");
    }...
}
```

现在，使用 WIFI_P2P_PEERS_CHANGED_ACTION 意图将触发对 P2P 列表请求的更新。

连接到 P2P

为了连接到 P2P，创建一个新的 WifiP2pConfig 对象，并从要连接的设备 WifiP2pDevice 复制数据。 然后调用connect（）方法。

```
@Override
public void connect() {
    // Picking the first device found on the network.
    WifiP2pDevice device = peers.get(0);

    WifiP2pConfig config = new WifiP2pConfig();
    config.deviceAddress = device.deviceAddress;
    config.wps.setup = WpsInfo.PBC;

    mManager.connect(mChannel, config, new ActionListener() {

        @Override
        public void onSuccess() {
            // WiFiDirectBroadcastReceiver will notify us. Ignore f
        }

        @Override
        public void onFailure(int reason) {
            Toast.makeText(WiFiDirectActivity.this, "Connect failed
                    Toast.LENGTH_SHORT).show();
        }
    });
}
```

如果你组中的每个设备都直接支持 Wi-Fi，则在连接时不需要明确要求组的密码。但是，要允许不支持 Wi-Fi Direct 的设备加入组，你需要通过调用 requestGroupInfo（） 来检索此密码，如以下代码片段所示：

```
mManager.requestGroupInfo(mChannel, new GroupInfoListener() {
  @Override
  public void onGroupInfoAvailable(WifiP2pGroup group) {
      String groupPassword = group.getPassphrase();
  }
});
```

请注意，在 connect（） 方法中实现的 WifiP2pManager.ActionListener 仅在启动成功或失败时通知你。要监听连接状态的变化，请执行 WifiP2pManager.ConnectionInfoListener 接口。 它的 onConnectionInfoAvailable（） 回调将在连接状态更改时通知你。 如果多个设备要连接到单个设备（如具有3个或更多播放器的游戏或聊天应用程序），则会将一个设备指定为“组所有者”。 你可以按照“创建组”部分中的步骤将特定设备指定为网络组所有者。

```
@Override
public void onConnectionInfoAvailable(final WifiP2pInfo info) {

    // InetAddress from WifiP2pInfo struct.
    InetAddress groupOwnerAddress = info.groupOwnerAddress.getHostA

    // After the group negotiation, we can determine the group owne
    // (server).
    if (info.groupFormed && info.isGroupOwner) {
        // Do whatever tasks are specific to the group owner.
        // One common case is creating a group owner thread and acc
        // incoming connections.
    } else if (info.groupFormed) {
        // The other device acts as the peer (client). In this case
        // you'll want to create a peer thread that connects
        // to the group owner.
    }
}
```

现在回到广播接收者的 onReceive（） 方法，并修改监听 WIFI_P2P_CONNECTION_CHANGED_ACTION 意图的部分。 收到这个意图后，调用 requestConnectionInfo（）。 这是一个异步调用，所以结果将被你提供的连接信息监听器作为参数接收。

```
    ...
    } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.eq

        if (mManager == null) {
            return;
        }

        NetworkInfo networkInfo = (NetworkInfo) intent
                .getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_IN

        if (networkInfo.isConnected()) {

            // We are connected with the other device, request conn
            // info to find group owner IP

            mManager.requestConnectionInfo(mChannel, connectionList
        }
        ...
```

创建一个组

如果你希望运行应用的设备作为包含旧设备的网络的组所有者（即不支持 Wi-Fi Direct 的设备），则遵循与“连接到 P2P ”相同的步骤部分，除了你使用 createGroup（） 而不是 connect（） 创建一个新的 WifiP2pManager.ActionListener 。 WifiP2pManager.ActionListener 中的回调处理是相同的，如下面的代码片段所示：

```
mManager.createGroup(mChannel, new WifiP2pManager.ActionListener()
    @Override
    public void onSuccess() {
        // Device is ready to accept incoming connections from peer
    }

    @Override
    public void onFailure(int reason) {
        Toast.makeText(WiFiDirectActivity.this, "P2P group creation
                Toast.LENGTH_SHORT).show();
    }
});
```

> 注意：如果网络中的所有设备都支持 Wi-Fi Direct ，则可以在每个设备上使用 connect（） 方法，因为该方法会自动创建组并选择组所有者。

创建组后，可以调用 requestGroupInfo（） 来检索网络上对等体的详细信息，包括设备名称和连接状态

翻译者：JackWaiting

翻译原址：https://developer.android.com/training/connect-devices-wirelessly/wifi-direct.html#create-group

使用Wifi P2P进行服务搜索

本阶段的第一课《使用网络服务搜索》，展示如何搜索连接了本地网络的服务。然而使用 WiFi 对等网络（P2P）搜索服务可以直接搜索到附近设备的服务，而不需要连接网络。你可以让该服务运行在手机。这些功能可以帮助我们在两个应用间通讯，即使没有有效的网络或热点。

> 尽管这一套 API 与之前课程《网络服务搜索》 API 大纲目的类似，但在代码实现还是有很大不同。这节课解说使用 WiFi P2P 如何从其它设备中搜索有效服务。这节课假定你已经熟悉了 WiFi P2P API。

设置清单文件

使用 WiFi P2P 时需要添加权限 CHANGE_WIFI_STATE, ACCESS_WIFI_STATE, 和 INTERNET 到清单文件。虽然 WiFi P2P 使用标准 Java Socket，并不需要有互联网连接，但是在 Android 中使用这些时需要请求权限。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/androi
    package="com.example.android.nsdchat"
    ...

    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.INTERNET"/>
    ...
```

添加本地服务

如果提供本地服务，需要为它注册服务搜索。一旦本地服务注册后，Framework 会自动响应服务搜索请求。

创建本地服务：

1. 创建 WifiP2pServiceInfo 对象
2. 配置关于服务的信息
3. 调用 addLocalService()) 方法注册本地服务

```
private void startRegistration() {
    //  Create a string map containing information about your servi
    Map record = new HashMap();
    record.put("listenport", String.valueOf(SERVER_PORT));
    record.put("buddyname", "John Doe" + (int) (Math.random() * 100
    record.put("available", "visible");

    // Service information.  Pass it an instance name, service type
    // _protocol._transportlayer , and the map containing
    // information other devices will want once they connect to thi
    WifiP2pDnsSdServiceInfo serviceInfo =
            WifiP2pDnsSdServiceInfo.newInstance("_test", "_presence

    // Add the local service, sending the service info, network cha
    // and listener that will be used to indicate success or failur
    // the request.
    mManager.addLocalService(channel, serviceInfo, new ActionLister
        @Override
        public void onSuccess() {
            // Command successful! Code isn't necessarily needed he
            // Unless you want to update the UI or add logging stat
        }

        @Override
        public void onFailure(int arg0) {
            // Command failed.  Check for P2P_UNSUPPORTED, ERROR, c
        }
    });
}
```

搜索附近服务

第一步应创建回调方法来通知应用中的有效服务。创建 WifiP2pManager.DnsSdTxtRecordListener 对象来监听回调的消息，该消息由其它设备随意的广播。当收到一条消息时，拿到设备地址或其它任何你需要的相关信息以便后续使用。下面的例子采用消息的 “buddyname” 字段作为它的标识。

```
final HashMap<String, String> buddies = new HashMap<String, String>
...
private void discoverService() {
    DnsSdTxtRecordListener txtListener = new DnsSdTxtRecordListener
        @Override
        /* Callback includes:
         * fullDomain: full domain name: e.g "printer._ipp._tcp.loc
         * record: TXT record dta as a map of key/value pairs.
         * device: The device running the advertised service.
         */

        public void onDnsSdTxtRecordAvailable(
                String fullDomain, Map record, WifiP2pDevice device
                Log.d(TAG, "DnsSdTxtRecord available -" + record.to
                buddies.put(device.deviceAddress, record.get("buddy
            }
        };
    ...
}
```

获取服务信息时，创建 WifiP2pManager.DnsSdServiceResponseListener 对象，它接收实际的描述和连接信息。之前的代码块实现的 Map 对象匹配设备地址和名称。服务返回 DNS 记录和相关的服务信息。当完成这两个监听的实现后，使用 setDnsSdResponseListeners()) 方法添加它们到 WifiP2pManager 中。

```
private void discoverService() {
...

    DnsSdServiceResponseListener servListener = new DnsSdServiceRes
        @Override
        public void onDnsSdServiceAvailable(String instanceName, St
                WifiP2pDevice resourceType) {

                // Update the device name with the human-friendly v
                // the DnsTxtRecord, assuming one arrived.
                resourceType.deviceName = buddies
                        .containsKey(resourceType.deviceAddress) ?
                        .get(resourceType.deviceAddress) : resource

                // Add to the custom adapter defined specifically f
                // wifi devices.
                WiFiDirectServicesList fragment = (WiFiDirectServic
                        .findFragmentById(R.id.frag_peerlist);
                WiFiDevicesAdapter adapter = ((WiFiDevicesAdapter)
                        .getListAdapter());

                adapter.add(resourceType);
                adapter.notifyDataSetChanged();
                Log.d(TAG, "onBonjourServiceAvailable " + instanceN
        }
    };

    mManager.setDnsSdResponseListeners(channel, servListener, txtLi
    ...
}
```

现在调用 addServiceRequest()) 方法创建服务请求，该方法接收一个监听来报告成功或失败。

```
serviceRequest = WifiP2pDnsSdServiceRequest.newInstance();
mManager.addServiceRequest(channel,
        serviceRequest,
        new ActionListener() {
            @Override
            public void onSuccess() {
                // Success!
            }

            @Override
            public void onFailure(int code) {
                // Command failed.  Check for P2P_UNSUPPORTED, ERRO
            }
        });
```

最后调用 discoverServices()) 方法。

```
mManager.discoverServices(channel, new ActionListener() {

    @Override
    public void onSuccess() {
        // Success!
    }

    @Override
    public void onFailure(int code) {
        // Command failed.  Check for P2P_UNSUPPORTED, ERROR, or BU
        if (code == WifiP2pManager.P2P_UNSUPPORTED) {
            Log.d(TAG, "P2P isn't supported on this device.");
        else if(...)
            ...
    }
});
```

如果一切顺利，很好，已经完成了。如果遇到问题，记得检查 WifiP2pManager.ActionListener 的异步监听，它提供了成功和失败的回调。如果回调 onFailure()) 方法，则可能是代码有问题。下面是可能的错误码和它们的意思：

1. `P2P_UNSUPPORTED`
   当前设备不支持 Wi-Fi P2P

2. `BUSY`
   系统忙

3. `ERROR`
   内部错误，操作失败

> 翻译：@iOnesmile
> 审核人：@JackWaiting
> 原始文档：https://developer.android.google.cn/training/connect-devices-wirelessly/nsd-wifi-direct.html#manifest

图标设计指导原则

> 已经有新版针对应用设计师的指南！
> 可在 **Android** 设计中查看最新的文档，包括更多关于图标的指导原则。

在整个用户界面中保持统一的外观和风格，对于你的产品来说是增值的。精简的图形风格也会让用户觉得你的用户界面对于用户来说，看起来更专业。

此文档提供帮助你创建针对你应用用户界面不同部分的图标，这些图标适配采用 Android 2.x 框架的一般样式。遵循这些设计指南将帮助你创一个优雅并且统一的用户体验。

以下文档是针对 Android 应用中使用的所有的常见类型的图标的细节指南探讨：

启动器图标

启动器图标是在启动器窗口和设备主屏幕上代表你应用的图片。

菜单图标

菜单图标是当用户点击菜单按钮的时候，放置在可选菜单中的图片元素。

操作栏图标

操作栏图标是在操作栏中代表操作条目的图片元素。

状态栏图标

状态栏图标是在状态栏中代表你应用的通知。

标签图标

标签图标是在多标签界面中的代表单个标签的图片元素。

对话框图标

对话框图标是在弹出的对话框中显示的，用来提醒用户进行交互的图标。

列表图标

列表图标是在 ListView 中代表列表条目的。例如设置应用。为了更加快速的创建你的图标，你可以下载 Android 图标模板包。

Android 图标模板包的使用

Android 图标模板包，是一个设计、结构和层样式的模板集合，能够让你更加容易的创建符合文档中指南的要求的图标。我们建议你开始图标设计之前，先下载模板包存档。

图片模板是以 Adobe Photoshop 文件格式（.psd）提供的，这样格式的文件保留了我们创建 Android 平台的标准图标时候的图层和设计处理。你可以用任何兼容的图片编辑程序载入模板文件，尽管你对图层和设计处理的能力，可能会因为你使用的程序不同而不同。

你可以从以下链接获取最新的图标模板包：

下载最新的 Android 4.0 图标模板包»

对于较早版本的图标模板包，请参见本页面右上角的下载部分。

提供特定密度的图标集合

Android 被设计成可以在各种不同屏幕尺寸和分辨率设备上运行的系统。当你设计应用图标的时候，切记你的应用将会被安装到这些设备上。如多屏幕支持文档中介绍的一样，Android 平台让你通过这种方式直截了当的提供图标，能够在任何设备上正确的显示，而不用考虑设备的屏幕尺寸和分辨率。

一般来说，推荐的针对每一个广义的屏幕密度创建一套单独的图标。然后将他们存储在你应用的指定密度资源目录。当你的应用运行时，Android 平台会检查设备屏幕特性并且载入正确的指定密度的图标资源。更多关于如何在你的应用中存储指定密度资源，请参见屏幕尺寸和密度的资源目录限定符。

如何创建和管理针对多密度的图标集合的小提示，请参见设计师的小提示。

当你开发你应用的图标或者其他图片资源的时候，你会发现以下小提示比较有用。小提示假设你使用 Adobe Photoshop 或者类似的光栅和矢量图片编辑程序。

针对图标资源使用常用的命名约定

在同一个目录中的文件命名可以尝试将相关资源分在一组，这样可以通过字母表排序归类到一起。尤其是当针对同一种图标类型使用相同前缀。例如：

资源类型	前缀	举例
图标	`ic_`	`ic_star.png`
启动器图标	`ic_launcher`	`ic_launcher_calendar.png`
菜单图标和操作栏图标	`ic_menu`	`ic_menu_archive.png`
状态栏图标	`ic_stat_notify`	`ic_stat_notify_msg.png`
标签图标	`ic_tab`	`ic_tab_recent.png`
对话框图标	`ic_dialog`	`ic_dialog_info.png`

请注意：并不是强制要求你使用任何类型共享前缀，这样做，仅仅是为了让你更加方便。

为多密度创建一个组织文件的工作空间

多屏幕密度的支持，意味着你需要创建同一个图标的多个版本。为了保证多份文件的拷贝的安全性和更容易找到，我们建议在你的工作空间中创建一个组织针对每个分辨率的资源文件目录结构。例如：

```
art/...
    ldpi/...
        _pre_production/...
            working_file.psd
        finished_asset.png
    mdpi/...
        _pre_production/...
            working_file.psd
        finished_asset.png
    hdpi/...
        _pre_production/...
            working_file.psd
        finished_asset.png
    xhdpi/...
        _pre_production/...
            working_file.psd
        finished_asset.png
```

这种结构和在你应用资源中存储的最终处理好的资源结构保持一致的。因为你工作空间的结构和应用的类似，你可以快速决定哪个资源可以复制到应用资源目录。根据不同密度区分的资源也能帮你检测不同密度文件名的差异，因为不同密度的相关资源必须采用相同的文件名称。

为了对比，以下给出的是一个典型应用的资源目录结构：

```
res/...
    drawable-ldpi/...
        finished_asset.png
    drawable-mdpi/...
        finished_asset.png
    drawable-hdpi/...
        finished_asset.png
    drawable-xhdpi/...
        finished_asset.png
```

尽可能的使用矢量图

许多诸如 Adobe Photoshop 的图片编辑程序允许你使用矢量图、栅格图层和效果的组合。如果可以的话，尽量使用矢量图，这样就可以在需要的时候，资源可以无损放大，而不会导致细节损失和脆边。

从大画板开始

因为你需要针对不同的屏幕密度创建资源，是在一个能包含多个目标图标尺寸的大画板上的图标设计的方式，是一个不错的选择。例如，启动器图标是 96，72，48，或 36 pixels 宽，取决于屏幕密度。如果你刚开始是在 864x864 的画板上开始设计启动器图标的话，就会比较容易和清晰的调整图标到最终目标尺寸的资源。

缩放的时候，如果有必要就重绘位图图层

如果你是从一个位图图层放大一张图片，而不是从一个矢量图层放大一张图片，这些图层需要手动重绘，以便在更高密度中清晰显示。例如，如果一个 60x60 的圆以 mdpi 密度的位图绘制的话，那么对于一个 90x90 的圆，应当以 hdpi 密度绘制。

保存图片资源的时候，移除非必须的元数据

尽管 Android SDK 工具会在应用资源打包成应用二进制文件的时候，自动压缩 PNG 图片，但是移除 PNG 资源的非必须的头和元数据是一个好的习惯。诸如 OptiPNG 或 Pngcrush 的工具可以确保元数据被移除，优化图片资源文件的大小。

确保不同密度相关资源使用相同的文件名

每个密度的对应的图片资源文件必须使用相同的文件名，但是需要分别存储到指定密度的资源目录中。这种方式可以让系统根据设备屏幕特性来查找和载入正确的资源。正因为如此，需要确保在每个目录下的资源保持一致，文件名不要使用特定密度前缀。

更多关于特定密度资源和系统如何使用这些资源来适配不同设备，请参见多屏幕支持。

启动器图标

> 已经有新版针对应用设计师的指南！
> 可在 **Android** 设计中查看最新的文档，包括更多关于图标的指导原则。

启动器图标代表着你的应用的图像。启动器图标是启动器应用使用的出现在用户主屏幕上的。启动器图标也可以用来表示进入你应用的快捷方式（例如，联系人快捷图标可以用来打开一个联系人的细节信息）。

如提供特定密度图标集合和多屏幕支持中所述，你应当为所有广义屏幕密度，包括低、中、高和超高密度屏幕创建单独的图标。这样可以确保可以安装在不同设备上你的应用图标可以正确显示。参见设计师小提示来获取如果处理多套图标的建议。

Google Play 也要求你提供应用高分辨率的启动器图标，用于应用列表的显示。更多关于这个的细节，请参见下面的 Google Play 上的应用图标。

> 备注：与所有 Android 版本有关的启动器图标指南已经被重写。如果你需要查看旧的指南，参见启动器图标指南归档。

启动器图标目标

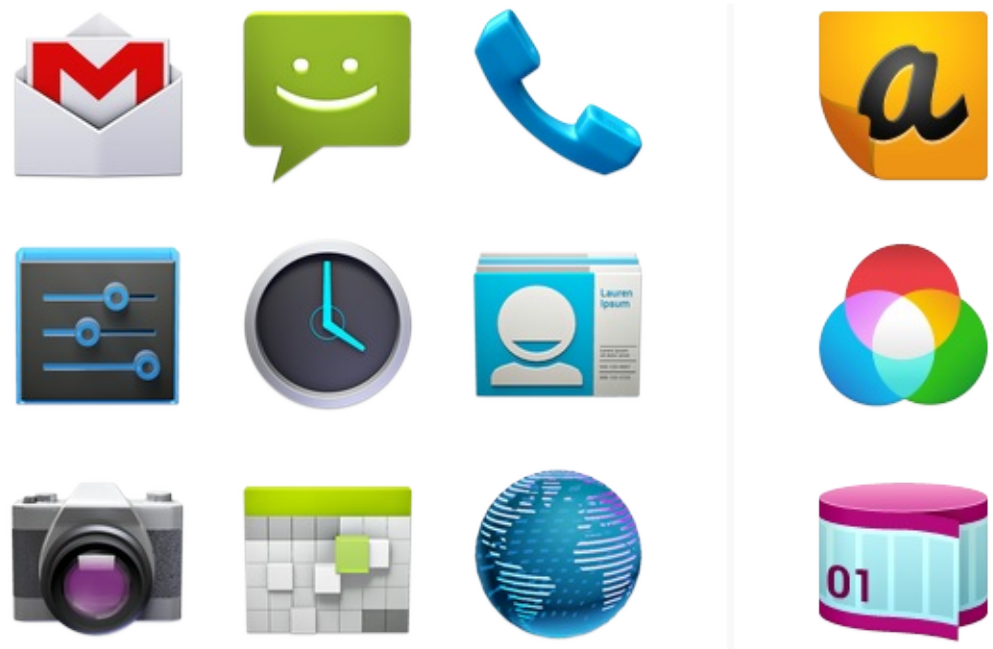


图**1**：系统应用（左侧）和第三方应用的启动器图标举例（右侧）。

应用启动器图标有以下三个基本目标：
1.宣传品牌和讲述应用故事。
2.帮用户在 Google Play 上发现应用。
3.在启动器上有良好的功能表现。

宣传品牌故事

应用启动器图标是一个展示品牌和提示关于你应用的故事。所以，你应当：

- 创建一个独一和好记的图标。
- 使用适合你品牌的配色方案。
- 不要尝试将图标复杂化。一个简单的图标会更有影响力和更容易记忆。
- 避免在图标中包含应用名称。应用名将会始终显示在图标旁边。

帮助用户在 Google Play 上发现应用

应用启动器图标是潜在用户在 Google Play 上获取你应用的第一印象。高质量的应用图标会在用户滑动应用列表的时候，影响用户去发现更多信息。

应用图标的质量是有影响的。一个设计良好的图标，可以作为你应用同样高质量的强烈信号。可以考虑让图标设计师来设计应用启动器图标。

> 备注：Google Play 要求你提供高分辨率的图标；更多关于这个细节，请参见下面的Google Play 应用图标。

在启动器上有良好的功能表现

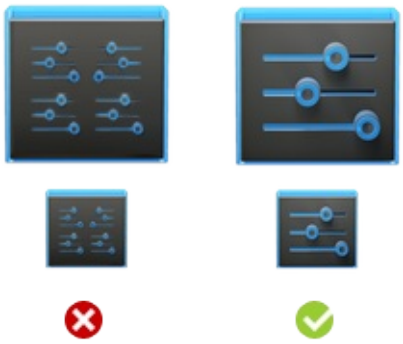
启动器上的应用图标是用户与应用最常交互的。一个成功的应用启动器图标在所有的情况下都看起来很好：在任何背景以及任何图标和应用小部件旁边。为了达到此效果，图标应当遵循：

- 小尺寸情况下，有良好的交互。
- 多种不同背景下，表现良好。
- 反射启动器的隐含照明模型（顶部照明）。
- 如果图标是 3D 的，使用前置透视图最佳。
    - 3D 图标采用浅啮合处理更加。
- 有一个快速识别的独一轮廓；并不是所有的 Android 应用图标应当是圆形的。
    - 图标不应当呈现较大图像的裁剪视图。

- 和其他图标保持类似的权重。图标太细长或者没有占用足够的空间，可能都不能成功的引起用户的注意，或者无法再所有背景上表现良好。

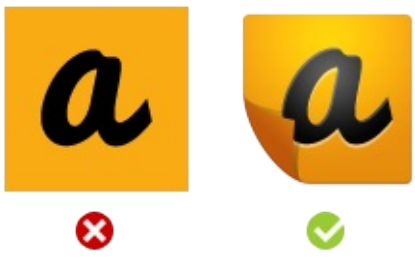
该做什么和不该做什么

以下有一些当你考虑创建应用图标时候的“该做与不该做”举例：



图标不能过于复杂。请记住启动器图标经常在小尺寸下使用，所以在小尺寸下也能够分辨。

不能裁剪图标。相对于其他图标来说应当有类似的权重。过于细长的图标，不能在所有背景下都表现良好。



图标应当使用 Alpha 通道，并且不能简单的处理为全帧图像。可以通过微妙的有吸引力的视觉处理来区分你的图标。

尺寸和格式

启动器图标应当是带有 Alpha 通道透明度的 32 位 PNG 图片。最终的启动器图标的尺寸与以下表格中给定的广义的屏幕密度有关：

表1：针对每一个给定的广义的屏幕密度最终启动器图标尺寸。

	ldpi (120 dpi)（低密度屏幕）	mdpi (160 dpi)（中等密度屏幕）	hdpi (240 dpi)（高密度屏幕）	xhdpi (320 dpi)（超高密度屏幕）	xxhdpi (480 dpi)(超超高密度屏幕)	xxxhdpi (640 dpi)（超超超高密度屏幕）
启动器图标尺寸	36 x 36 px	48 x 48 px	72 x 72 px	96 x 96 px	144 x 144 px	192 x 192 px

你来可以在启动器图标中包含几个像素的内边距，从而与相邻图标保持一个较好的视觉权重。例如，一个 96 x 96 像素的超高分辨率的启动器图标可以包含一个 88 x 88 像素图形，然后每边包含一个 4 像素的内边距。这个边距可以用来为巧妙的阴影绘制腾出空间，这样可以保证你的启动器图标在任何背景颜色下都是易读的。

Google Play 上的应用图标

如果你要在 Google Play 上发布你的应用，你需要在开发者控制台上传的时候，提供一个 512 x 512 像素的高分辨率的应用图标。这个图标会在 Google Play 上的不同位置使用，但是不会替换你的启动器图标。

关于创建易于放大到 512x512 的高分辨率启动器图标的小提示和建议，请参阅设计师小提示。

关于 Google Play 高分辨率应用图标的信息和规范，请参阅以下文章：

你应用图片资源（Google Play 帮助文档）»

> 翻译：@ifeegoo
> 校对：?
> 原始文档：https://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html

请求共享文件

当应用程序想要访问由其他应用程序共享的文件时，请求应用程序（客户端）通常会向共享文件的应用程序（服务器）发送请求。在大多数情况下，请求在服务器应用程序中启动一个 Activity ，显示它可以共享的文件。 用户选择一个文件，之后服务器应用程序将文件的内容URI返回给客户端应用程序。

本课将向您展示客户端应用程序如何从服务器应用程序请求文件，从服务器应用程序接收文件的内容URI，以及如何使用内容URI打开文件。

发送文件请求

要从服务器应用程序请求文件，客户端应用程序使用包含操作（如 ACTION_PICK ）和客户端应用程序可以处理的 MIME 类型的 Intent 调用 startActivityForResult 。

例如，以下代码段演示了如何向服务器应用程序发送 Intent 以启动共享文件中描述的Activity ：

```
    public class MainActivity extends Activity {
    private Intent mRequestFileIntent;
    private ParcelFileDescriptor mInputPFD;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mRequestFileIntent = new Intent(Intent.ACTION_PICK);
        mRequestFileIntent.setType("image/jpg");
        ...
    }
    ...
    protected void requestFile() {
        /**
         * When the user requests a file, send an Intent to the
         * server app.
         * files.
         */
            startActivityForResult(mRequestFileIntent, 0);
        ...
    }
    ...
}
```

访问请求的文件

服务器应用程序在 Intent 中将文件的内容URI发送回客户端应用程序。 这个 Intent 传递给客户端应用程序的覆盖 startActivityForResult 。 一旦客户端应用程序具有文件的内容URI，它就可以通过获取其 FileDescriptor 来访问文件。

在此过程中保留文件安全性，因为内容URI是客户端应用程序接收的唯一数据。 由于此URI不包含目录路径，因此客户端应用程序无法发现和打开服务器应用程序中的任何其他文件。 只有客户端应用程序可以访问该文件，并且只有服务器应用程序授予的权限。 权限是临时的，因此一旦客户端应用程序的任务堆栈完成，该文件就不再在服务器应用程序之外访问。

下一个片段演示了客户端应用程序如何处理从服务器应用程序发送的 Intent ，以及客户端应用程序如何使用内容URI获取 FileDescriptor ：

```
/*
 * When the Activity of the app that hosts files sets a result and
 * finish(), this method is invoked. The returned Intent contains t
 * content URI of a selected file. The result code indicates if the
 * selection worked or not.
 */
@Override
public void onActivityResult(int requestCode, int resultCode,
        Intent returnIntent) {
    // If the selection didn't work
    if (resultCode != RESULT_OK) {
        // Exit without doing anything else
        return;
    } else {
        // Get the file's content URI from the incoming Intent
        Uri returnUri = returnIntent.getData();
        /*
         * Try to open the file for "read" access using the
         * returned URI. If the file isn't found, write to the
         * error log and return.
         */
        try {
            /*
             * Get the content resolver instance for this context,
             * to get a ParcelFileDescriptor for the file.
             */
            mInputPFD = getContentResolver().openFileDescriptor(ret
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            Log.e("MainActivity", "File not found.");
            return;
        }
        // Get a regular file descriptor for the file
        FileDescriptor fd = mInputPFD.getFileDescriptor();
        ...
    }
}
```

方法 openFileDescriptor（）为文件返回一个 ParcelFileDescriptor 。 从此对象，客户端应用程序获取一个 FileDescriptor 对象，然后可以使用它来读取文件。

翻译人：JackWaiting

原文地址：https://developer.android.com/training/secure-file-sharing/request-file.html#OpenFile

USB

USB 主机及附件概览

Android 通过两种模式：USB 附件和 USB 主机，来支持多种 USB 外围设备和 Android USB 附件（实现了 Android 附件协议的硬件）。在 USB 附件模式下，外部 USB 硬件扮演 USB 主机的角色。附件的例子可能包含机器人控制器、停车站、诊断设备和音乐设备，公用电话亭，读卡器等多种设备。这种方式无法让 Android 驱动的设备具备与 USB 硬件交互的主机能力。Android USB 附件要能够与 Android 驱动的设备协同工作的话，必须遵循《Android 附件通信协议》。在 USB 主机模式下，Android 驱动的设备扮演主机的角色。设备的例子包括数码相机、键盘、鼠标和游戏控制器。设计出来的范围很广的应用和环境下的 USB 设备可以与在 Android 设备正确通信的 Android 应用进行交互。

图 1 展示了两种模式之间的区别。当 Android 驱动的设备处于主机模式的情况下，它扮演着 USB 主机的角色并为总线供电。当 Android 驱动的设备处于 USB 附件模式的情况下，连接的 USB 硬件（这种情况下作为 Android USB 附件）扮演着主机的角色并为总线供电。

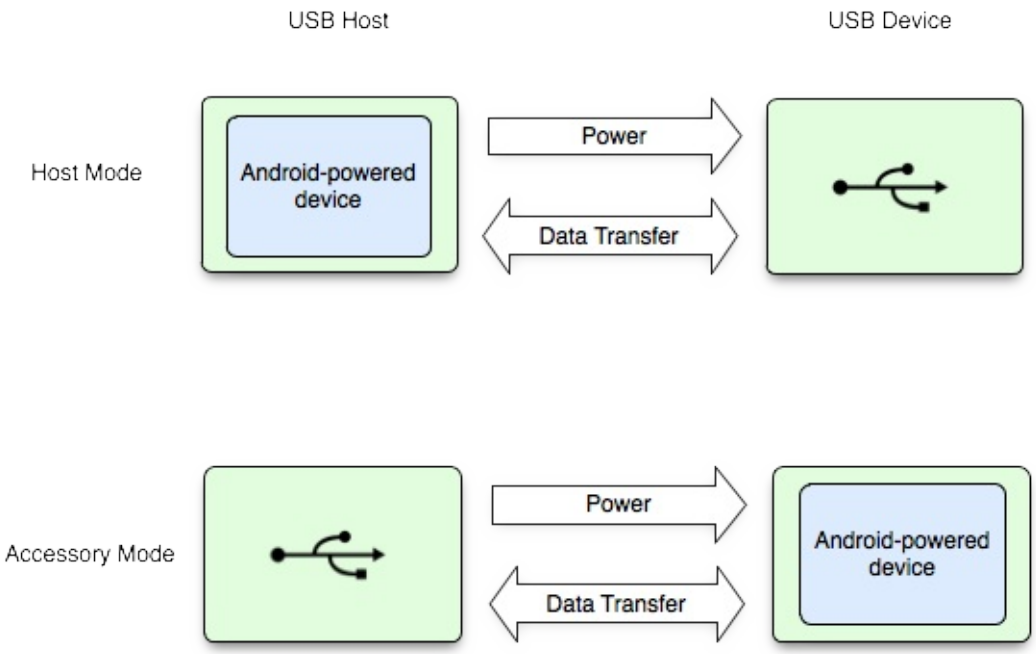


图 1：USB 主机及附件模式

USB 附件和主机模式在 Android 3.1 (API Level 12) 及以后的版本直接支持。USB 附件模式同样可以作为一个附加库移植到 Android 2.3.4 （API Level 10）上，这样可以支持更大范围的设备。设备制造商可以选择是否在设备系统镜像中包含附加库。

> 备注：针对 USB 主机和附件模式的支持，最终取决于设备的硬件，而和系统平台无关。你可以通过 `<uses-feature>` 元素来过滤出那些支持 USB 主机和附件的设备，更多细节请参见《USB 附件和主机》文档。

调试注意事项

当调试使用 USB 附件和主机功能的应用，你很有可能会将 USB 硬件连接到你的 Android 驱动的设备上。这种情况会阻止你通过 USB 与 Android 驱动的设备建立 adb 连接。你仍然可以通过网络连接访问 adb。通过网络连接启用 adb 步骤如下：

1.通过 USB 将 Android 驱动设备与你的电脑连接。
2.在你的 SDK `platform-tools/` 目录下，在命令提示符中输入 `adb tcpip 5555` 。
3.输入 `adb connect <device-ip-address>:5555` 然后就可以连接上 Android 驱动设备，然后就可以像 `adb logcat` 使用 adb 常用命令了。
4.可以通过输入 `adb usb` 来使你的设备监听 USB。