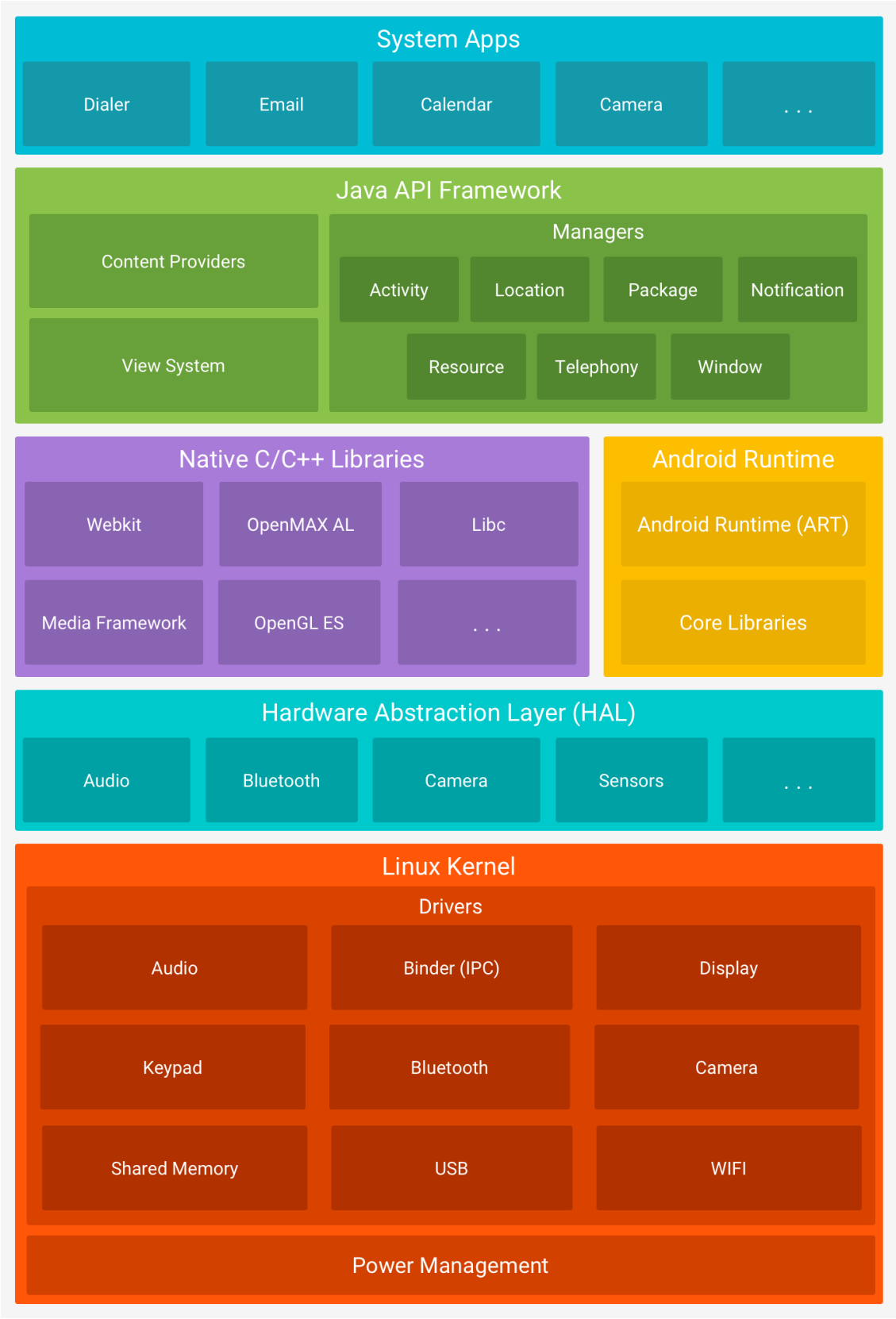
Android应用和开发环境

Android的简介

- Android是由Andy Ruby创立的一个手机操作系统，后被Google收购。
- 目标：Google希望能与各方共同建立一个标准化、开放式的移动电话软件平台，从而在移动产业内形成一个开放式的操作平台。

Android9.x平台特性

-



- Android的底层建立在linux之上，主要由操作系统、中间件、用户界面和应用软件4层组成
  - 设计思路：采用一种被称为软件叠层(Software Stack)的方式进行构建。
  - 作用
    - 使得层与层之间互相分离，明确各层的分工
    - 保证了层与层的低耦合，当下层的层内或层下发生改变时，上层应用程序无需任何改变
  - 和网络以及java web的后端一样都使用了分层的思想去解决问题。
- Android系统主要由6部分组成。**序号从小到大，越来越底层。**

  1. 系统App层

    - Android系统将包含一系列核心App，如电话拨号，联系人。
    - 普通开发者开发的各种Android程序，都属于这一层
2. Java API框架层
    - 提供了大量API供开发者使用
    - 调用原生C/C++库的服务。
3. 原生C/C++库
    - Android包含一套被不同组件所使用的C/C++库的集合。一般来说，Android应用开发者不能直接调用这套C/C++库集，而是通过它上面的JavaAPI框架调用这些库
    - SQLlite：供所有应用程序使用的功能强大的轻量级关系数据库
    - WebKit
    - OpenMAX
    - Libc
    - Media Framework
    - SGL
    - OpenGL ES
4. Android运行时
    - 由Android核心库和ART（Android runtime）组成
5. 硬件抽象层（HAL）
    - 主要提供了对linux内核的封装，向上提供蓝牙、摄像头等设备的编程接口，向下隐藏底层的实现细节。
6. Linux内核
    - 提供了安全性，内存管理、进程管理、网络协议栈和驱动模型等核心服务。

Gradle自动化构建

- Gradle的运行需要Java_Home环境变量
- 使用gradle 4.10.2

Gradle文件结构

- bin：包含gradle的命令-gradle
- docs：包含用户手册、DSL参考指南、API文档
- lib：包含Gradle核心，以及它依赖的JAR包
- media：主要包含Gradle的一些图标
- sample：样例
- src：Gradle源代码

Gradle项目结构

- Gradle构建文件的默认名字为build.gradle
- project:存放整个项目的全部资源
    - src:分类存放各种源文件、资源文件
        - main：主要存放与项目有关的源文件和资源
            - java：存放与项目相关的资源
            - resource：存放与项目相关的资源
        - test：主要存放与测试有关的源文件和资源
            - java：存放与测试相关的资源
            - resource：存放与测试相关的资源
    - build：存放编译后的class文件、资源的文件夹，该文件夹与src具有对应关系

- libs：存放第三方JAR包的文件夹
- build.gradle：gradle构建文件

构建文件

- Gradle采用**领域对象模型**的概念来组织构建文件，在整个构建文件中涉及最核心的API
- Project：代表项目，通常一份构建文件代表一个项目。Project包含大量属性和方法
- TaskContainer：任务容器。每个Project都会维护一个TaskContainer类型的tasks属性。Project和TaskContainer有一一对应的关系
- Task：代表Gradle要执行的一个任务。Task允许指定它依赖的任务、该任务的类型，也可以通过configure（）方法配置任务。它还提供doFirst（）、doLast（）方法添加Action。
  - Action对象和Closure对象都可以代表Action。
  - Closure代表一个闭包，所以Action实际上就代表一个代码块

Task创建

- 调用Project的task（）方法创建Task
- 调用TaskContainer的create（）方法创建Task

Task指定如下3个常用属性

- dependsOn：指定该Task所依赖的其他Task
- type:指定该task的类型
- 代码块：通过传入的代码块参数配置Task
  - ```
    task hello{
        println "hello world"
    }
    //使用
    gradle hello
    ```

属性定义

1. 为已有的属性指定属性值
   - Project、Task都是Gradle提供的API，他们本身具有内置属性，因此可以在构建文件中为这些属性指定属性值
   - project的常用属性
     - name：项目的名字
     - path：项目的绝对路径
     - description：项目的描述信息
     - buildDir：项目的构建结果存放路径
     - version：项目的版本号
     - ```
       version = 1.0
       description = "project的属性"
       task hello{
           description = "task的属性"
           println "hello world"
       }
       ```
2. 通过ext添加属性

- Gradle中所有实现了ExtensionAware接口的API都可通过ext来添加属性
- ```
  ext.prop1 = "111"
  ext{
      prop2 = "222"
  }
  task hello{
      prop3 = "333"
      println "hello world"
  }
  ```

3. 通过-P选项添加属性
4. 通过JVM参数添加属性

增量式构建

- 背景：如果每次执行任务都没有造成任何改变，那么重复执行就没有意义
- 如何判断是否产生变化？
  - Gradle将每个任务当做一个黑盒子：只要该任务的输入部分和输出部分都没有改变，Gradle就认为该任务的执行没有造成任何改变
    - 输入：Gradle的task使用inputs属性来代表任务的输入
      - 是一个TaskInput的对象
      - 可以设置文件、文件集、目录、属性等
    - 输出：Gradle的task使用outputs属性来代表任务的输出
      - 该属性是一个TaskOutputs类型的对象
      - 可以设置文件、文件集、目录、属性等
    - ```
      task hello{
          //def 定义属性的关键字
          def sourceTxt = fileTree("source")
          def dest = file('dist.txt')
          inputs.dir sourceTxt
          outputs.file dest
          println "hello world"
      }
      ```

Gradle插件

- 应用插件就相当于引入了该插件包含的所有任务类型、任务和属性等，使得Gradle可执行插件中预先定好的任务
- ```
  //使用java插件
  apply plugin: 'java'
  //查看构建文件支持的所有任务
  gradle tasks -all
  ```

java插件

- task
  - assemble:装配整个项目
  - build：装配并测试该项目
  - buildDependents：装配并测试该项目，以及该项目的依赖项目
  - classes：装配该项目的所有类
  ```

    - clean：删除构建目录的所有内容。属于Delete类型的任务
    - jar：将项目的所有类打包成JAR包
    - init:执行构建的初始化操作
    - wrapper:生成Gradle包装文件
    - compileJava：编译项目的所有主java源文件
    - processResources：处理该项目的所有主资源
- java插件要求将项目源代码和资源都放在src目录下，将构建生成的字节码文件和资源放在build目录下，插件会自动管理该目录下的两类源代码和资源
    - main：项目的主代码和资源
    - test：项目的测试代码和资源
- 可以在构建文件中通过sourceSets方法改变项目代码、资源的存储路径，第三方或额外依赖的源代码也是通过sourceSets方法配置
    - ```
      sourceSets{
          testlib
      }
      ```
        - 可以将testlib的java源代码放在src/testlib/java目录下，资源放在src/testlib/resources目录下

依赖管理

- 背景：在构建java项目时通常都要依赖一个或多个框架
- 步骤
    1. 为Gradle配置仓库。配置仓库的目的是告诉Gradle到哪里下载JAR包
        - ```
          //定义仓库，在构建文件中
          repositories{
            //使用Maven默认的中央仓库
            mavenCentral()
          }
          ```
    2. 为不同的组配置依赖JAR包。配置依赖JAR包的目的是告诉Gradle要下载哪些JAR包
        - 组：相当于一个文件夹，里面放不同的jar，谁要用就直接复制这个文件夹，而不需要一个一个jar的去复制
            - ```
              configurations{
                group1
              }
              ```
        - 使用dependencies来为依赖组配置JAR包
            - group：JAR所属的组织名
            - name：JAR名称
            - version：JAR版本
            - ```
              dependencies{
                group1 group:'commons-logging',name:'commons-logging',version:'1.2'
                //或者
                group1 'commons-logging:commons-logging:1.2'
              }
              ```

        - java插件提供的组
            - implementation:主项目源代码(src/main/java）所依赖的组。
            - compileOnly:主项目源代码（ src/main/java）编译时才依赖的组。
            - runtimeOnly:主项目源代码(src/main/java）运行时才依赖的组。
            - testImplementation:项目测试代码(src/test/java）所依赖的组。
            - testCompileOnly:项目测试代码（src/test/java）编译时才依赖的组。
            - testRuntimeOnly:项目测试代码（src/test/java）运行时才依赖的组。
            - archives:项目打包时依赖的组。

自定义任务

- 自定义任务就是一个实现Task接口的Groovy类,该接口定义了大量抽象方法需要实现，因此一般自定义任务类会继承DefaultTask基类。自定义任务的Groovy类可以自定义多个方法，这些方法可作为Action使用。
- 使用了@TaskAction修饰的方法将会自动添加为该任务的，反之需要使用doLast或doFirst方法手动将它添加成Action

Android环境搭建

1. 安装Android Studio
2. 安装Android SDK
    - 目录结构
        - build-tools:存放不同版本的 Android项目编译工具。
        - docs:存放Android SDK开发文件和API文档等。
        - emulator:存放Android自带的各种模拟器程序。
        - extras:存放Google提供的USB驱动、Intel提供的硬件加速等附加工具包。
        - licenses:存放 Android的相关授权文档。
        - patcher:存放Android 的一些兼容性补丁。
        - platforms:存放不同版本的Android系统。刚解压缩时该目录为空。
        - platform-tools:存放 Android平台相关工具。
        - skins:存放针对不同Android模拟器的皮肤。
        - sources:存放不同版本的Android系统的源代码。
        - system-images:存放不同Android平台针对不同CPU架构提供的系统镜像。这些系统镜像用于启动、运行 Android模拟器。
        - tools:存放大量的Android开发、调试工具。

第一个Android应用

- Android应用程序建立在应用程序框架之上，所以Android编程就是面向应用程序框架 API编程——这种开发方式与编写普通的 Java或Kotlin应用程序并没有太大的区别，只是Android新增了一些API而已。
- 步骤
    1. 建立一个Android项目或Android模块
    2. 在XML布局文件中定义应用程序的用户界面
        - Android 把用户界面放在XML文档中定义，就可以让XML文档专门负责用户UI设置，而Java程序则专门负责业务实现这样可以**降低程序的耦合性**
        - res/layout存放Android的应用用户界面，类似于html
            - RelativeLayout：它代表一个相对布局
            - UI控件

        - Button:代表一个普通按钮。
        - TextView:代表一个文本框。
            - `android:id`:**指定该控件的唯一标识**，在Java或Kotlin程序中可通过findViewByIld("id")来获取指定的 Android界面组件。
            - `android:layout width`:**指定该界面组件的宽度**。如果该属性值为match parent，则说明该组件与其父容器具有相同的宽度;如果该属性值为 wrap content，则说明该组件的宽度取决于它的内容——能包裹它的内容即可。
            - `android:layout height`:**指定该界面组件的高度**。如果该属性值为match parent，则说明该组件与其父容器具有相同的高度;如果该属性值为 wrap content，则说明该组件的高度取决于它的内容——能包裹它的内容即可。
    3. 在Java或Kotlin代码中编写业务实现
        - main/java存放Android的业务逻辑

Android应用结构分析

- app
    - build:存放该项目的构建结果
    - libs:存放该项目依赖的第三方类库
    - src：存放源代码和资源
        - androidTest：代表了Android测试项目
        - main
            - java（存放Java或Kotlin源文件)
            - res（存放资源）
                - drawable
                - layout
                - mipmap-xxx
                - values
            - AndroidManifest.xml
        - test
    - build.gradle (Gradle构建文件)
    - .gitignore（该文件配置Git工具忽略管理哪些文件)

简述

- lib：Android Studio默认会从国外的中央仓库下载JAR包，如果不能顺畅的连接国外网络，将会报错
- build\generated\source\r\
    - Android Studio自动生成的R.java文件，R.java文件由AAPT工具根据应用中的资源文件自动生成
    - Android应用的资源字典
    - 生成规则
        - 每类资源都对应于R类的一个内部类。比如所有页面布局资源对应于layout内部类
        - 每个具体的资源项都对应于内部类的一个public static final int类型的字段。
        - **类似于散列表**

res

- 对应标准的Gradle程序叫做resources目录
- 存放Android项目的各种资源文件
- layout：存放页面布局文件
- values：存放各种XML格式的资源文件
  - string.xml：字符串资源文件
  - colors.xml:颜色资源文件
  - dimens.xml:尺寸资源文件
  - drawable:存放XML文件定义的Drawable资源
    - drawable-ldpi/mdpi/hdpi/xdpi/xxdpi

流程

1. ```
   //举例在res/value/string.xml定义一个字符串
   <resources>
       <string name = "hello">helloworld</string>
   </resources>
   ```

2. ```
   //在java代码中使用
   R.string.hello
   ```

3. ```
   //在xml中使用
   @string/hello
   ```

   - 注意，如果在XML文件中使用标识符则无须使用专门的资源定义，直接在XML文档中使用`@+id/<标识符代号>`形式即可

     - ```
       //定义
       android:id="@+id/ok"
       //在xml中使用
       @id/ok
       ```

     - ```
       //在Activity中使用
       findViewById(R.id.ok)
       ```

AndroidManifest.xml

- 控制Android应用的名称、图标、访问权限和包名等整体属性，包名将作为应用的唯一标识,
- **Android应用的Activity、Service、ContentProvider和BroadcastReceiver组件都需要在该文件中配置**
- 应用程序兼容的最低版本
- 应用程序使用系统所需的权限声明
  - 一个Android应用可能需要权限才能调用Android系统的功能，因此他也需要声明调用自身所需要的权限
       ```

    - ```
      <uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
      ```
- 其他程序访问该程序所需的权限声明

基本组件和通信

- Android应用通常由一个或多个基本组件组成

Activity

- Activity是Android应用中负责与用户交互的组件，使用setContentView(View)来显示指定组件
- Activity是Window的容器，Activity包含一个getWindow()方法，该方法返回该Activity所包含的窗口
    - 问题：有setWindow方法，设置Activity的可视化用户界面吗？如果没有，为什么要那样设计？
        - Activity没有setWindow方法
- 如果需要多个用户界面，那么Android应用会包含多个activity，多个Activity组成Activity栈，当前活动的Activity位于栈顶

View

- View组件是所有UI控件、容器控件的基类，View组件就是Android应用中用户实实在在看到的部分。
- View组件需要放到容器组件，或者使用Activity将它显示出来

Service

- Service与Activity的地位是并列的，它也代表一个单独的Android组件。
- Service与Activity的区别：Service通常位于后台运行，它一般不需要与用户交互，因此Service组件没有图形用户界面

BroadcastReceiver

- 广播信息接收器
- 类似于时间编程中的事件监听器
- BroadcastReceiver与**普通**事件监听器的区别
    - 普通事件监听器监听的事件源是程序中的对象
    - BroadcastReceiver监听的事件源是Android应用中的其他组件，相当于一个全局的事件监听器
- 使用
    1. 实现BroadcastReceiver子类，重写onReceive（Context context，Intent intent）方法
    2. 其他组件通过sendBroadcast（）、sendStikyBroadcast（）或sendOrderedBroadcast（）方法发送广播信息
    3. 如果该BroadcastReceiver能对广播信息响应（通过IntentFilter配置），该BroadcastReceiver的onReceive（Context context，Intent intent）方法将会被触发
- 注册
    - 在java代码中通过Context.registReceiver()方法注册BroadcastReceiver
    - 在AndroidManifest.xml文件中使用`<receiver.../>`元素完成注册
      ```

ContentProvider

- **多个应用间进行数据交互**
- 当实现ContentProvider时，需要实现的抽象方法
  - 类似于SQL的增删改查
  - inset（Uri，ContentValues）
  - delete（Uri，ContentValues）
  - update（Uri，ContentValues，String，String[]）
  - query（Uri，String[]，String，String[]，String）
- 与之结合使用的是ContentResolver,应用程序使用ContentProvider暴露自己的数据，通过ContentResolver来访问数据
  - 问题：多进程，共享数据，是如何保证事务的ACID和怎么进行同步的？
    - ContentProvider是个单例模式，所有的数据库操作都是用户通过ContentResolver发起请求的，统一由ContentProvider处理。

Intent和IntentFilter

- Android应用内不同组件之间通行的载体
- Intent可以启动应用中另一个Activity，也可以启动一个Service组件，还可以发送一条广播消息来触发系统中的BroadcastReceiver
- Activity、Service、BroadcastReceiver三种组件之间的通信都以Intent作为载体，只是不同组件使用Intent的机制略有不同
- 分类
  - 启动Activity，可调用Context的startActivity（Intent intent）或startActivityForResult（Intent intent，int requestCode），这两个方法中的Intent参数封装了需要启动的目标Activity的信息
  - 启动Service，可调用Context的startService（Intent intent）或bindService（Intent service， ServiceConnection conn，int flags），这两个方法中的Intent参数封装了需要启动的目标Service的信息
  - 启动BroadcastReceiver，可调用Context的sendBroadcast（Intent intent）或sendStikyBroadcast（Intent service）或sendOrderedBroadcast（Intent，String receiverPermission）这两个方法中的Intent参数封装了需要启动的目标Service的信息
- 当一个组件通过Intent表示了启动或触发另一个组件的“意图”，该意图可分为两类
  - 显式Intent：明确指定需要启动或触发的组件的类名
  - 隐式Intent：指定需要启动或触发的组件应满足怎样的条件
    - 判断是否符合隐式Intent
      - 使用IntentFilter来实现
      - 被调用组件通过IntentFilter来声明自己所满足的条件

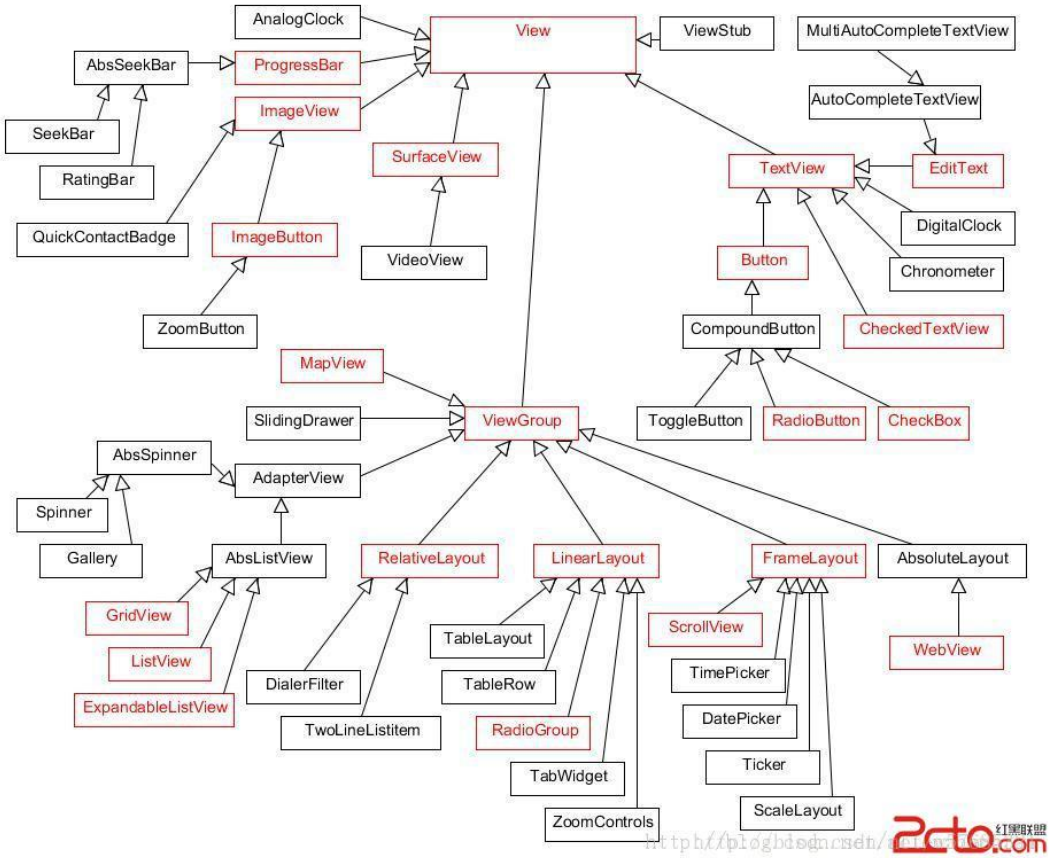
签名APK

- Android项目以它的包名作为唯一标识，如果安装两个包名一致的应用，后安装的将会覆盖前面安装的
- 作用
  - 确定发布者的身份。
  - 确保应用的完整性。

界面编程

视图View

- 位置：Android的所有UI组件都放在android.widget包及其子包、android.view包及其子包中
- 关系：Android应用的所有UI组件都继承了View类
- 问题：view和子类的继承关系是什么样的？



- Android开发API：https://developer.android.com/reference/packages?hl=zh-cn
- 所有组件都提供了两种方式来控制组件的行为。
  - 在XML布局文件中通过XML属性进行控制。
  - 在Java或Kotlin程序代码中通过调用方法进行控制。
- 对于View类而言，它是所有UI组件的基类，因此它包含的XML属性和方法是所有组件都可使用的。

ViewGroup

- ViewGroup继承了View 类，当然也可以当成普通View来使用，但ViewGroup主要还是当成容器类使用。但由于**ViewGroup是一个抽象类**，因此在实际使用中通常总是使用ViewGroup的子类来作为容器，例如各种布局管理器。
- ViewGroup容器控制其子组件的分布依赖于ViewGroup.LayoutParams 、ViewGroup.MarginLayoutParams两个**内部类**。这两个内部类中都提供了一些XML属性，ViewGroup 容器中的子组件可以指定这些XML属性。
- ViewGroup.LayoutParams支持的XML属性

XML属性	说明	值（都支持）	说明
android:layout height	指定该子组件的布局高度	match_parent (早期叫fill parent)	指定子组件的高度、宽度与父容器组件的高度、宽度相同(实际上还要减去填充的空白距离)。
android:layout width	指定该子组件的布局宽度	wrap content	指定子组件的大小恰好能包裹它的内容即可。

  - 注意：Android 组件的大小不仅受它实际的宽度、高度控制，还受它的布局高度与布局宽度控制。比如设置-一个组件的宽度为30pt, 如果将它的布局宽度设为match parent, 那么该组件的宽度将会被“拉宽"到占满它所在的父容器:如果将它的布局宽度设为wrap content, 那么该组件的宽度才会是30pt。

- ViewGroup.MarginLayoutParams用于控制子组件周围的页边距(Margin,也就是组件四周的留白)

XML属性	相关方法	说明
android:layout margin	setMargins(int,int,int,int)	指定该子组件四周的页边距
android:layout marginBottom	setMargins(int, int,int,int)	指定该子组件下边的页边距
android:layout marginEnd	setMargins(int,int,int,int)	指定该子组件结尾处的页边距
android:layout marginHorizontal	setMargins(int,int,int,int)	指定该子组件左、右两边的页边距
android:layout marginLeft	setMargins(int,int,int,int)	指定该子组件左边的页边距

控制UI界面

- XML控制：使用XML布局文件将视图和逻辑分离开
  - 在activity显示
    - `setContentView(R.layout.<资源文件名字>)`
  - 当在布局文件中添加多个UI组件时，都可以为该UI组件指定android:id属性，该属性的属性值代表该组件的唯一标识。
    - `findViewById(R.id. <android.id属性值>)`

- java控制：虽然Android推荐使用XML布局文件来控制UI界面，但如果开发者愿意可以完全抛弃XML布局文件，完全在Java或Kotlin代码中控制UI界面。
  - 无论创建哪种UI组件，都需要传入一个this参数，这是由于创建UI组件时传入一个Context参数，Context 代表访问Android应用环境的全局信息的API。让UI组件持有一个Context参数，可以让这些UI组件通过该Context参数来获取Android应用环境的全局信息。

    - Context本身是一个抽象类，Android应用的Activity 、Service都继承了Context,因此Activity、Service都可直接作为Context使用。
- XML和java控制：当混合使用XML布局文件和代码来控制UI界面时，习惯上把变化小、行为比较固定的组件放在XML布局文件中管理，而那些变化较多、行为控制比较复杂的组件则交给代码来管理。

自定义View

- View组件只是一-个矩形的空白区域，View组件没有任何内容。对于Android应用的其他UI组件来说，它们都继承了View组件，然后在View组件提供的空白区域绘制外观。
- 基于Android UI组件的实现原理，开发者完全可以开发出项目定制的组件——当 Android系统提供的UI组件不足以满足项目需要时，开发者可以通过继承View来派生自定义组件。
- 方法：当开发者打算派生自己的UI组件时，首先要定义1个继承View基类的子类，然后重写View类的一个或多个方法。
    -

方法	说明
构造器	重写构造器是定制View的最基本方式，当Java或Kotlin代码创建一个View实例，或根据XML布局文件加载并构建界面时将，需要调用该构造器。
onFinishInflate()	这是-一个回调方法，当应用从XML布局文件加载该组件并利用它来构建界面之后，该方法将会被回调。
onMeasure(int,int)	调用该方法来检测View组件及其所包含的所有子组件的大小。
onLayout(boolean, int, int, int, int)	当该组件需要分配其子组件的位置、大小时，该方法就会被回调。
onDraw(Canvas)	当该组件将要绘制它的内容时回调该方法。
onSizeChanged(int, int, int, int)	当该组件的大小被改变时回调该方法。
onKeyDown(int, KeyEvent)	当某个键被按下时触发该方法。

- 注意：
    - 当需要开发自定义View时，开发者并不需要重写上面列出的所有方法，而是可以根据业务需要重写上面的部分方法。
    - 如果要新建的自定义组件只是组合现有组件，不需要重新绘制所有组件内容，只要实现自 定义组件的构造器,在自定义构造器中使用LayoutInflater加载布局文件即可。

布局管理器LayoutManager

- 背景：不同手机屏幕的分辨率、尺寸开不完全相同，如果让程序手动控制每个组件的大小、位置，则将给编程带来巨大的困难。
- 目的：让组件在不同的手机屏幕上都能运行良好
- 作用：布局管理器可以根据运行平台来调整组件的大小
- Android的布局管理器本身就是一个UI组件，所有的布局管理器都是ViewGroup的子类。
    - 所有布局都可作为容器类使用，因此可以调用多个重载的addView()向布局管理器中添加组件。

- 可以将一个布局管理器嵌套到其他布局管理器中，因为布局管理器也继承了View,也可以作为普通UI组件使用。

线性布局

- 线性布局由LinearLayout类来代表，它们都会将容器里的组件**一个挨着一个排列**起来。LinearLayout可以控制各组件横向排列(通过设置android:orientation属性控制)，也可控制各组件纵向排列。
- Android的线性布局**不会换行**，当组件一个挨着一个排列到头之后，**剩下的组件将不会被显示出来。**
-

XML属性	相关方法	说明
android:baselineAligned	setBaselineAligned(boolean)	该属性设为false, 将会阻止该布局管理器与它的子元素的基线对齐
android:divider	setDividerDrawable(Drawable)	设置垂直布局时两个按钮之间的分隔条
android:gravity	setGravity(int)	设置布局管理器内组件的对齐方式。该属性支持top、bottom、left、 right、center_ vertical、fill_ vertical center_ horizontal、 fill *horizontal*、*center*、*fill*、 *clip* vertical、 clip_ horizontal 几个属性值。也可以同时指定多种对齐方式的组合，例如left|center_vertical 代表出现在屏幕左边，而且垂直居中
android:measureWithlargestChild	setMeasureWithLargestChildEnabled(boolean)	当该属性设为true时，所有带权重的子元素都会具有最大子元素的最小尺寸
android:orientation	setOrientation(int)	设置布局管理器内组件的排列方式，可以设置为horizontal (水平排列)、vertical (垂直排列，默认值)两个值的其中之一
android:weightSum		设置该布局管理器的最大权值和

- LinearLayout包含的所有子元素都受LinearLayout.LayoutParams 控制，因此LinearLayout包含的**子元素**可以额外指定属性。
  -

XML属性	说明
android:layout_gravity	指定该子元素在LinearLayout中的对齐方式
android:layout_weight	指定该子元素在LinearLayout中所占的权重

表格布局

- 表格布局由TableLayout所代表，**TableLayout 继承了LinearLayout**,因此它的本质依然是线性布局管理器。
- 表格布局**采用行、列的形式**来管理UI组件，TableLayout并不需要明确地声明包含多少行、多少列，而是通过添加TableRow、其他组件来控制表格的行数和列数。

- 每次向TableLayout中添加一个TableRow,该TableRow就是一个表格行，TableRow也是容器，因此它也可以不断地添加其他组件，每添加一个子组件该表格就增加一列。
- 如果直接向TableLayout中染加组件，那么这个组件将直接占用一行。
- 在表格布局中，列的宽度由该列中最宽的那个单元格决定，整个表格布局的宽度则取决于父容器的宽度(默认总是占满父容器本身)。
- 单元格行为
    - Shrinkable: 如果某个列被设为Shrinkable, 那么该列的所有单元格的宽度可以被收缩，以保证该表格能适应父容器的宽度。
    - Stretchable: 如果某个列被设为Stretchable,那么该列的所有单元格的宽度可以被拉伸，以保证组件能完全填满表格空余空间。
    - Collapsed: 如果某个列被设为Collapsed,那么该列的所有单元格会被隐藏。
- TableLayout继承了LinearLayout,因此它完全可以支持LinearLayout所支持的全部XML属性。除此之外，TableL ayout还有自己独特的方法
XML属性	相关方法	说明
android:collapseColumns	setColumnCollapsed(int,boolean)	设置需要被隐藏的列的列序号。多个列序号之间用逗号隔开，从0开始
android:shrinkColumns	setShrinkllColumns(boolean)	设置允许被收缩的列的列序号。多个列序号之间用逗号隔开，从0开始
android:stretchColumns	setStretchAllColumns(boolean)	设置允许被拉伸的列的列序号。多个列序号之间用逗号隔开，从0开始

帧布局

- 帧布局由FrameLayout所代表，**FrameLayout 直接继承了ViewGroup** 组件。
- 帧布局容器为每个加入其中的组件都创建一个空白的区域(称为一帧), 每个子组件占据1帧，这些帧都会根据gravity 属性执行自动对齐。帧布局的效果有点类似于AWT编程的CardLayout,都是**把组件一一个个叠加在一起。**与CardLayout的区别在于，CardLayout可以将下面的Card移上来，但FrameLayout则没有提供相应的方法。

XML属性	相关方法	说明
android:foregroundGravity	setForegroundGravity(int)	定义绘制前景图像的gravity 属性
android:measureAllChildren	setMeasureAllChildren(boolean)	设置该布局管理器计算大小时是否考虑所有子组件，默认为false,即只考虑可见状态的组件
- FrameLayout包含的子元素也受FrameLayout.LayoutParams控制，因此它所包含的子元素也可指定android:layout_gravity 属性，该属性控制该子元素在FrameLayout中的对齐方式。
- Android的View和UI组件不是线程安全的，所以Android不允许开发者启动线程访问用户界面的UI组件。

绝对布局

- 绝对布局由**AbsoluteLayout**所代表。**AbsoluteLayout继承自LinearLayout**
- Android**不提供任何布局控制**，而是由开发人员自己通过X、Y坐标来控制组件的位置。当使用AbsoluteLayout作为布局容器时，布局容器不再管理子组件的位置、大小一这 些都需要开发人员自己控制。
- 注意：使用绝对布局会很难兼顾不同屏幕大小、分辨率的问题。因此，AbsoluteLayout 布局管理器已经过时。
XML属性	说明
android:layout_x	指定该子组件的X坐标。
android:layout_y	指定该子组件的Y坐标。

  - Android常用的距离单位。
    - px (像素) :每个px对应屏幕上的一个点。
    - dip或dp (device independent pixels,设备独立像素) :-种基于屏幕密度的抽象单位。在每英寸160点的显示器上，ldip= lpx。但随着屏幕密度的改变，dip与px的换算会发生改变
    - sp (scaled pixels,比例像素) :主要处理字体的大小，可以根据用户的字体大小首选项进行缩放。
    - in (英寸) :标准长度单位。
    - mm(毫米):标准长度单位。
    - pt (磅) :标准长度单位，1/72英寸。

约束布局

- 从功能上讲，约束布局相当于相对布局的改进
- 背景：相对布局只能控制组件在父容器中居中，或者与父容器(或另一个组件)左对齐、右对齐、顶端对齐、底端对齐，或者控制组件在另一个组件的左边、右边、上方或下方。但如果要控制组件位于父容器的某个百分比处，或者控制组件位于另一个组件的左边25dp处，相对布局就无能为力了，此时就可借助于约束布局。
- 使用Android Studio 新建项目时添加的Actvitiy 的默认布局文件自动使用约束布局
- 约束布局中的组件的layout_width、layout_height属性值

1. wrap content: 该组件的宽度或高度刚够显示组件内容。
2. 固定长度值:设置该组件的宽度或高度为固定值。
3. match constraint (0dp) :该组件的宽度或高度完全按约束计算。

- 设置垂直方向的位置的百分比为5%
    - `app: layout_ constraintVertical_bias="0.05"`
-

XML属性	说明
app:layout constraintXxx _toXxxOf	设置该组件与父容器(或其他组件)对齐。其中Xxx可以是Start、 End、 Top、 Bottom 等值，具体可根据实际情况设置。该属性的属性值可以是parent (代表父容器)或其他组件的ID。
android:layout_ marginXxx	设置该组件与参照组件的间距。其中Xxx可以是Start、End、Top、Bottom等值，具体可根据实际情况设置。
app:layout constraintXxx_ bias	设置该组件在参照组件中的百分比。其中Xxx可以是Horizontal或Vertical。

- 约束布局还提供了两种自动创建约束的方式。
    - 自动连接(Autoconnect) :。自动连接默认是关闭的，可以通过单击约束布局设计器上方的Autoconnect图标来打开自动连接。
        - 如果打开了约束布局的自动连接，那么每次向约束布局中拖入新组件时，约束布局设计器都会自动为该组件添加上、下、左、右4个约束一一并不保证所添加的约束符合开发者的期望，因此通过自动连接添加的约束通常都需要手动修改。
    - 推断(Infer) :可以通过单击约束布局设计器上方的Infer图标执行推断。
        - “推断”功能则是另一种用法:用户先向界面中添加所有的UI组件，再通过拖曳摆放这些组件的位置，然后单击约束布局设计器上方的Infer 图标执行推断，约束布局设计器会自动为所有组件添加上、下、左、右4个约束一依然不保证所添 加的约束符合开发者的期望。

引导线

- 背景：有时候，一批组件都需要放在容器的某个位置，或者某个组件需要参照的组件并不存在，为此约束布局为之提供了引导线(GuideLine)。
- 介绍：引导线是约束布局中不可见的、却实实在在存在的组件，引导线唯一的作用就是被其他组件作为定位的参照。
- 种类:水平引导线和垂直引导线。
- 引导线对应于Guideline元素，属性

XML属性	说明
android:orientation	该属性指定引导线的方向。该属性支持vertical和horizontal两个值，代表垂直引导线和水平引导线。
app:layout_constraintGuide_begin	该属性指定引导线到父容器起始处的距离。该属性值可以是一个距离值。
app:layout_constraintGuide_end	该属性指定引导线到父容器结束处的距离。该属性值可以是一个距离值。
app:layout_*constraintGuide*_percent	该属性指定引导线位于父容器的指定百分比处。

- 注意：后3个属性的作用是相同的，只是代表引导线的3种不同的定位方式，它们都用于控制引导线的位置，因此只要指定其中之一即可。

分界线

- 约束布局还提供了分界线(Barrier) 来定位其他组件，分界线同样是不可见的、但实实在在存在的组件。
- 分界线同样可分为水平分界线和垂直分界线两种

引导线和分界线的区别

- 引导线以父容器为基础进行定位，比如设置与父容器的起始处、结束处的距离来控制引导线的定位，也可通过设置位于父容器的百分比来控制引导线的定位;
- 分界线根据容器中的组件进行定位，分界线可位于多个组件的上方、下方、左边或右边。

TextView及其子类

文本框(TextView) 和编辑框(EditText)

- TextView直接继承了View,它还是EditText、Button 两个UI组件类的父类。
- TextView 的作用就是在界面上显示文本
- TextView其实就是一一个文本编辑器，只是Android关闭了它的文字编辑功能。
  - 如果开发者想要定义一个可编辑内容的文本框，则可以使用它的子类: EditText, EditText 允许用户编辑文本框中的内容。
  - TextView还派生了一个CheckedTextView, CheckedTextView 增加了一个checked状态，开发者可通过setChecked(boolean)和isChecked()方法来改变、访问该组件的checked状态。该组件还可通过setCheckMarkDrawable0方法来设置它的勾选图标。
  - TextView 还派生出了Button类。

EditText

- EditText与TextView 非常相似，它甚至与TextView共用了绝大部分XML属性和方法。
- EditText与TextView的最大区别在于: EditText 可以接受用户输入。
- EditText组件最重要的属性是inputType,该属性相当于HTML的<input..>元素的type属性，用于将EditText设置为指定类型的输入组件。
- 子类

    - AutoCompleteTextView:带有自动完成功能的EditText
    - ExtractEditText: 它并不是UI组件，而是EditText组件的底层服务类，负责提供全屏输入法支持。

按钮(Button) 组件

- Button继承了TextView,它主要是在UI界面上生成一个按钮，该按钮可以供用户单击，当用户单击按钮时，按钮会触发一个onClick事件。

9patch图片做背景

- 背景：当按钮的内容太多时，Android会自动缩放整张图片，以保证背景图片能覆盖整个按钮。但这种缩放整张图片的效果可能并不好。
- 作用：可能存在的情况是我们只想缩放图片中某个部分，这样才能保证按钮的视觉效果。比如聊天框

单选钮(RadioButton) 和复选框(CheckBox)

- 单选钮(RadioButton)、复选框(CheckBox)是用户界面中最普通的UI组件，它们都**继承了Button类**，因此都可直接使用Button支持的各种属
  性和方法。
- RadioButton、CheckBox与普通按钮的不同：它们多了一个可选中的功能，因此RadioButton、CheckBox都可额外指定一个android:checked属性，该属性用于指定RadioButton、CheckBox 初始时是否被选中。
- RadioButton与CheckBox 的不同：一组RadioButton 只能选中其中一个，因此RadioButton通常要与RadioGroup一起使用，用于定义一组单选钮。
- 委托式事件处理机制的原理：当事件源上发生事件时，该事件将会激发该事件源上的监听器的特定方法。
- 注意：如果在XML布局文件中默认选中了某个单选钮，则必须为该组单选钮的每个按钮都指定android:id 属性值;否则，这组单选钮不能正常工作。

状态开关按钮(ToggleButton) 和开关(Switch)

- 状态开关按钮(ToggleButton) 和开关(Switch)也是由Button派生出来的，因此它们的本质也是按钮，Button 支持的各种属性、方法也适用于ToggleButton 和Switch。
-

	CheckBox	**ToggleButton、Switch**
相同	都可以提供两种状态	都可以提供两种状态
不同	只是自身是否被选中	切换程序的状态

时钟(AnalogClock 和TextClock)

- TextClock 本身就继承了TextView也就是说， 它本身就是文本框，只是它里面显示的内容总是当前时间。
    - 与TextView不同的是，为TextClock设置android:text属性没什么作用。
    - TextClock 能以24小时制或12小时制来显示时间，而且可以由程序员来指定时间格式。
- AnalogClock则继承了View 组件，它重写了View的OnDraw(方法，它会在View上绘制模拟时钟。
- TextClock和AnalogClock的不同
    - TextClock 显示数字时钟，可以显示当前的秒数;
    - AnalogClock 显示模拟时钟，不会显示当前的秒数。

计时器(Chronometer)

- Chronometer组件与TextClock都继承自TextView,因此它们都会显示一段文本。但Chronometer并不显示当前时间，它显示的是从某个起始时间开始，一共过去了多长时间。

ImageView 及其子类

- ImageView继承自View组件，不仅能显示图片，任何Drawable 对象都可使用ImageView来显示。
- ImageView 派生了ImageButton、ZoomButton、 FloatingActionButton 等组件
- 能够设置图片的最大高度，最大长度，缩放，src路径
- **ImageButton**:图片按钮。
    - Button与ImageButton的区别：Button 生成的按钮上显示文字，而ImageButton上则显示图片。
    - 为ImageButton 按钮指定android:text 属性没用(ImageButton 的本质是ImageView),即使指定了该属性，图片按钮上也不会显示任何文字。
    - 使用ImageButton,图片按钮可以指定android:src属性，该属性既可使用静止的图片，也可使用自定义的Drawable对象，这样即可开发出随用户动作改变图片的按钮。
- **QuickContactBadge**:显示关联到特定联系人的图片。
- **ZoomButton**：ZoomButton 可以代表“放大”和“缩小”两个按钮。
    - ZoomButton的行为基本类似于ImageButton,只是Android默认提供了btn minus、 btn plus两个Drawable资源，只要为ZoomButton的android:src 属性分别指定btn minus、 btn plus,即可实现“缩小”和“放大”按钮。
    - Android还提供了一个ZoomControls组件，该组件相当于同时组合了“放大”和“缩小”两个按钮，并允许分别为两个按钮绑定不同的事件监听器。
- **FloatingActionButton**：它代表一个 悬浮按钮。该按钮默认是 一个带默认填充色的圆形按钮，当用户单击该按钮时，该按钮可以显示-一个波纹效果。
    - FloatingActionButton 位于support库中，因此必须先添加然后才能使用。可以直接修改对应模块的build. gradle文件来添加FloatingActionButton

AdapterView及其子类

- AdapterView 本身是一个抽象基类，它派生的子类在用法上十分相似，只是显示界面有一定的区别
- 特征
    - AdapterView 继承了ViewGroup, 它的本质是**容器。**
    - AdapterView可以包括多个“列表项”，并将多个“列表项”以合适的形式显示出来。
    - AdapterView显示的多个“列表项”由Adapter提供。调用AdapterView的setAdapter(Adapter)方法设置Adapter即可。

Adapter接口及其实现类

- Adapter本身只是一个接口，它派生了ListAdapter 和SpinnerAdapter 两个子接口，其中ListAdapter为AbsListView**提供列表项**，而SpinnerAdapter为AbsSpinner提供列表项。几乎所有的Adapter都继承了BaseAdapter, 而BaseAdapter同时实现了ListAdapter、SpinnerAdapter两个接口，因此BaseAdapter及其子类可以同时为AbsListView、AbsSpinner 提供列表项。
-

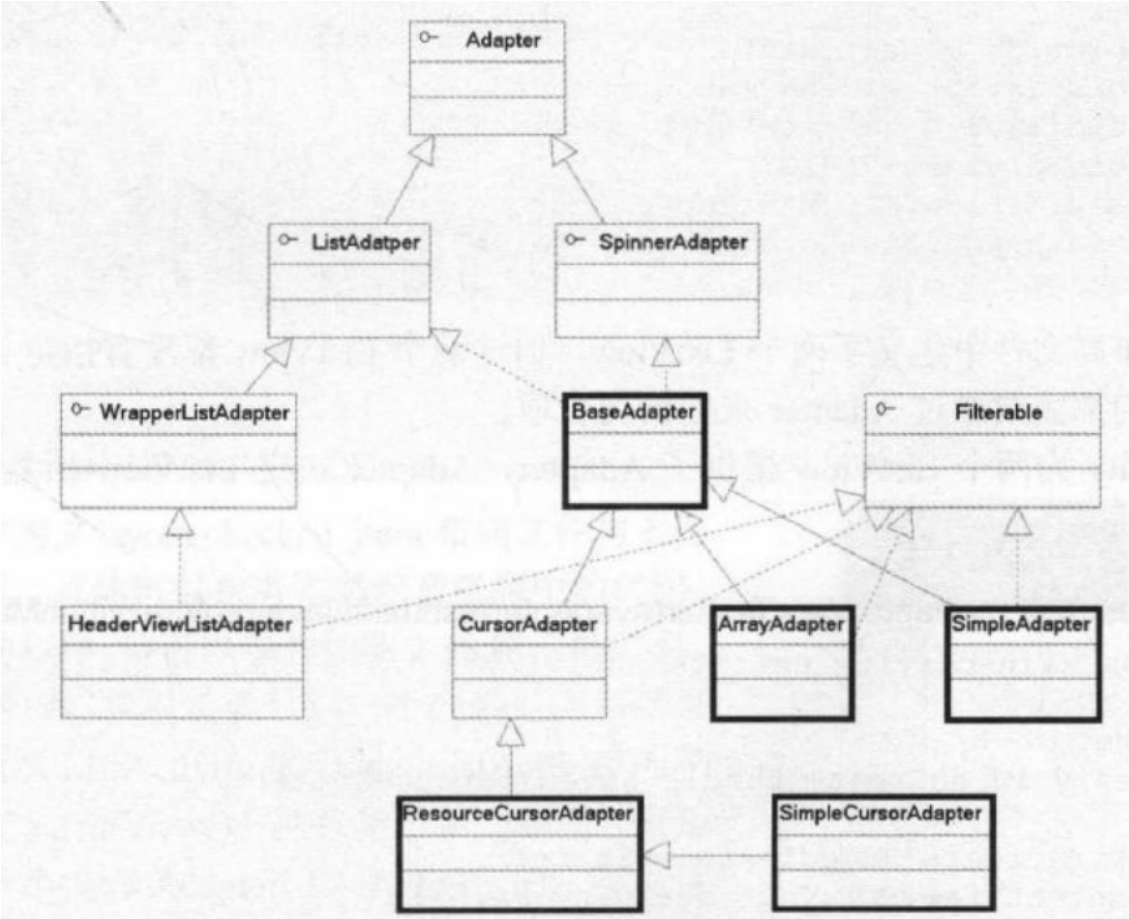


图 2.37　Adapter 接口及其实现类的继承关系图

- 常用的实现类
  - **ArrayAdapter**: 通常用于将数组或List集合的多个值包装成多个列表项。
    - new ArrayAdapter（ ）时必须指定如下三个参数。
      1. Context: 这个参数无须多说，它代表了访问整个Android应用的接口。几乎创建所有组件
         都需要传入Context对象。
      2. textViewResourceId:一个资源 ID,该资源ID代表一个TextView,该TextView组件将作为ArrayAdapter的列表项组件。
      3. 数组或List:该数组或List将负责为多个列表项提供数据。
    - 如果程序的窗口仅仅需要显示一个列表，则可以直接让Activity继承ListActivity 来实现，ListActivity 的子类无须调用setContentView()方法来显示某个界面，而是可以直接传入一个内容Adapter, ListActivity的子类就呈现出一个列表。
  - **SimpleAdapter**: 可用于将List集合的多个对象包装成多个列表项。
    - new SimpleAdapter（ ）时必须指定如下五个参数
      1. Context
      2. 该参数应该是一个`List<? extends Map<String, ?>>`类型的集合对象，该集合中每个`Map<String, ?>`对象生成一一个列表项。
      3. 该参数指定一个界面布局的ID。
      4. 该参数应该是一个String[]类型的参数，该参数决定提取`Map<String, ?>`对象中哪些key对应的value来生成列表项。
      5. 该参数应该是一个int[]类型的参数，该参数决定填充哪些组件。
    - 问题：提供的列表项的value怎么和组件相对应？
      - 猜测按两个数组对应的下标进行匹配（待验证）

        - **SimpleCursorAdapter**: 与SimpleAdapter基本相似，只是用于包装Cursor提供的数据。
        - **BaseAdapter**: 通常用于被扩展。扩展BaseAdapter可以对各列表项进行最大限度的定制。

自动完成文本框(AutoComplete TextView)

- 自动完成文本框(AutoCompleteTextView)**从EditText派生**而出，实际上它也是一个文本编辑框，但它比普通编辑框多了一个功能:当用户输入一定字符之后，自动完成文本框会显示一个下拉菜单，供用户从中选择，当用户选择某个菜单项之后，AutoCompleteTextView按用户选择自动填写该文本框。
    - 使用：setAdapter（ ）只要为它设置-一个Adapter即可，该Adapter 封装了AutoCompleteTextView预设的提示文本。
- AutoCompleteTextView还派生了一个子类: MultiAutoCompleteTextView, MultiAutoCompleteTextView允许输入多个提示项，多个提
示项以分隔符分隔。MultiAutoCompleteTextView 提供了setTokenizer( ) 方法来设置分隔符。

可展开的列表组件(ExpandableListView)

- ExpandableListView是ListView 的子类，它在普通ListView 的基础上进行了扩展，它把应用中的列表项分为几组，每组里又可包含多个列表项。
    - ExpandableListView的用法与普通ListView的用法非常相似，只是ExpandableListView所显示的列表项应该由ExpandableListAdapter提供。ExpandableListAdapter也是一一个接口
    - 实现ExpandableListAdapter
        - 扩展BaseExpandableListAdapter实现ExpandableListAdapter.
        - 使用SimpleExpandableListAdapter将两个List集合包装成ExpandableListAdapter.
        - 使用SimpleCursorTeeAdapter将Cursor中的数据包装成SimpleCursorTreeAdapter.

Spinner

- 弹出(下拉)一个菜单供用户选择。
- Spinner与Gallery都继承了AbsSpinner, AbsSpinner 继承了AdapterView, 因此它也表现出AdapterView的特征:只要为AdapterView提供Adapter即可。Gallery和Spinner都是一个列表选择框。
    - Spinner与Gallery区别
        1. Spinner 显示的是一个垂直的列表选择框;而Gallery显示的是一个水平的列表选择框。
        2. Spinner 的作用是供用户选择;而Gallery则允许用户通过拖动来查看上一个、下一个列表项。
- Spinner与ListView的区别：Spinner可以以弹出或下拉方式显示列表选择框。

AdapterViewFlipper

- AdapterViewFilpper继承了AdapterViewAnimator, 它也会显示Adapter提供的多个View组件，但它**每次只能显示一个View组件**，程序可通过showPrevious()和showNext()方法控制该组件显示上一个、下一个组件。
- AdapterViewFilpper可以在多个View 切换过程中使用渐隐渐显的动画效果。除此之外，还可以调用该组件的startFlipping()控制它“自动播放”下一个View组件。

StackView

- StackView也是AdapterViewAnimator的子类，它也用于显示Adapter 提供的一系列View。StackView将会以“堆叠(Stack)"的方式来显示多个列表项。
- 控制方式
    - 拖走StackView中处于顶端的View, 下一个View 将会显示出来。将上一个View 拖进StackView,将使之显示出来。
    - 通过调用StackView的showNext()、showPrevious()控制显示下一个、 上一个组件。

RecyclerView

- RecyclerView的用法与ListView相似，同样使用Adapter来生成列表项
- RecyclerView需要使用改进的RecyclerView.Adapter,改进后RecyclerView.Adapter 只要实现三个方法。
    - onCreateViewHolder(ViewGroup viewGroup, int i):该方法用于创建列表项组件。使用该方法所创建的组件会被自动缓存。
    - onBindViewHolder(ViewHolder viewHolder, int i):该方法负责为列表项组件绑定数据，每次组件重新显示出来时都会重新执行该方法。
    - gettemCount():该方法的返回值决定包含多少个列表项。

ProgressBar及其子类

- ProgressBar 本身代表了进度条组件，它还派生了两个常用的组件: SeekBar 和RatingBar。ProgressBar 及其子类在用法上十分相似，只是显示界面有一定的区别

进度条(ProgressBar)

- 通常用于向用户显示某个耗时操作完成的百分比。进度条可以动态地显示进度，因此避免长时间地执行某个耗时操作时，让用户感觉程序失去了响应，从而更好地提高用户界面的友好性。
- 通过style为ProgressBar指定风格，可以使用水平和环形进度条

拖动条(SeekBar)

- 拖动条和进度条的区别：进度条采用颜色填充来表明进度完成的程度，而拖动条则通过滑块的位置来标识数值一而且拖动条允许用户拖动滑块来改变值
- 使用场景：对系统的某种数值进行调节，比如调节音量等。
- 改变滑块的属性
    - android:thumb:指定一个Drawable对象，该对象将作为自定义滑块。
    - android:tickMark:指定一个Drawable对象，该对象将作为自定义刻度图标。

星级评分条(RatingBar)

- 星级评分条与拖动条的用法、功能都十分接近:它们都允许用户通过拖动来改变进度。
- RatingBar 与SeekBar的区别: RatingBar 通过星星来表示进度。

ViewAnimator及其子类

- ViewAnimator 是一个基类，它继承了FrameLayout,因此它表现出FrameLayout的特征，可以将多个View组件“叠”在一起。ViewAnimator可以在View切换时表现出动画效果。
    - **FrameLayout**
        - **ViewAnimator**

        - **ViewSwitcher**：当程序控制从一个View切换到另一一个View时，ViewSwitcher支持指定动画效果。
            - 使用：为了给ViewSwitcher添加多个组件，通过调用ViewSwitcher 的setFactory(ViewSwitcher.ViewFactory)方法为之设置ViewFactory,并由该ViewFactory为之创建View即可。
            - **ImageSwitcher**
                - ImageSwitcher继承了ViewSwitcher, 并重写了ViewSwitcher 的showNext()、showPreviousQ方法，因此ImageSwitcher使用起来更加简单。
                - 使用：
                    1. 为ImageSwitcher提供一个ViewFactory,该ViewFactory生成的View组件必须是**ImageView**。
                    2. 需要切换图片时，只要调用ImageSwitcher 的setImageDrawable(Drawable drawable)、setImageResource(int resid)和setImageURI(Uri uri)方法更换图片即可。
            - **TextSwitcher**
                - 与ImageSwitcher相似，使用TextSwitcher也需要设置一个ViewFactory.但是TextSwitcher 所需的ViewFactory 的makeView0方法必须返回一个**TextView**组件。
    - **ViewFlipper**
        - ViewFlipper与前面介绍的AdapterViewFlipper 有较大的相似性，它们可以控制组件切换的动画效果。
        - ViewFlipper和AdapterViewFlipper 的区别：ViewFlipper 需要开发者通过addView(View v)添 加多个View, 而AdapterViewFlipper则只要传入一个Adapter, Adapter 将会负责提供多个View。
        - ViewFlipper可以指定与AdapterViewFlipper相同的XML属性。

各种杂项组件

消息（Toast）组件

- Toast是一种非常方便的提示消息框，它会在程序界面上显示一个简单的提示信息。
    - 特点
        - Toast提示信息不会获得焦点。
        - Toast提示信息过一段时间会自动消失。
    - 流程
        1. 调用Toast的构造器或makeText()静态方法创建一个Toast对象。
        2. 调用Toast的方法来设置该消息提示的对齐方式、页边距等。
        3. 调用Toast的show0方法将它显示出来。

日历视图(CalendarView) 组件

- 日历视图(CalendarView)可用于显示和选择日期，用户既可选择一个日期，也可通过触摸来滚动日历。
- 监听：如果希望监控该组件的日期改变，则可调用CalendarView的setOnDateChangeListener()方法为此组件的点击事件添加事件监听器。

日期、时间选择器(DatePicker 和TimePicker)

- DatePicker和TimePicker都从FrameLayout 派生而来，其中DatePicker供用户选择日期;而TimePicker则供用户选择时间。
- DatePicker和TimePicker在FrameLayout的基础上提供了一些方法来获取当前用户所选择的日期、时间
- 监听：如果程序需要获取用户选择的日期、时间，则可通过为DatePicker 添加OnDateChangedListener进行监听、为TimePicker添加OnTimerChangedListener进行监听

数值选择器(NumberPicker)

- 数值选择器用于让用户输入数值，用户既可以通过键盘输入数值，也可以通过拖动来选择数值。
方法	说明
setMinValue(int minVal)	设置该组件支持的最小值
setMaxValue(int maxVal)	设置该组件支持的最大值
setValue(int value)	设置该组件的当前值

搜索框(SearchView)

- SearchView是搜索框组件，它可以让用户在文本框内输入文字，并允许通过监听器监控用户输入，当用户输入完成后提交搜索时，也可通过监听器执行实际的搜索。
方法	说明
setlconifiedByDefault (boolean iconified)	设置该搜索框默认是否自动缩小为图标。
setSubmitButtonEnabled(boolean enabled)	设置是否显示搜索按钮。
setQueryHint(CharSequence hint)	设置搜索框内默认显示的提示文本。
setOnQueryTextListener (SearchView .OnQueryTextListener listener)	为该搜索框设置事件监听器。
- 提供结果：如果为SearchView增加一个配套的ListView,则可以为SearchView提供搜索的结果

滚动视图(ScrollView)

- SrollView**由FrameLayout派生**而出，用于为普通组件添加滚动条的组件。ScrollView里**最多只能包含一个组件**，而ScrollView的作用就是**为该组件添加垂直滚动条**。
- **HorizontalScrollView**：**为组件添加水平滚动条**
- 水平滚动条和垂直滚动条可以同时使用

通知Channel

- Notification是显示在手机状态栏的通知。Notification 所代表的是一种具有全局效果的通知，程序一般通过NotificationManager服务来发送Notification。
- Notification.Builder 类，可以轻松地创建Notification对象。常用方法：

方法	说明
setDefaults()	设置通知LED灯、音乐、振动等。
setAutoCancel()	设置点击通知后，状态栏自动删除通知。
setContentTitle()	设置通知标题。
setContentText()	设置通知内容。
setSmallIcon()	为通知设置图标。
setLargeIcon( )	为通知设置大图标。
setTick()	设置通知在状态栏的提示文本。
setContentIntent()	设置点击通知后将要启动的程序组件对应的PendingIntent。

- Android 8加入了通知Channel帮助用户来统一管理通知，开发者可以为不同类型的通知创建同一个通知Channel,而用户则可通过该Channel统一管 理这些通知的行为一所有使用同一个Channel的通知都具有相同的行为。
  - 重要性、闪光灯、声音、震动、锁屏、免打扰
- Android 9增强了通知参与者的支持和消息支持更丰富的数据
- 使用流程
  1. 调用getSystemService(NOTIFICATION_ SERVICE)方 法获取系统的NotificationManager 服务。
  2. 创建NotificationChannel对象，并在NotificationManager 上创建该Channel对象。
  3. 通过构造器创建一个Notification. Builder对象。
  4. 为Notification.Builder设置通知的各种属性。
  5. 创建MessagingStyle和Message,通过Message设置消息内容，为Notification. Builder设置MessagingStyle后创建Notification。
  6. 通过NotificationManager发送Notification。

对话框Dialog

- 分类
  - AlertDialog:功能最丰富、实际应用最广的对话框。
    - ProgressDialog:进度对话框，这种对话框只是对进度条的包装。
    - DatePickerDialog:日期选择对话框，这种对话框只是对DatePicker的包装。
    - TimePickerDialog:时间选择对话框，这种对话框只是对TimePicker的包装。

AlertDialog

- 分区
  - 标题
    - 图标区
    - 文字区
  - 内容区
  - 按钮区
    - 取消
    - 确认
- 使用流程

1. 创建AlertDialog. Builder对象。
2. 调用AlertDialog. Builder的setTitle()或 setCustomTitle()方法设置标题。
3. 调用AlertDialog Builder的setIcon()方法设置图标。
4. 调用AlertDialog Builder的相关设置方法设置对话框内容。
   -

方法	说明
setMessage()	设置对话内容为简单文本。
setItems()	设置对话框内容为简单列表项。
setSingleChoiceItems()	设置对话框内容为单选列表项。
setMultiChoiceItems()	设置对话框内容为多选列表项。
setAdapter()	设置对话框内容为自定义列表项。
setView()	设置对话框内容为自定义View。

5. 调用AlertDialog.Builder 的setPositiveButton()、setNegativeButton()或 setNeutralButton()方法添加多个按钮。
6. 调用AlertDialog. Builder的create()方法创建AlertDialog 对象，再调用AlertDialog 对象的show()方法将该对话框显示出来。

对话框风格的窗口

- ```
  <!-在AndroidManifest.xml文件中指定该窗口以对话框风格显示->
  <activity android:name=".MainActivity" android:
  theme="@android:style/Theme.Material.Dialog"></activity>
  ```

PopupWindow

- PopupWindow可以创建类似对话框风格的窗口
- 使用流程
  1. 调用 PopupWindow的构造器创建PopupWindow对象。
  2. 调用PopupWindow的showAsDropDown(View v)将PopupWindow作为V组件的下拉组件显示出来;或调用PopupWindow的showAtLocation()方法将PopupWindow在指定位置显示出来。

DatePickerDialog、TimePickerDialog

- 类似于picker，只是用对话框显示出来
- 使用流程
  1. 通过构造器创建DatePickerDialog、TimePickerDialog实例，调用它们的show0方法即可将日期选择对话框、时间选择对话框显示出来。
  2. 为DatePickerDialog 、TimePickerDialog 绑定监听器，这样可以保证用户通过DatePickerDialog、TimePickerDialog 选择日期、时间时触发监听器，从而通过监听器来获取用户所选择的日期、时间。
- 注意：选择日期对话框、选择时间对话框只是供用户来选择日期、时间的，对Android的系统日期、时间没有任何影响。
  ```

ProgressDialog

- ProgressDialog代表了进度对话框，程序只要创建ProgressDialog 实例，并将它显示出来就是一个进度对话框。（已被Android 8 弃用，deprecated）

菜单Menu

- 关系
  - Menu：选项菜单不支持勾选标记，并且只显示菜单的"浓缩(condensed) "标题。add()方法用于添加菜单项; addSubMenu()用于添加子菜单。
    - SubMenu:它代表 一个子菜单。可以包含1~N个MenuItem (形成菜单项)。不支持菜单项图标，不支持嵌套子菜单。可以设置菜单头的图标和标题，用view设置菜单头
    - ContextMenu:它代表一个上下文菜单。可以包含1~N个MenuItem (形成菜单项)。不支持菜单快捷键和图标。
- 添加子菜单和菜单
  1. 重写Activity的onCreateOptionsMenu(Menu menu)方法，在该方法里调用Menu对象的方法来添加菜单项或子菜单。
     - 如果希望所创建的菜单项是单选菜单项或多选菜单项，则可以调用MenuItem的setCheckable(boolean checkable)来设置该菜单项是否可以被勾选。默认是多选菜单项。
     - 如果希望将一组菜单里的菜单项都设为可勾选的菜单项使用setGroupCheckable(int group, boolean checkable, boolean exclusive)来设置group组里的所有菜单项是否可勾选:如果将exclusive 设为true,那么它们将是一组单选菜单项。
  2. 如果希望应用程序能响应菜单项的单击事件，那么重写Activity 的onOptionsItemSelected(MenuItem mi)方法即可。
     - 使用监听器可以为每个事件单独设置监听菜单事件。setOnMenuItemClickListener(MenuItem.OnMenuItemClickListener menuItemClickListener)
- 点击菜单项跳转其他Activity：调用MenuItem 的setIntent(Intent intent)方法

ContextMenu

- Android 用ContextMenu来代表上下文菜单。因为ContextMenu继承了Menu,因此程序可用相同的方法为它添加菜单项。
- 开发上下文菜单与开发选项菜单的区别:开发上下文菜单不是重写onCreateOptionsMenu(Menu menu)方法，而是重写onCreateContextMenu (ContextMenu menu, View source, ContextMenu.ContextMenuInfo menuInfo)方法。
  - source：参数代表触发上下文菜单的组件。
- 使用步骤
  1. 重写Activity的onCreateContextMenu(ContextMenu menu, View source, ContextMenu.ContextMenuInfo menuInfo)方法。
  2. 调用Activity的registerForContextMenu(View view)方法为view组件注册上下文菜单。
  3. 如果希望应用程序能为菜单项提供响应，则可以重写onContextItemSelected(MenuItem mi)方法，或为指定菜单项绑定事件监听器。

使用XML定义菜单

- 菜单资源文件通常应该放在\res\menu目录下，菜单资源的根元素通常是<men.../>, <men.../>元素无须指定任何属性。
    - <item..>元素:定义菜单项。<ite.../>元素用于定义菜单项，<iterm...> 元素又可包含<men.../>元素，位于<ite.../>元素内部的<men.../>就代表了子菜单。
        - 可以指定id，icon，title等
    - <group..>子元素:将多个iten.../>定义的菜单项包装成一个菜单组。用于控制整组菜单的行为，该元素常用属性：
        - checkableBehavior: 指定该组菜单的选择行为。可指定为none (不可选)、all (多选)和single (单选)三个值。
        - menuCategory: 对菜单进行分类，指定菜单的优先级。有效值为container、system、secondary和alternative。
        - visible:指定该组菜单是否可见。
        - enable:指定该组菜单是否可用。
- 使用菜单：重写Activity的onCreateOptionsMenu (用于创建选项菜单)、onCreateContextMenu (用于创建上下文菜单)方法，在这些方法中调用MenuInflater对象的inflate、方法加载指定资源对应的菜单即可。

使用PopupMenu创建弹出式菜单

- PopupMenu代表弹出式菜单，它会在指定组件上弹出PopupMenu,在默认情况下，PopupMenu会显示在该组件的下方或者上方。
- 使用流程
    1. 调用PopupMenu(Context context, View anchor)构造器创建下拉菜单，anchor 代表要激发该弹出菜单的组件。
    2. 调用MenuInflater的inflate()方法将菜单资源填充到PopupMenu中。
    3. 调用PopupMenu的show()方法显示弹出式菜单。

活动条ActionBar

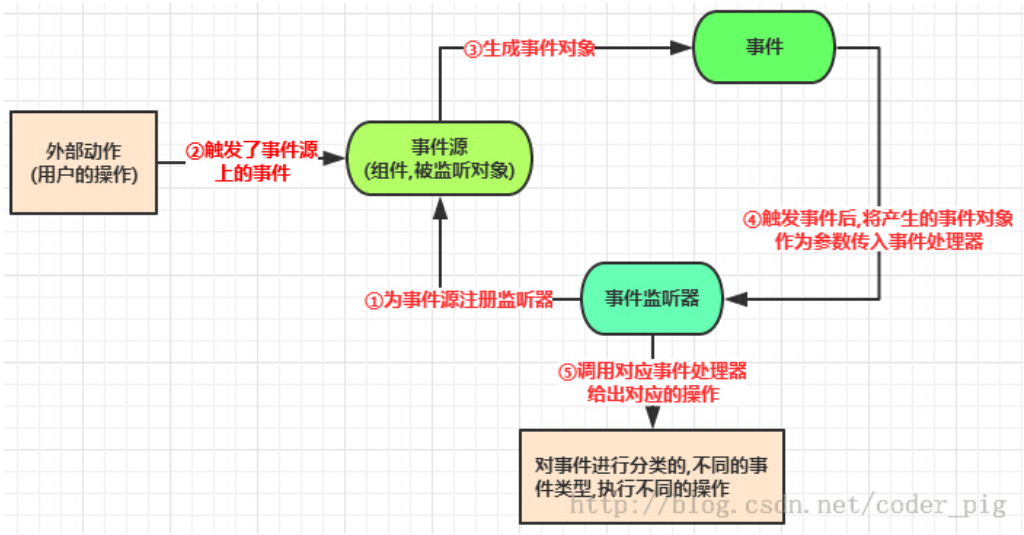
- ActionBar位于屏幕的顶部。ActionBar 可显示应用的图标和Activity 标题，ActionBar 的右边还可以显示活动项(Action Item)。
- 功能
    - 显示选项菜单的菜单项(将菜单项显示成Action Item)。
    - 使用程序图标作为返回Home主屏或向上的导航操作。
    - 提供交互式View作为Action View。
    - 提供基于Tab的导航方式，可用于切换多个Fragment。
    - 提供基于下拉的导航方式。
- ActionBar关闭：安卓不低于3.0的版本都自动打开ActionBar，关闭
    - ```
      <application android: icon="@drawable/ic launcher" android: theme=
      "@android: style/Theme.Material.NoActionBar" android: label="@string/app
      name">
      </application>
      ```
    - ActionBar提供show（ ）和hide（ ）显示和隐藏ActionBar
- ActionBar显示选项菜单项：
    - setShowAsAction(int actionEnum):该方法设置是否将该菜单项显示在ActionBar上，作为Action Item。
      ```

    - android:showAsAction：XML属性设置
    - 如果ActionBar显示空间不足，有Menu按键的按Menu就可以看到，没有Menu按键提供三个的图标，点击可看到剩余的选项菜单项。
- 启用程序图标导航
    - ActionBar方法
方法	说明
setDisplayHomeAsUpEnabled(boolean showHomeAsUp)	设置是否将应用程序图标转变成可点击的图标，并在图标上添加一个向左的箭头。
setDisplayOptions(int options)	通过传入int 类型常量来控制该ActionBar的显示选项。
setDisplayShowHomeEnabled(boolean showHome)	设置是否显示应用程序图标。
- 添加Action View
    - 在ActionBar上除可以显示普通的Action Item 之外，还可以显示普通的UI组件。
        1. 定义Action Item时使用android:actionViewClass属性指定Action View的实现类。
        2. 定义Action Item时使用android:actionI ayout属性指定Action View对应的视图资源。

事件机制

- 当用户在程序界面上执行各种操作时，应用程序必须为用户动作提供响应动作，这种响应动作就需要通过事件处理来完成。
- 方式
    - 基于回调的事件处理
        - 做法：为Android界面组件绑定特定的事件监听器
    - 基于监听的事件处理
        - 做法：重写Android组件特定的回调方法，或者重写Activity的回调方法。

基于监听的事件处理

-


- 处理模型对象
    - Event Source (事件源) ;事件发生的场所，通常就是各个组件，例如按钮、窗口、菜单等。

    - Event (事件) :事件封装了界面组件上发生的特定事情(通常就是一次用户操作)。如果程序需要获得界面组件上所发生事件的相关信息，一般通过Event对象来取得。
    - EventListener(事件监听器);负责监听事件源所发生的事件，并对各种事件做出相应的响应。
- 基于监听的事件处理机制是一种委派式(Delegation) 事件处理方式:普通组件(事件源)将整个事件处理委托给特定的对象(事件监听器);当该事件源发生指定的事件时，就通知所委托的事件监听器，由事件监听器来处理这个事件。
    - 每个组件均可以针对特定的事件指定一个事件监听器
    - 每个事件监听器也可监听一个或多个事件源。
    - 同一个事件源上可能发生多种事件，委派式事件处理方式可以把事件源上所有可能发生的事件分别授权给不同的事件监听器来处理;同时也可以让一类事件都使用同一个事件监听器来处理。
- 使用流程
    1. 获取普通界面组件(事件源)，也就是被监听的对象。
    2. 实现事件监听器类，该监听器类是一个特殊的类，必须实现一个XxxListener接口。
    3. 调用事件源的setXxxListener方法将事件监听器对象注册给普通组件(事件源)。

事件和事件监听器

- 事件监听器：**实现了特定接口的实例**
- **Android对事件监听模型做了进一步简化**:如果事件源触发的事件足够简单，事件里封装的信息比较有限，那就无须封装事件对象，将事件对象传入事件监听器。
- 但对于**键盘事件、屏幕触碰**事件等，此时**程序需要获取事件发生的详细信息**。例如，键盘事件需要获取是哪个键触发的事件;触摸屏事件需要获取事件发生的位置等，对于这种包含更多信息的事件，Android 同样会将事件信息封装成XxxEvent对象，并把该对象作为参数传入事件处理器。
- 在基于监听的事件处理模型中，事件监听器必须实现事件监听器接口，Android为不同的界面组件提供了不同的监听器接口，这些接口通常以内部类的形式存在。
    - View类的内部接口

接口	说明
View.OnClickListener	单击事件的事件监听器必须实现的接口。
View.OnCreateContextMenuListener	创建上下文菜单事件的事件监听器必须实现的接口。
View.onFocusChangeListener	焦点改变事件的事件监听器必须实现的接口。
View.OnKeyListener	按键事件的事件监听器必须实现的接口。
View.OnlongClickListener	长按事件的事件监听器必须实现的接口。
View.OnTouchListener	触摸事件的事件监听器必须实现的接口。

- **事件监听器的形式**
    - 内部类形式:将事件监听器类定义成当前类的内部类。
    - 外部类形式:将事件监听器类定义成一个外部类。
    - Activity本身作为事件监听器类:让Activity本身实现监听器接口，并实现事件处理方法。
    - Lambda表达式或匿名内部类形式:使用Lambda表达式或匿名内部类创建事件监听器对象。

内部类作为事件监听器类

- 优点
    - 使用内部类可以在当前类中复用该监听器类。
    - 监听器类是外部类的内部类，所以可以自由访问外部类的所有界面组件。

外部类作为事件监听器类

- 优点：如果某个事件监听器需要被多个GUI界面所共享，而且主要是完成某种业务逻辑的实现，则可以考虑使用外部类形式来定义事件监听器类。
- 缺点
    - 事件监听器通常属于特定的GUI界面，定义成外部类不利于提高程序的内聚性。
    - 外部类形式的事件监听器不能自由访问创建GUI界面的类中的组件，编程不够简洁。

Activity本身作为事件监听器类

- 优点：形式非常简洁
- 缺点：可能造成程序结构混乱，Activity的主要职责应该是完成界面初始化工作，但此时还需包含事件处理器方法，违背了设计时的本意，从而引起混乱。

Lambda表达式作为事件监听器类

- 优点：大部分时候，事件处理器都没有什么复用价值(可复用代码通常都被抽象成了业务逻辑方法)，因此大部分事件监听器只是临时使用一次，所以使用Lambda表达式形式的事件监听器更合适

直接绑定到标签

- 优点：简单，直接在界面布局文件中为指定标签绑定事件处理方法。

基于回调的事件处理

- 对于基于回调的事件处理模型来说，事件源与事件监听器是统一的，或者说事件监听器完全消失了。当用户在GUI组件上激发某个事件时，组件自己特定的方法将会负责处理该事件。
- 使用回调机制类处理GUI组件上所发生的事件，需要为该组件提供对应的事件处理方法，这种事件处理方法通常都是系统预先定义好的，因此通常需要继承GUI组件类，并通过重写该类的事件处理方法来实现。
    - 为了实现回调机制的事件处理，Android为所有GUI组件都提供了一些事件处理的回调方法
    - View类的事件处理回调方法

方法	说明
boolean onKeyDown(int keyCode, KeyEvent event)	当用户在该组件上按下某个按键时触发该方法。
boolean onKeyLongPress(int keyCode, KeyEvent event)	当用户在该组件上长按某个按键时触发该方法。
boolean onKeyShortcut(int keyCode, KeyEvent event)	当一个键盘快捷键事件发生时触发该方法。
boolean onKeyUp(int keyCode, KeyEvent event)	当用户在该组件上松开某个按键时触发该方法。
boolean onTouchEvent(MotionEvent event)	当用户在该组件上触发触摸屏事件时触发该方法。
boolean onTrackballEvent(MotionEvent event)	当用户在该组件上触发轨迹球事件时触发该方法。

- 使用流程
    1. 自定义View，自定义View时重写该View的事件处理方法
    2. 在界面布局文件中使用这个自定义View
- 基于监听和回调的事件处理的区别
    - 对于基于监听的事件处理模型来说，事件源和事件监听器是**分离**的，当事件源上发生特定事件时，该事件交给事件监听器负责处理。
    - 对于基于回调的事件处理模型来说，事件源和事件监听器是**统一**的，当事件源发生特定事件时，该事件还是由事件源本身负责处理。

基于回调的事件传播

- 几乎所有基于回调的事件处理方法都有一个boolean类型的返回值，该返回值用于标识该处理方法是否能完全处理该事件。
    - 如果处理事件的回调方法返回true, 表明该处理方法已完全处理该事件，该事件不会传播出去。
    - 如果处理事件的回调方法返回false,表明该处理方法并未完全处理该事件，该事件会传播出去。
- 对于基于回调的事件传播而言，某组件上所发生的事件不仅会激发该组件上的回调方法，也会触发该组件所在Activity的回调方法，只要事件能传播到该Activity。
- 传播顺序
    - 如果让任何一个事件处理方法返回了true,那么该事件将不会继续向外传播。
    - 当该组件上发生触碰事件时
        1. 触发该组件绑定的事件监听器
        2. 触发该组件提供的事件回调方法
        3. 传播到该组件所在的Activity

响应系统设置的事件

- 背景：有时候可能需要让应用程序随系统设置而进行调整

Configuration 类

- Configuration类专门用于描述手机设备上的配置信息，这些配置信息既包括用户特定的配置项，也包括系统的动态设备配置。
- ```
  //获得Activity的Configuration对象
  Configuration cfg = getResources().getConfiguration();
  ```
  - Configuration对象的常用属性
属性	说明
float fontScale	获取当前用户设置的字体的缩放因子。
int keyboard	获取当前设备所关联的键盘类型。该属性可能返回KEYBOARD NOKEYS、KEYBOARD QWERTY (普通电脑键盘)、KEYBOARD 12KEY (只有12个键的小键盘)等属性值。
int keyboardHidden	该属性返回一个boolean值用于标识当前键盘是否可用。该属性不仅会判断系统的硬件键盘，也会判断系统的软键盘(位于屏幕上)。如果该系统的硬件键盘不可用，但软键盘可用，该属性也会返回KEYBOARDHIDDEN NO;只有当两个键盘都不可用时才返回KEYBOARDHIDDEN YES。
Locale locale	获取用户当前的Locale。
int mcc	获取移动信号的国家码。
int mnc	获取移动信号的网络码。
int navigation	判断系统上方向导航设备的类型。该属性可能返回NAVIGATION_ NONAV(无导航)、NAVIGATION DPAD (DPAD导航)、NAVIGATION_ TRACKBALL (轨迹球导航)、NAVIGATION WHEEL (滚轮导航)等属性值。
int orientation	获取系统屏幕的方向，该属性可能返回ORIENTATION LANDSCAPE (横向屏幕)、ORIENTATION PORTRAIT (竖向屏幕)、ORIENTATION SQUARE (方形屏幕)等属性值。
int touchscreen	获取系统触摸屏的触摸方式。该属性可能返回TOUCHSCREEN NOTOUCH (无触摸屏)、TOUCHSCREEN STYLUS (触摸笔式的触摸屏)、TOUCHSCREEN FINGER (接受手指的触摸屏)等属性值。

响应系统设置更改

- 基于回调的事件处理方法：重写Activity 的onConfigurationChanged(Configuration newConfig)方法， 当系统设置发生更改时，该方法会被自动触发。
- 为了让该Activity能监听更改的事件，需要在配置该Activity时指定android:configChanges属性

Handler消息传递机制

- 背景：出于性能优化考虑，Android 的UI 操作并不是线程安全的。为了解决这个问题，Android 制定了一条简单的规则:只允许UI线程修改Activity 里的UI组件。Android平台只允许UI线程修改Activity里的UI组件，这样就会导致新启动的线程无法动态改变界面组件的属性值。
  ```

Handler类

- 作用
  - 在新启动的线程中发送消息。
  - 在主线程中获取、处理消息。
- **新启动的线程何时发送消息呢?主线程何时去获取并处理消息呢?**
  - 为了让主线程能"适时"地处理新启动的线程所发送的消息，显然只能通过回调的方式来实现——开发者只要重写Handler类中处理消息的方法，当新启动的线程发送消息时，消息会发送到与之关联的MessageQueue, 而Handler 会不断地从MessageQueue中获取并处理消息——这将 导致Handler类中处理消息的方法被回调。
-

方法	说明
handleMessage(Message msg)	处理消息的方法。该方法通常用于被重写。
hasMessages(intwhat)	检查消息队列中是否包含what属性为指定值的消息。
hasMessages(int what, Object object)	检查消息队列中是否包含what属性为指定值且object属性为指定对象的消息。
Message obtainMessage()	获取消息。
sendEmptyMessage(int what)	发送空消息。
sendEmptyMessageDelayed(int what, long delayMillis)	指定多少毫秒之后发送空消息。
sendMessage(Message msg)	立即发送消息。
sendMessageDelayed(Message msg, long delayMillis)	指定多少毫秒之后发送消息。

Handler、Loop、MessageQueue的工作原理

- 介绍
  - Message: Handler 接收和处理的消息对象。
  - Looper:每个线程只能拥有一个Looper.它的loop方法负责读取MessageQueue中的消息，读到信息之后就把消息交给发送该消息的Handler进行处理。
  - MessageQueue:消息队列，它采用先进先出的方式来管理Message。程序创建Looper对象时，会在它的构造器中创建MessageQueue对象。
  - Handler:它的作用有两个，即发送消息和处理消息，程序使用Handler发送消息，由Handler发送的消息必须被送到指定的MessageQueue。
    - 如果希望Handler正常工作，必须在当前线程中有一个Looper对象。为了保证当前线程中有Looper对象，可以分如下两种情况处理。
      - 在主UI线程中，系统已经初始化了一个Looper对象，因此程序直接创建Handler即可，然后就可通过Handler来发送消息、处理消息了。
      - 程序员自己启动的子线程，必须自己创建一个Looper对象，并启动它。创建Looper对象调用它的prepare( ) 方法即可。
        - prepare( ) 方法保证一个线程只会有一个Looper对象。
- 作用

    - Looper:每个线程只有一一个 Looper,它负责管理MessageQueue,会不断地从MessageQueue中取出消息，并将消息分给对应的Handler处理。
    - MessageQueue:由Looper负责管理。它采用先进先出的方式来管理Message。
    - Handler:它能把消息发送给Looper管理的MessageQueue,并负责处理Looper分给它的消息。
- 线程中使用Handler的流程
    1. 调用Looper的prepare()方法为当前线程创建Looper对象，创建Looper对象时，它的构造器会创建与之配套的MessageQueue。
    2. 有了Looper之后，创建Handler子类的实例，重写handleMessage()方法， 该方法负责处理来自其他线程的消息。
    3. 调用Looper的loop()方法启动Looper。
- ANR:Application Not Responding

异步任务(AsyncTask)

- 背景
    - 获取网络数据之后，新线程不允许直接更新UI组件。
        - 解决方案。
            - 使用Hanlder实现线程之间的通信。
            - Activity.runOnUiThread(Runnable)。
            - View.post(Runnable)。
            - View.postDelayed(Runnable, long)。
- 作用：异步任务(AsyncTask)则可进一步简化这种操作。相对来说AsyncTask更轻量级一些，适用于简单的异步处理，不需要借助线程和Handler即可实现。
- Async Task<Params, Progress, Result>是一个抽象类，通常用于被继承，继承AsyncTask时需要泛型参数。
    - Params: 启动任务执行的输入参数的类型。
    - Progress: 后台任务完成的进度值的类型。
    - Result: 后台执行任务完成后返回结果的类型。
- 使用流程
    1. 创建AsyncTask的子类,并为三个泛型参数指定类型。如果某个泛型参数不需要指定类型，则可将它指定为Void。
    2. 根据需要，实现AsyncTask的如下方法。

方法	说明
doInBackground(Param..)	重写该方法就是后台线程将要完成的任务。该方法可以调用publishProgress(Progress.. values)方法更新任务的执行进度。
onProgressUpdate(Progress... values)	在doInBackground0方法中调用publishProgress( ) 方法更新任务的执行进度后，将会触发该方法。
onPreExecute()	该方法将在执行后台耗时操作前被调用。通常该方法用于完成一~些初始化的准备工作，比如在界面上显示进度条等。
onPostExecute(Result result)	当doInBackground()完成后，系统会自动调用onPostExecute( ) 方法，并将doInBackground0方法的返回值传给该方法。

3. 调用AsyncTask子类的实例的execute(Params... params)开始执行耗时任务。
4. 规则
    - 必须在UI线程中创建AsyncTask的实例。
    - 必须在UI线程中调用AsyncTask的execute(方法。
    - AsyncTask的onPreExecute( )、onPostExecute(Result result) 、doInBackground (Param...params)、onProgressUpdate(rogres... values)方法， 不应该由程序员代码调用，而是由Android系统负责调用。
    - 每个AsyncTask只能被执行一-次，多次调用将会引发异常。

Activity和Fragment

- Activity 充当了应用与用户互动的入口点。可以将 Activity 实现为 Activity 类的子类。
- Android应用的多个Activity组成Activity 栈，当前活动的Activity位于栈顶。
- 当Activity处于Android应用中运行时，同样受系统控制，有其自身的生命周期

Activity

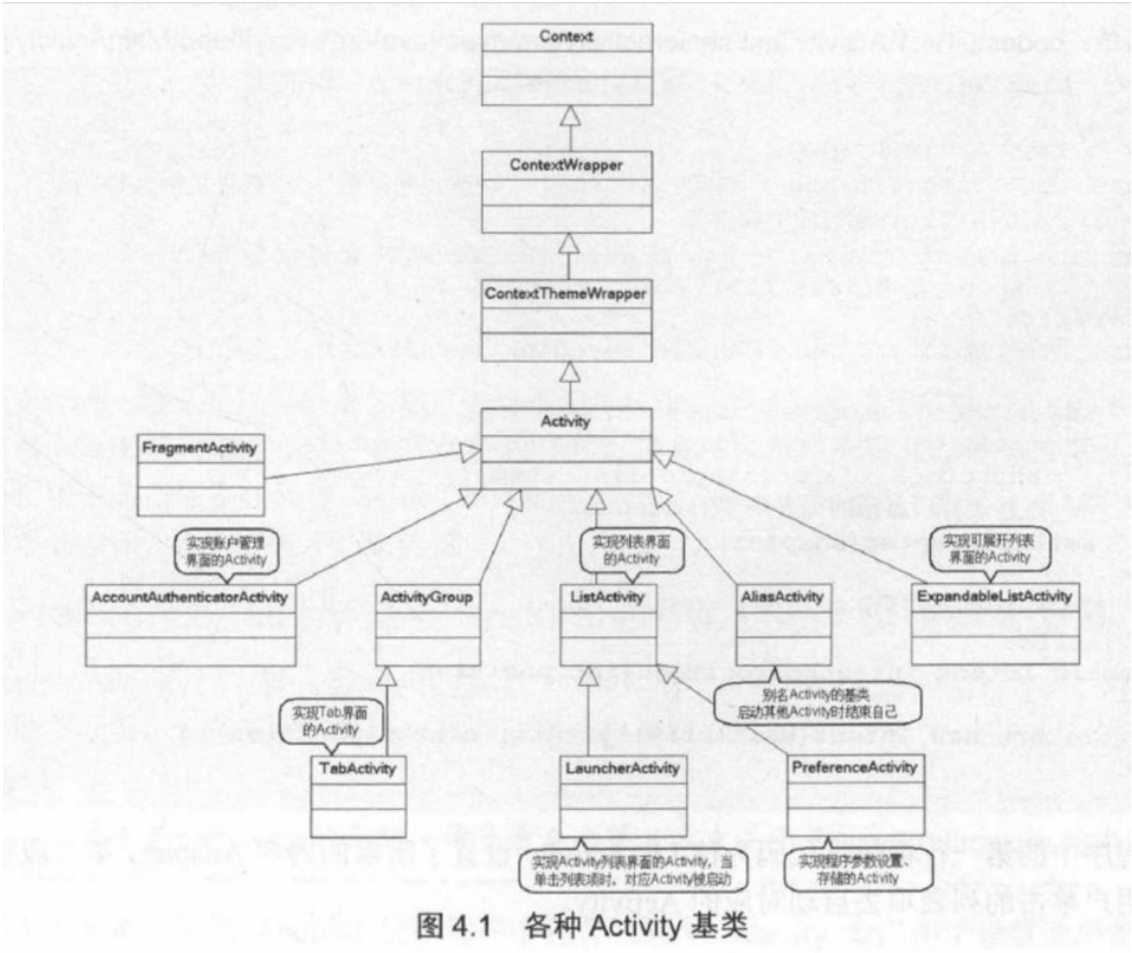
- Activity是Android应用中最重要、最常见的应用组件(此处的组件是**粗粒度的系统组成部分**，并非指界面控件: widget)。
- 

图 4.1　各种 Activity 基类
- 当一个Activity 类定义出来之后，这个Activity类何时被实例化、它所包含的方法何时被调用，这些都不是由开发者决定的，都应该由Android系统来决定。
- 创建一个 Activity 也需要实现一个或多个方法，其中最常见的就是实现onCreate(Bundle status)方法，该方法将会在Activity创建时被回调，该方法调用Activity 的setContentView(View view)方法来显示要展示的View。为了管理应用程序界面中的各组件，调用Activity的findViewByld(int id)方法来获取程序界面中的组件。

ListActivity

- 如果应用程序界面只包括列表，则可以让应用程序继承ListActivity。

LauncherActivity

- LauncherActivity**继承了ListActivity**, 因此它本质上也是一个开发列表界面的Activity, 但它开发出来的列表界面与普通列表界面有所不同。它开发出来的列表界面中的每个列表项都对应于一个Intent,因此当用户单击不同的列表项时，应用程序会自动启动对应的Activity。

配置Activity

- Android应用要求所有应用程序组件(Activity、Service、 ContentProvider、 BroadcastReceiver)都必须显式进行配置。
- 只要为<application..>元素添加<activity...>子元素即可配置Activity。
  - 配置Activity 时通常指定属性

属性	说明
name	指定该Activity的实现类的类名
icon	指定该Activity对应的图标。
label	指定该Activity的标签。
exported	指定该Activity是否允许被其他应用调用。如果将该属性设为true,那么该Activity将可以被其他应用调用。
launchMode	指定该Activity 的加载模式，该属性支持standard、singleTop、singleTask 和singleInstance

  - 在配置Activity时通常还需要指定一个或多个<intent-filter../>元素，该元素用于指定该Activity可响应的Intent。

启动、关闭Activity

- Activity启动其他Activity

方法	说明
startActivity(Intent intent)	启动其他Activity。
startActivityForResult(Intent intent, int requestCode)	以指定的请求码(requestCode)启动 Activity,而且程序将会获取新启动的Activity 返回的结果(通过重写onActivityResult()方法来获取)。

    - 启动Activity时可指定一个requestCode 参数，该参数代表了启动Activity的请求码。这个请求码的值由开发者根据业务自行设置，用于标识请求来源。
- 关闭Activity

方法	说明
finish()	结束当前Activity。
finishActivity(int requestCode)	结束以startActivityForResult(Intent intent, int requestCode)方法启动的Activity.

Bundle交换数据

- Activity之间进行数据交换使用“信使": Intent, 主要将需要交换的数据放入Intent 中。
  - 使用getIntent()方法就可以获得启动该Activity的Intent
- Intent数据交换方法
方法	说明
putExtras(Bundle data)	向Intent中放入需要“携带”的数据包。
Bundle getExtras()	取出Intent中所“携带”的数据包。
putExtra(String name, Xxx value)	向Intent中按key-value 对的形式存入数据。
getXxxExtra(String name)	从Intent中按key取出指定类型的数据。
  - Bundle数据交换方法
方法	说明
putXxx(String key，Xxx data)	向Bundle中放入Int、Long 等各种类型的数据。
putSerializable(String key, Serializable data)	向Bundle中放入一个可序列化的对象。
getXxx(String key)	从Bundle 中取出Int、Long 等各种类型的数据。
getSerializable(String key, Serializable data)	从Bundle中取出一一个可序列化的对象。

启动其他Activity并返回结果

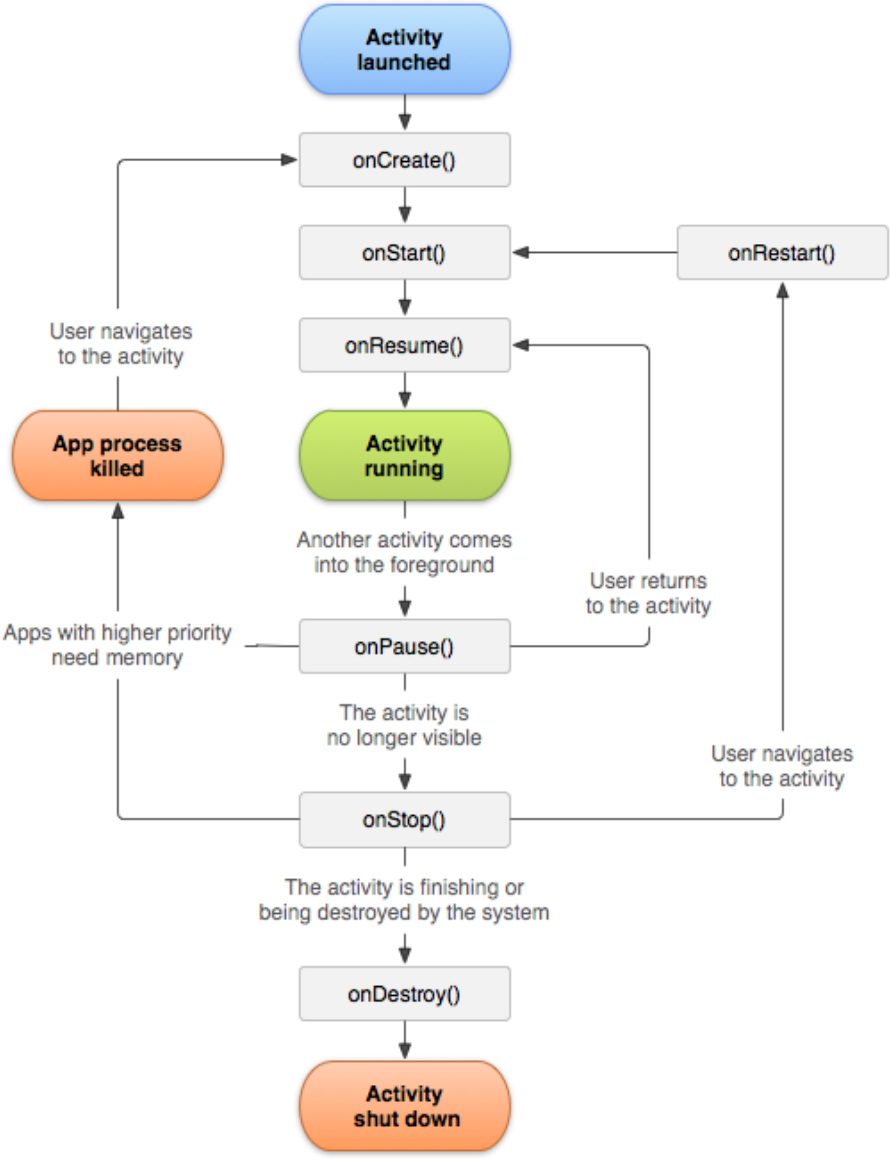
- Activity 还提供一个startActivityForResult(Intent intent, int requestCode)方法来启动其他Activity。该方法用于启动指定Activity,而且期望获取指定Activity 返回的结果。
  - 一个Activity中可能调用多个startActivityForResult()方法来打开多个不同的Activity处理不同的业务，当这些新Activity 关闭后，系统都将回调前面Activity 的onActivityResult(int requestCode, int resultCode, Intent data)方法。
    - **requestCode**：为了知道该方法是由哪个请求的结果所触发的
    - **resultCode**：为了知道返回的数据来自哪个新的Activity
  - 获取被启动的Activity所返回的结果
    1. 当前 Activity需要重写onActivityResult(int requestCode, int resultCode，Intent intent)，当被启动的Activity返回结果时，该方法将会被触发，其中requestCode代表请求码,而resultCode代表Activity 返回的结果码，这个结果码也是由开发者根据业务自行设定的。
    2. 被启动的Activity需要调用setResult()方法设置处理结果。

回调机制

- 回调函数，或简称回调（Callback 即call then back 被主函数调用运算后会返回主函数），是指通过参数将函式传递到其它代码的，某一块可执行代码的引用。这一设计允许了底层代码调用在高层定义的子程序。
- java实现方式

1. 以接口形式存在:该接口由开发者实现，实现该接口时将会实现该接口的方法，那么通用的程序架构就会回调该方法来完成业务相关的处理。
2. 以抽象方法(也可以是非抽象方法)的形式存在:这就是Activity 的实现形式。在这些特定的点上方法已经被定义了，如onCreate、onActivityResult 等方法，开发者可以有选择性地重写这些方法，通用的程序架构就会回调该方法来完成业务相关的处理。

生命周期

- Activity运行过程所处的不同状态也被称为生命周期
- 系统中的活动作为 活动堆栈进行管理。当一个新的activity启动时，它通常被放置在当前栈的顶部，成为正在运行的activity——前一个activity在栈中始终保持在它的下方，并且在新的activity退出之前不会再次来到前台。屏幕上可以看到一个或多个活动堆栈。
  - 状态
    - 运行状态:当前Activity位于前台，用户可见，可以获得焦点。
    - 暂停状态:其他Activity位于前台，该Activity依然可见，只是不能获得焦点。
    - 停止状态:该Activity不可见，失去焦点。
    - 销毁状态:该Activity结束，或Activity所在的进程被结束。
  - 

  - 方法

方法	说明
onCreate(Bundle savedStatus)	创建Activity时被回调。该方法只会被调用一一次。
onStart()	启动Activity时被回调。
onRestart()	重新启动Activity时被回调。
onResume()	恢复Activity 时被回调。在onStart0方法后一定 会回调onResume0方法。
onPause()	暂停Activity时被回调。
onStop()	停止Activity 时被回调。
onDestroy()	销毁Activity 时被回调。该方法只会被调用一次。

Activity与Servlet

- 相同
  1. Activity、 Servlet的职责都是向用户呈现界面。
  2. 开发者开发Activity、Servlet 都继承系统的基类。
  3. Activity、Servlet 开发出来之后都需要进行配置。
  4. Activity运行于Android应用中，Servlet 运行于Web应用中。
  5. 开发者无须创建Activity、Servlet 的实例，无须调用它们的方法。Activity、 Servlet 的方法都由系统以回调的方式来调用。
  6. Activity、Servlet都有各自的生命周期，它们的生命周期都由外部负责管理。
  7. Activity、Servlet 都不会直接相互调用，因此都不能直接进行数据交换。Servlet之间的数据交换需要借助于Web应用提供的requestScope、sessionScope 等; Activity 之间的数据交换要借助于Bundle。
- 区别

Activity	Servlet
Activity是Android窗口的容器，因此Activity最终以窗口的形式显示出来	Servlet 并不会生成应用界面，只是向浏览者生成文本响应。
Activity运行于Android应用中，因此Activity 的本质还是通过各种界面组件来搭建界面	Servlet则主要以I0流向浏览者生成文本响应，浏览者看到的界面其实是由浏览器负责生成的。
Activity之间的跳转主要通过Intent对象来控制	而Servlet之间的跳转则主要由用户请求来控制。

加载模式

- 配置Activity时可指定android:launchMode 属性，该属性用于配置该Activity的加载模式。该属性支持4个属性值。

属性值	说明
standard	标准模式，这是默认的加载模式。
singleTop	Task 栈顶单例模式。
singleTask	Task 内单例模式。
singleInstance	全局单例模式。

- Android对Activity 的管理：Android 采用Task来管理多个Activity, 当启动一个应用时，Android 就会为之创建一个Task, 然后启动这个应用的入口Activity (即`<intent- ft...>`中配置为MAIN和LAUNCHER的Activity)。
  - **Task**：开发者无法真正去访问Task,只能调用Activity的getTaskld()方法来获取它所在的Task的ID.可以把Task理解成**Activity栈**，Task以栈的形式来管理Activity:先启动的Activity被放在Task栈底，后启动的Activity被放在Task栈项。
- Activity的加载模式:负责管理实例化、加载Activity的方式，并可以控制Activity 与Task之间的加载关系。

standard模式

- 每次通过standard模式启动目标Activity时，Android总会为目标Activity创建一个新的实例，并将该Activity添加到当前Task栈中,这种 模式**不会启动新的Task,新Activity 将被添加到原有的Task中。**

singleTop模式

- singleTop模式与standard模式基本相似。不同之处为：当将要启动的目标Activity已经位于Task栈项时，系统不会重新创建目标Activity的实例，而是直接复用已有的Activity实例。
  - 如果将要启动的目标Activity没有位于Task栈顶，此时系统会重新创建目标Activity的实例，并将它加载到Task栈项一此时与standard模式完全相同。

singleTask 模式

- 采用singleTask这种加载模式的Activity 能保证在同一个Task内只有一个实例，当系统采用singleTask 模式启动目标Activity时，可分为如下三种情况。
  - 如果将要启动的目标Activity
    - 不存在，系统将会创建目标Activity的实例，并将它加入Task栈顶。
    - 已经存在
      - 并且位于Task栈项，此时与singleTop模式的行为相同。
      - 但没有位于Task栈顶，系统将会把位于该Activity上面的所有Activity移出Task栈，从而使得目标Activity转入栈顶。

singleInstance模式

- 在singleInstance这种加载模式下，系统保证无论从哪个Task中启动目标Activity, 只会创建一个目标Activity实例，并会使用一个全新的Task栈来加载该Activity实例。
  - 如果将要启动的目标Activity
    - 不存在，系统会先创建-一个全新的Task,再创建目标Activity的实例，并将它加入新的Task栈顶。
    - 已经存在，无论它位于哪个应用程序中、位于哪个Task中，系统都会把该Activity所在的Task转到前台，从而使该Activity显示出来。

- 注意：需要指出的是，采用singleInstance模式加载Activity总是位于Task栈顶，且采用singleInstance模式加载的Activity所在Task将只包含该Activity。

Fragment

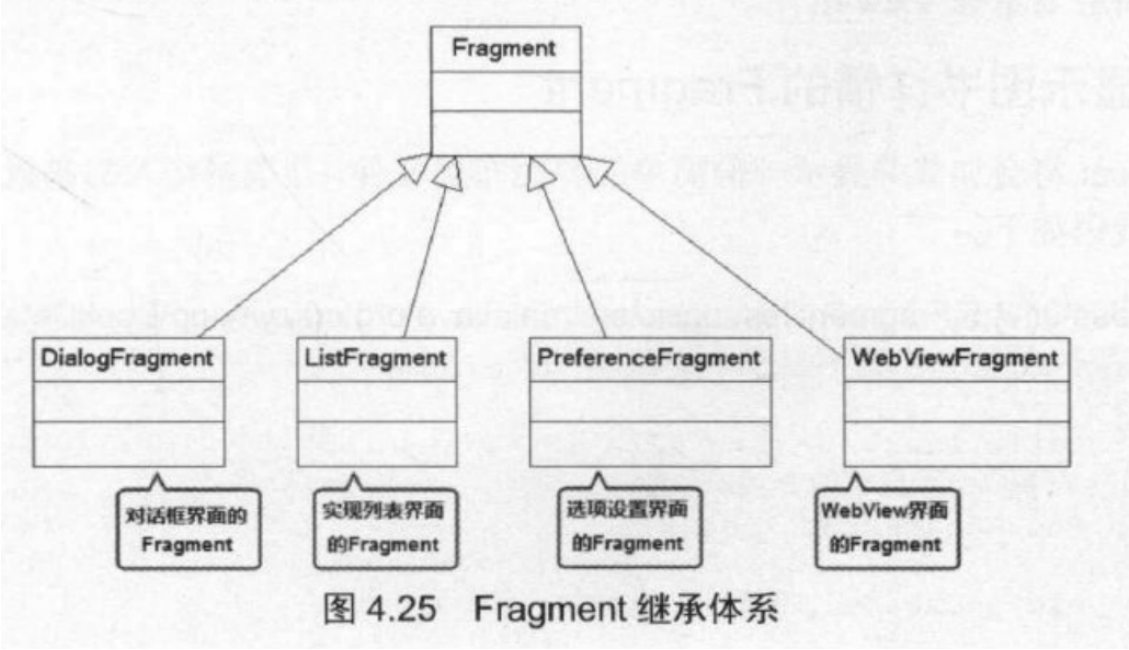
- Fragment代表了Activity 的子模块，因此可以把Fragment理解成Activity 片段(Fragment 本身就是片段的意思)。Fragment 拥有自己的生命周期，也可以接受它自己的输入事件。
- Fragment必须被"嵌入"Activity 中使用，因此，虽然Fragment也拥有自己的生命周期，但Fragment的生命周期会受它所在的Activity的生命周期控制。
- 特征
    1. Fragment总是作为Activity 界面的组成部分。Fragment可调用getActivity()方法获取它所在的Activity, Activity 可调用FragmentManager的findFragmentById()或findFragmentByTag()方法来获取Fragment。
    2. 在Activity运行过程中，可调用FragmentManager的add()、remove()、 replace()方法动态地添加、删除或替换Fragment。
    3. 一个Activity 可以同时组合多个Fragment;反过来，一个Fragment也可被多个Activity 复用(多对多)。
    4. Fragment可以响应自己的输入事件，并拥有自己的生命周期，但它们的生命周期直接被其所属的Activity的生命周期控制。
- 作用
    - 适应大屏幕的平板电脑，由于平板电脑的屏幕比手机屏幕更大，因此可以容纳更多的UI组件，且这些UI组件之间存在交互关系。
    - 简化了大屏幕UI的设计，它不需要开发者管理组件包含关系的复杂变化，开发者使用Fragment对UI组件进行分组、模块化管理，就可以更方便地在运行过程中动态更新Activity的用户界面。
- 

图 4.25　Fragment 继承体系

创建Fragment

- 实现Fragment与实现Activity 非常相似，它们都需要实现与Activity类似的回调方法，例如onCreate()、onCreateView()、 onStart()、 onResume()、 onPause()、 onStop()等 。
    - onCreateView():当Fragment 绘制界面组件时会回调该方法。该方法必须返回一个View,该View也就是该Fragment所显示的View。
- 创建ListFragment的子类:无须重写onCreateView()方法,与 ListActivity 类似，只要调用ListFragment 的setListAdapter()方法为该Fragment设置Adapter即可。该LitFragment将会显示该Adapter提供的列表项。

Fragment和Activity通信

- 为了在Activity中显示Fragment,还必须将Fragment添加到Activity 中。
  - 方法
    1. 在布局文件中使用<fragment..>元素添加Fragment, <fragmen.../> 元素的android:name属性指定Fragment的实现类。
       - 在界面布局文件中使用<fragment...>元素添加Fragment 时，可以为<fragmen...>元素指定android:id或android:tag 属性，这两个属性都可用于标识该Fragment
    2. 在代码中通过FragmentTransaction对象的add()方法来添加Fragment。
  - Activity的getSupportFragmentManager()方法可返回FragmentManager, FragmentManager 对象的beginTransaction()方法即可开启并返回FragmentTransaction 对象。

Fragment和Activity传递数据

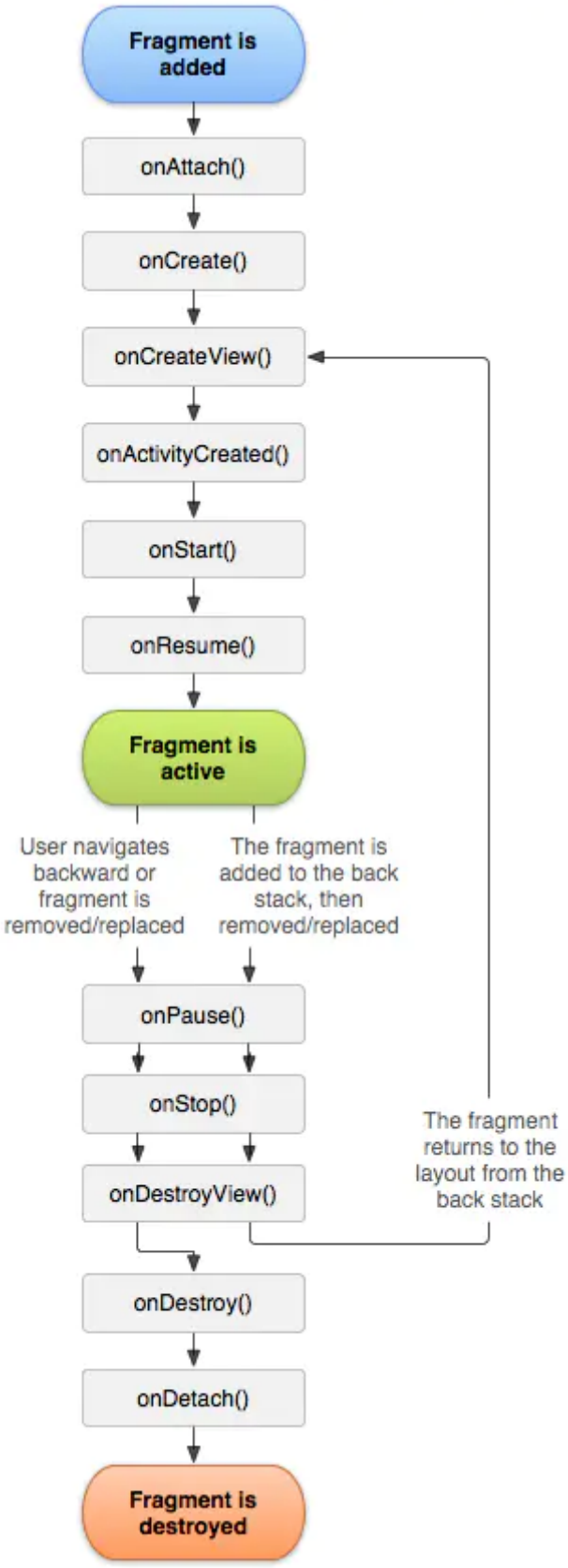
- Activity向Fragment 传递数据：在Activity 中创建Bundle 数据包，并调用Fragment 的setArguments(Bundle bundle)方法即可将Bundle数据包传给Fragment。
- Fragment向Activity传递数据或Activity需要在Fragment运行中进行实时通信:在Fragment中定义一个内部回调接口，再让包含该Fragment的Activity实现该回调接口，这样Fragment即可调用该回调方法将数据传给Activity。

FragmentManager与FragmentTransaction

- FragmentManager的功能。
  - 使用findFragmentById()或findFragmentByTag()方法来获取指定Fragment。
  - 调用popBackStack()方法将Fragment从后台栈中弹出(模拟用户按下BACK按键)。
  - 调用addOnBackStackChangeListener()注册一个监听器，用于监听后台栈的变化。
- FragmentTransaction功能
  - 每个FragmentTransaction可以包含多个对Fragment 的修改，比如包含调用了多个add()、remove()、和replace()操作，最后调用commit()方法提交事务即可。
  - 在调用commit()之前，开发者也可调用addToBackStackO将事务添加到Back栈，该栈由Activity负责管理，这样允许用户按BACK按键返回到前一个Fragment状态。

生命周期

-


Fragment is added
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
Fragment is active
User navigates backward or fragment is removed/replaced
The fragment is added to the back stack, then removed/replaced
onPause()
onStop()
onDestroyView()
The fragment returns to the layout from the back stack
onDestroy()
onDetach()
Fragment is destroyed

方法	说明
onAttach()	当该Fragment被添加到它所在的Context时被回调。该方法只会被调用一次。
onCreate()	创建Fragment时被回调。该方法只会被调用一次。
onCreateView()	每次创建、绘制该Fragment的View组件时回调该方法，Fragment 将会显示该方法返回的View组件。
onActivityCreated()	当Fragment 所在的Activity被启动完成后回调该方法。
onStart()	启动Fragment时被回调。
onResume()	恢复Fragment 时被回调，在onStart()方法后一定会回调onResume()方法。
onPause()	暂停Fragment时被回调。
onStop()	停止Fragment时被回调。
onDestroyView( )	销毁该Fragment 所包含的View组件时调用。
onDestroy()	销毁Fragment时被回调。该方法只会被调用一次。
onDetach()	将该Fragment 从它所在的Context中删除、替换完成时回调该方法，在onDestroy()方法后一定会回调onDetach( ) 方法。该方法只会被调用一次。

Fragment导航

- 背景：当一个Activity包含多个Fragment时，用户可能需要在多个Fragment之间导航。
- Android 在support-fragment下提供了一个ViewPager组件，该组件可以非常方便地实现分页导航。
  - ViewPager组件的用法与AdapterView有点类似，ViewPager 只是一个容器(它继承了ViewGroup),该组件所显示的内容由它的Adapter 提供，因此使用ViewPager时必须为它设置一个Adapter。
  - ViewPager所使用的Adpater 也是它独有的PagerAdapter, 该PagerAdapter 还有一个FragmentPagerAdapter子类，专门用于管理多个Fragment。
- tab导航，将TableLayout和ViewPage关联

Intent和IntentFilter

- 优点
  - Android使用统一的Intent对象来封装这种“启动意图”,很明显使用Intent提供了一致的编程模型。
  - 在某些时候，应用程序只是想启动具有某种特征的组件，而不启动具体的组件
  - 将intent和组件解耦，提高系统的可拓展性和可维护性
  - 应用程序组件之间通信的重要媒介

Intent对象

- Activity、 Service、 BroadcastReceiver都是依靠 Intent来启动，Intent就封装了程序想要启动程序的意图。Intent还可用于与被启动组件交换信息。

- PendingIntent对象：PendingIntent 是对Intent的包装，一般通过调用PendingIntent的getActivity()、getService()、getBroadcastReceiver()静态方法来获取PendingIntent对象。与Intent对象不同的是，PendingIntent 通常会传给其他应用组件，从而由其他应用程序来执行PendingIntent所包装的“Intent"。
组件类型	启动方法
Activity	startActivity(Intent intent)
	startActivityForResult(Intent intent, int requestCode)
Service	ComponentName startService(Intent service)
	boolean bindService(Intent service, ServiceConnection conn, int flags)
BroadcastReceiver	sendBroadcast(Intent intent)
	sendBroadcast(Intent intent, String receiverPermission)
	sendOrderedBroadcast(Intent intent, String receiverPermission)
	sendOrderedBroadcast(Intent intent, String receiverPermnission,BroadcastReceiver resultReceiver, Handler scheduler, int initialCode, String initialData, Bundle initialExtras)
	sendStickyBroadcast(Intent intent)
	sendStickyOrderedBroadcast(Intent intent, BroadcastReceiver resultReceiver, Handler scheduler, int initialCode,String initialData, Bundle initialExtras)

Intent的属性及intent-filter 配置

- 属性：Intent对象大致包含Component、Action、 Category、 Data、 Type、Extra 和Flag这7种属性，其中Component用于明确指定需要启动的目标组件，而Extra则用于"携带”需要交换的数据。
  - Intent代表了Android应用的启动“意图”，Android应用将会根据Intent来启动指定组件，至于到底启动哪个组件，则取决于Intent的各属性。

Component属性

- Intent的Component属性需要接受一个ComponentName对象
构造器	说明
ComponentName(String pkg, String cls)	创建pkg所在包下的cls类所对应的组件。
ComponentName(Context pkg, String cls)	创建pkg所对应的包下的cls类所对应的组件。
ComponentName(Context pkg, Class<?> cls)	创建pkg所对应的包下的cls类所对应的组件。

    - 一个ComponentName需要指定包名和类名,这就可以唯一地确定一个组件类，这样应用程序即可根据给定的组件类去启动特定的组件。
  - Intent 还包含了如下三个方法。

方法	说明
setClass(Context packageContext, Class<?> cls)	设置该Intent将要启动的组件对应的类。
setClassName(Context packageContext, String className)	设置该Intent将要启动的组件对应的类名。
setClassName(String packageName, String className)	设置该Intent将要启动的组件对应的类名。

- 显式Intent：指定Component属性的Intent已经明确了它将要启动哪个组件，因此这种Intent也被称为显式Intent
- 隐式Intent：没有指定Component属性的Intent被称为隐式Intent,隐式Intent没有明确指定要启动哪个组件，应用将会根据Intent指定的规则去启动符合条件的组件，但具体是哪个组件则不确定。

Action、Category 属性与intent-filter 配置

- Intent的Action、Category 属性的值都是一个普通的字符串，其中Action代表该Intent所要完成的一个抽象“动作”，而Category则用于为Action增加额外的附加类别信息。
  - Action要完成的只是一个抽象动作，这个动作具体由哪个组件(或许是Activity, 或许是BroadcastReceiver)来完成，Action 这个字符串本身并不管。
- `<intent- filter..>` 元素是AndroidManifest.xml文件中<activtit.../>元素的子元素，<activity../> 元素用于为应用程序配置Activity, <activity.../>的<intent-fite...>.子元素则用于配置该Activity 所能“响应”的Intent。
  - <intent-fiter...>元素的子元素。
    - 0~N个<acion.../>子元素。
      - <action..>、 <category../> 子元素的配置非常简单，都可指定android:name属性，该属性的值就是一个普通字符串。
      - 当<activity...>元素 的<intent-filter..>.子元素里包含多个<action..>子元素(相当于指定了多个字符串)时，就表明该Activity 能响应Action属性值为其中任意一个字符串的Intent。
    - 0~N个<category...>子元素。
    - 0~1个<da...>子元素。
  - 注意：<intent-filter...>元素也可以是<service...>. <receiver..>两个元素的子元素，用于表明它们可以响应的Intent.
- 每个Intent只能指定一个 Action “要求”，但可以指定多个Category“要求”
  - 一个Intent 对象最多只能包括一个Action属性，程序可调用Intent 的setAction(String str)方法来设置Action属性值;
  - 一个Intent对象可以包括多个Category属性，程序可调用Intent的addCategory (String str)方法来为Intent添加Category属性。
  - 当程序创建Intent时，该Intent 默认启动Category 属性值为Intent.CATEGORY DEFAULT常量(常量值为android.intent.category.DEFAULT)的组件。

指定Action、Category调用系统Activity

- Intent对象不仅可以启动本应用内程序组件，也可以启动Android系统的其他应用的程序组件，包括系统自带的程序组件(只要权限允许)。
- Android内部提供了大量标准的Action、Catetory 常量。

Data、Type 属性与intent-filter 配置

- Data属性通常用于向Action属性提供操作的数据。Data 属性接受一个 Uri对象，该Uri对象通常通过 `scheme://host:port/path` 形式的字符串来表示。
- Type属性用于指定该Data属性所指定Uri对应的MIME类型，这种MIME类型可以是任何自定义的MIME类型，只要符合abc/xyz 格式的字符串即可。
    - MIME(Multipurpose Internet Mail Extensions)多用途互联网邮件扩展类型。它是一个互联网标准，扩展了电子邮件标准，使其能够支持：非ASCII字符文本；非文本格式附件（二进制、声音、图像等）；由多部分（multiple parts）组成的消息体；包含非ASCII字符的头信息（Header information）。在万维网中使用的HTTP协议中也使用了MIME的框架，标准被扩展为互联网媒体类型。
- Data属性与Type属性会相互覆盖
    - 如果为Intent先设置Data属性，后设置Type属性，那么Type属性将会覆盖Data属性。
    - 如果为Intent先设置Type属性，后设置Data属性，那么Data属性将会覆盖Type属性。
    - 如果希望Intent 既有Data属性，也有Type属性，则应该调用Intent 的setDataAndType()方法。
- 在AndroidManifest.xml文件中为组件声明Data、Type 属性都通过<data../>元素
    - mimeType指定Type
        - mimeType和Type必须一致才能启动
    - scheme、host、port、path、pathPrefix、pathPattern指定Data
        - pathPrefix：指定Data属性的path前缀。
        - pathPattern：指定Data属性的path字符串模板。
        - 并不要求完全一致。Data属性的“匹配”过程则有些差别，它会先检查<intent-filte.../>里的<data...>子元素
            - 如果目标组件的<data.../>子元素只指定了android:scheme 属性，那么只要Intent的Data属性的scheme部分与android:scheme属性值相同，即可启动该组件。
            - 如果目标组件的<data..>子元素只指定了android:scheme、 android:host属性，那么只要Intent的Data属性的scheme、host部分与android:scheme、android:host 属性值相同，即可启动该组件。
            - 如果目标组件的<data.../> 子元素指定了android:scheme 、android:host、android:port 属性,那么只要Intent 的Data属性的scheme、host、 port 部分与android:scheme 、android:host、android:host属性值相同，即可启动该组件。
            - 如果<data.../>子元素只有android:port 属性，没有指定android:host 属性，那么android:port属性将不会起作用。
            - android:port属性可省略。
    - **注意**：如果希望<data.../>子元素能正常起作用，至少要配置一个<action.../>子元素，但该子元素的android:name属性值可以是任意的字符串。

使用Action、Data 属性启动系统Activity

- 一旦为Intent同时指定了Action、Data属性，Android就可根据指定的数据类型来启动特定的应用程序，并对指定数据执行相应的操作。

Extra属性

- Intent的Extra属性通常用于在多个Action之间进行数据交换，Intent的Extra属性值应该是一个Bundle对象，Bundle 对象就像一个 Map对象，它可以存入多个key-value对，这样就可以通过Intent在不同Activity之间进行数据交换了。

Flag属性

- Intent的Flag属性用于为该Intent添加一些额外的控制旗标，Intent 可调用addFlags()方法来添加控制旗标。
    - 例如：FLAG_ ACTIVITY CLEAR_ _TOP: 该Flag相当于加载模式中的singleTask,通过这种Flag启动的Activity将会把要启动的Activity之上的Activity 全部弹出Activity 栈。

应用资源

概述

- 物理存在形式分类
    - 界面布局文件:XML文件，文件中每个标签都对应于相应的View标签。
    - 程序源文件:应用中的Activity、 Service、 BroadcastReceiver、 ContentProvider 四大组件都是由Java或Kotin源代码来实现的。
    - 资源文件:主要以各种XML文件为主，还可包括`*.png`、`*jpg`、 `*.gif` 图片资源。
- 是否能通过R资源清单类访问分类
    - 无法通过R资源清单类访问的原生资源，保存在assets目录下。
    - 可通过R资源清单类访问的资源，保存在/res/目录下。

目录	存放的资源
/res/animator/	存放定义属性动画的XML文件
/res/anim/	存放定义补间动画的XML文件
/res/color/	存放定义不同状态下颜色列表的XML文件
/res/drawable/	存放适应不同屏幕分辨率的各种位图文件(如*.png、*.9.png、 *jpg、 *.gif 等)。此外，也可能编译成如下各种Drawable对象的XML文件。
	BitmapDrawable 对象
	NinePatchDrawable对象
	StateListDrawable对象
	ShapeDrawable对象
	AnimationDrawable 对象
	Drawable的其他各种子类的对象
/res/mipmap	主要存放适应不同屏幕分辨率的应用程序图标，以及其他系统保留的Drawable资源
/res/layout	存放各种用户界面的布局文件
/res/menu/	存放为应用程序定义各种菜单的资源，包括选项菜单、子菜单、上下文菜单资源
/res/raw/	存放任意类型的原生资源(比如音频文件、视频文件等)。在Java或Kotlin代码中可通过调用Resources 对象的openRawResource(int id)方法来获取该资源的二进制输入流。实际上，如果应用程序需要使用原生资源，也可把这些原生资源保存到/assets/目录下，然后在应用程序中使用AssetManager来访问这些资源
/res/values/	存放各种简单值的XML文件。这些简单值包括字符串值、整数值、颜色值、数组等。这些资源文件的根元素都是<resources../>,为该Tesources..>元素添加不同的子元素则代表不同的资源。如string/integer/bool、color 、array 等子元素 由于各种简单值都可定义在/res/values/目录下的资源文件中，如果在同一份资源文件中定义各种值，势必增加程序维护的难度。为此，Android 建议使用不同的文件来存放不同类型的值
	arrays.xml: 定义数组资源
	colors.xml:定义颜色值资源
	dimens.xml: 定义尺寸值资源
	strings.xml:定义字符串资源
	styles.xml:定义样式资源

目录	存放的资源
/res/xm/	存放任意的原生XML文件。这些XML文件可以在Java或Kotlin代码中使用Resources. getXML()方法进行访问

- /res/目录下的drawable 和mipmap子目录都可针对不同的分辨率建立对应的子目录，比如drawable-ldpi (低分辨率)、drawable-mdpi (中等分辨率)、drawable -hdpi (高分辨率)、drawable-xhdpi(超高分辨率)、 drawable-xxhdpi (超超高分辨率)等子目录。 这种做法可以让系统根据屏幕分辨
  率来选择对应子目录下的图片。如果开发时为所有分辨率的屏幕提供的是同一张图片，则可直接将该图片放在drawable目录下。

使用资源

- 在Android应用中使用资源可分为在Java或Kotlin代码和XML文件中使用资源，其中Java或Kotlin程序用于为Android应用定义四大组件;而XML文件则用于为Android应用定义各种资源。

在源程序中使用资源清单项

- 由于Android SDK会在编译应用时在R类中为/res/目录下所有资源创建索引项，因此在Java或Kotlin代码中访问资源主要通过R类来完成。
- 格式：`[<package_ name>. ]R.<resource type>.<resource name>`
  - `<package _name>`:指定R类所在包，实际上就是使用全限定类名。当然，如果在源程序中导入R类所在包，就可以省略包名。
  - `<resource_type>`: R类中代表不同资源类型的子类，例如string代表字符串资源。
  - `<resource_name>`:指定资源的名称。该资源名称可能是无后缀的文件名(如图片资源)，也可能是XML资源元素中由android:name属性所指定的名称。

在源代码中访问实际资源

- 背景：R资源清单类为所有的资源都定义了一个资源清单项，但这个清单项只是一个int类型的值，并不是实际的资源对象。有些时候，程序也需要使用实际的Android资源，为了通过资源清单项来获取实际资源，可以借助于Android提供的Resources类。
- Resources由Context调用getResources()方法来获取。
方法	说明
getXxx(int id)	根据资源清单ID来获取实际资源。
getAssets()	获取访问/assets/目录下资源的AssetManager对象。

在XML文件中使用资源

- 当定义XML资源文件时，其中的XML元素可能需要指定不同的值，这些值就可设置为已定义的资源项。
- 格式：`@[<package name>: ]<resource_ type>/ <resource_ name>`

Values

字符串、颜色、尺寸

资源类型	资源文件的默认名	对应于R类中的内部类的名称
字符串资源	/res/values/strings.xml	R.string
颜色资源	/res/values/colors.xml	R.color
尺寸资源	/res/values/dimens.xml	R.dimen

- Android中的颜色值是通过红(Red)、绿(Green)、蓝(Blue)三原色以及一个透明度(Alpha)值来表示的，颜色值总是以井号(#)开头，就是Alpha-Red-Green-Blue的形式。其中Alpha值可以省略，如果省略了Alpha值，那么该颜色默认是完全不透明的。
  - `#RGB`: 分别指定红、绿、蓝三原色的值(只支持0~f这16级颜色)来代表颜色。
  - `#ARGB`: 分别指定红、绿、蓝三原色的值(只支持0~f这16级颜色)及透明度(只支持0~f这16级透明度)来代表颜色。
  - `#RRGGBB`:分别指定红、绿、蓝三原色的值(支持00~ff这256级颜色)来代表颜色。
  - `#AARRGGBB`:分别指定红、绿、蓝三原色的值(支持00~ff这256级颜色)以及透明度(支持00~ff这256级透明度)来代表颜色。
- 定义
  - 字符串资源文件位于/res/values/目录下，字符串资源文件的根元素是<resources...>，,该元素里每个<string..>.子元素定义一一个字符串 常量，其中<string../>元素的 name 属性指定该常量的名称，<string../> 元素开始标签和结束标签之间的内容代表字符串值
    - ```
      <string name="hello">Hello World</ string>
      ```
  - 颜色 资源文件位于/res/values/目录下，.颜色资源文件的根元素是<resoures...>,.该元素里每个<color...>子元素定义一个字符串常量，其中<colo...元素的name属性指定该颜色的名称，<color..>元素开始标签和结束标签之间的内容代表颜色值.
  - 尺寸资源文件位于/res/values/目录下，尺寸资源文件的根元素是<resources..>，该元素里每个<dimen.../>子元素定义一个尺寸常量，其中<dimen.../>元素的name 属性指定该尺寸的名称，<dimen...>元素开始标签和结束标签之间的内容代表尺寸值.
  - Android 允许使用资源文件来定义boolean 常量。在/res/values/目录下增加一一个 bools.xml 文件，该文件的根元素也是<resources.../>，根元素内通过 <bool...>子元素来定义boolean常量，整形integer也类似

国际化

- 引入国际化的目的是为了提供自适应、更友好的用户界面。程序国际化指的是同一个应用在不同语言、国家环境下，可以自动呈现出对应的语言等用户界面，从而提供更友好的界面。
- 为了给这些消息提供不同国家、语言的版本，开发者需要为values 目录添加几个不同的语言国家版本。不同values文件夹的命名方式为: `values-语言代码-r国家代码`
- 在不同语言的国际化资源文件中所有消息的key是相同的，在不同国家、语言环境下，消息资源key对应的value不同。

数组

- Android采用位于/res/values目录下的arrays.xml文件来定义数组资源，定义数组时XML资源文件的根元素也是<resources.../>,该元素内可包含如下三种子元素。
  - <array....>子元素: 定义普通类型的数组，例如Drawable数组。
  - <string-array...>子元素: 定义字符串数组。
  - <integer-array..> 子元素:定义整型数组。
      ```

- 为了能在Java或Kotlin程序中访问到实际数组，Resources 类提供了方法。
    - String[] getStringArray(int id):根据资源文件中字符串数组资源的名称来获取实际的字符串数组。
    - int[] getIntArray(int id):根据资源文件中整型数组资源的名称来获取实际的整型数组。
    - TypedArray obtainTypedArray(int id):根据资源文件中普通数组资源的名称来获取实际的普通数组。
        - TypedArray代表一个通用类型的数组，该类提供了getXxx(int index)方法来获取指定索引处的数组元素。

Drawable

- 图片资源是最简单的Drawable 资源，只要把`*.png`、`*jpg`、`*.gif`等格式的图片放入/res/drawable-xxx目录下，Android SDK就会在编译应用中自动加载该图片，并在R资源清单类中生成该资源的索引。
    - 注意：Android要求图片资源的文件名必须符合Java或Kotlin标识符的命名规则;否则，Android SDK无法为该图片在R类中生成资源索引。
- 在程序中获得实际的Drawable对象：Resources 提供了getDrawable(int id)方法，该方法即可根据Drawable资源在R资源清单类中的ID来获取实际的Drawable对象。

StateListDrawable资源

- StateListDrawable用于组织多个Drawable对象。当使用StateListDrawable作为目标组件的背景、前景图片时，StateListDrawable对象所显示的Drawable对象会随目标组件状态的改变而自动切换。
- StateListDrawable对象的XML文件的根元素为<selector../>,该元素可以包含多个<item.../>元素
    - android:color或android:drawable:指定颜色或Drawable对象。
    - android:state_ xxx: 指定一个 特定状态。比如激活、已勾选、可勾选、可用、开始
- StateListDrawable不仅可以让文本框里文字的颜色随文本框状态的改变而切换，也可让按钮的背景图片随按钮状态的改变而切换。StateListDrawable的功能非常灵活，它可以让各种组件的背景、前景随状态的改变而切换。

LayerDrawable资源

- LayerDrawable包含一个Drawable数组，系统将会按这些Drawable对象的数组顺序来绘制它们，索引最大的Drawable对象将会被绘制在最上面。
- LayerDrawable对象的XML文件的根元素为<layer-list...>,该元素可以包含多个<item..>元素，item元素的属性：
    - android:drawable: 指定作为LayerDrawable元素之一的 Drawable对象。
    - `android:id`:为该Drawable对象指定一个标识。
    - android:buttom|top|left|right: 它们用于指定一个长度值，用于指定将该Drawable对象绘制到目标组件的指定位置。

ShapeDrawable资源

- ShapeDrawable用于定义一个基本的几何图形(如矩形、圆形、线条等)
- ShapeDrawable的XML文件的根元素是<shape.../>元素，子元素有corners、gradient、padding、size、solid、stroke该元素可指定属性
    - android:shape=["rectangle" |"oval" I "line" I "ring"]:指定定义哪种类型的几何图形。

ClipDrawable资源

- ClipDrawable代表从其他位图上截取的一个“图片片段”。在XML文件中定义ClipDrawable对象使用<clip../>元素
  - clip元素的属性
    - android:drawable:指定截取的源Drawable对象。
    - android:clipOrientation:指定截取方向，可设置水平截取或垂直截取。
    - android:gravity:指定截取时的对齐方式。
  - 使用ClipDrawable对象时可调用setLevel(int level)方法来 设置截取的区域大小，当level为0时，截取的图片片段为空;当level为10000时，截取整张图片。

AnimationDrawable资源

- AnimationDrawable代表一个动画Android既支持传统的逐帧动画(类似于电影方式，一张图片、一张图片地切换)， 也支持通过平移、变换计算出来的补间动画。
- 定义补间动画的XML资源文件以<set...>元素作为根元素，该元素内可以指定如下4个元素。
属性	说明
alpha	设置透明度的改变。
scale	设置图片进行缩放变换。
translate	设置图片进行位移变换。
rotate	设置图片进行旋转。
  - 定义动画的XML资源应该放在/res/anmi/路径下，Android Studio 创建Android应用时，默认不会包含该路径，开发者需要自行创建该路径。
  - 元素都可指定一个android:interpolator属性，该属性指定动画的变化速度，可以实现匀速、正加速、负加速、无规则变加速
  - 如果程序想让<set.../>元素下所有的变换效果使用相同的动画速度，则可指定android:shareInterpolator="true"
  - 在Java或Kotlin代码中获取实际的Animation对象，可调用AnimationUtils的loadAnimation(Context ctx, int resId)方法。使用View的startAnimation（ ）方法可以执行动画。

屏幕自适应

- 屏幕资源需考虑的方面。
  - 屏幕尺寸:屏幕尺寸可分为small (小屏幕)、normal (中等屏幕)、large (大屏幕)、xlarge (超大屏幕) 4种。
    - 屏幕分辨率：Android 默认把drawable 目录(存放图片等Drawable 资源的目录)分为drawable-ldpi、drawable- mdpi、drawable-hdpi、 drawable xhdpi、drawable-xxhdpi 这些子目录，它们正是用于为不同分辨率屏幕准备图片的。
  - 屏幕分辨率: 屏幕分辨率可分为ldpi (低分辨率)、mdpi (中等分辨率)、hdpi (高分辨率)、xhdpi (超高分辨率)、xxhdpi (超超高分辨率) 5种。
    - 屏幕尺寸:可以为layout、values 等目录增加后缀small、normal、 large 和xlarge,分别为不同尺寸的屏幕提供相应资源。
  - 屏幕方向:屏幕方向可分为land (横屏)和port (竖屏)两种。
- 最低屏幕要求

    - xlarge屏幕尺寸至少需要960dpx720dp。
    - large屏幕尺寸至少需要640dpx480dp。
    - normal屏幕尺寸至少需要470dpx320dp。
    - small屏幕尺寸至少需要426dpx 320dp。

PropertyAnimation 资源

- Animator代表一个属性动画，但它只是一个抽象类，通常会使用它的子类: AnimatorSet、ValueAnimator、ObjectAnimator、TimeAnimator。
- 定义属性动画的XML资源文件能以如下三个元素中的任意一个作为根元素。
    - <set.../>: 它是一一个父元素，用于包含<objectAnimator...>.、<animator...>或<set..>.子元素，该元素定义的资源代表AnimatorSet对象。
    - <objectAnimator.../>: 用于定义ObjectAnimator动画。
    - <animator../>: 用于定义ValueAnimator动画。

原始XML资源

- Android 应用有一些初始化的配置信息、应用相关的数据资源需要保存，一般推荐使用XML文件来保存它们，这种资源就被称为原始XML资源。
- 定义：原始XML资源一般保存在/res/xml/路径下，Android Studio创建Android应用时,/res/目录下并没有包含xml子目录，开发者应该自行手动创建xml子目录。Android应用对原始XML资源没有任何特殊的要求，只要它是一份格式良好的XML文档即可。
- 为了在Java或Kotlin程序中获取实际的XML文档，可以通过Resources的方法来实现。
    - XmlResourceParser getXml(int id):获取XML文档，并使用一个XmlPullParser来解析该XML文档，该方法返回一个解析器对象(XmlResourceParser是XmlPullParser的子类)。
    - InputStream openRawResource(int id):获取XML文档对应的输入流。
    - 大部分时候直接调用getXml(int id)方法来获取XML文档，并对该文档进行解析。Android默认使用内置的Pull解析器来解析XML文件。
    - 除使用Pull解析之外，也可使用DOM或SAX对XML文档进行解析。一般的Java或Kotlin应用会使用JAXP API来解析XML文档。
        - Pull解析方式有点类似于SAX解析，它们都采用事件驱动的方式来进行解析。当Pull 解析器开始解析之后，开发者可不断地调用Pull解析器的next()方法获取下一一个解析事件(开始文档、结束文档、开始标签、结束标签等)，当处于某个元素处时，可调用XmlPullParser的getAttributeValue()方法来获取该元素的属性值，也可调用XmlPullParser的nextText( ) 方法来获取文本节点的值。
        - 如果开发者希望使用DOM、SAX或其他解析器来解析XML资源，那么可调用openRawResource (int id)方法来获取XML资源对应的输入流，这样即可自行选择解析器来解析该XML资源了。

Layout

- Layout资源文件应放在/res/layout/目录下，Layout 资源文件的根元素通常是各种布局管理器，比如LinearLayout、TableLayout、FrameLayout等。

Menu

- Android菜单资源文件放在/res/menu目录下，菜单资源的根元素通常是<menu...>元素，<menu../>元素无须指定任何属性。

Style

- Style：Android样式包含一组格式， 为一个组件设置使用某个样式时，该样式所包含的全部格式将会应用于该组件。
- Android的样式资源文件也放在/res/values/目录下，样式资源文件的根元素是<resources..>元素，该元素内可包含多个<style...>子元素，每个<style...> 子元素定义一个样式。
    - <style...>元素属性
        - name:指定样式的名称。
        - parent: 指定该样式所继承的父样式。当继承某个父样式时，该样式将会获得父样式中定义的全部格式。当然，当前样式也可以覆盖父样式中指定的格式。
    - 在<style...>元素内可包含多个<item...>子元素，每个<item...> 子元素定义一一个格式项。

Theme

- 主题资源的XML文件通常也放在/res/values/目录下，主题资源的XML文件同样以<resources...>元素作为根元素，同样使用<style...> 元素来定义主题。
- 主题与样式的区别主要体现在:
    - 主题不能作用于单个的View组件，主题应该对整个应用中的所有Activity起作用，或对指定的Activity起作用。
    - 主题定 义的格式应该是改变窗口外观的格式， 例如窗口标题、窗口边框等。
- 使用
    - 在代码中使用setTheme()方法
    - 在AndroidManifest.xml文件中对指定应用、指定Activity 应用主题，使用android:theme属性
- Android中提供了几种内置的主题资源，这些主题通过查询Android.R.style类可以看到。

Attribute

- 当在XML布局文件中使用Android系统提供的View组件时，开发者可以指定多个属性，这些属性可以很好地控制View组件的外观行为。如果用户开发的自定义View组件也需要指定属性，可以使用属性资源
- 属性资源文件也放在/res/values目录下，属性资源文件的根元素也是<resources../>元素，该元素包含如下两个子元素。
    - attr子元素:定义一个属性。
    - declare-styleable 子元素:定义一个styleable 对象，每个styleable 对象就是一组attr属性的集合。

原始资源

- 类似于声音文件及其他各种类型的文件，只要Android没有为之提供专门的支持，这种资源都被称为原始资源。Android 的原始资源可以放在如下两个地方。
- 位置
    - 位于/res/raw/目 录下，Android SDK会处理该目录下的原始资源，AndroidSDK 会在R清单类中为该目录下的资源生成一个索引项。
        - Android SDK会为位于/res/raw/目录下的资源在R清单类中生成一个索引项
    - 位于/assets/目录下，该目录下的资源是更彻底的原始资源。Android 应用需要通过AssetManager来管理该目录下的原始资源。
        - AssetManager是一个专门管理/assets/目录下原始资源的管理器类, AssetManager提供了如下两个常用方法来访问Assets资源。
            - InputStream open(String fileName):根据文件名来获取原始资源对应的输入流。

        - AssetFileDescriptor openFd(String fileName): 根据文件名来获取原始资源对应的AssetFileDescriptor。AssetFileDescriptor 代表了一项原始资源的描述，应用程序可通过AssetFileDescriptor来获取原始资源。

数据存储和IO

- 数据的持久化

SharedPreferences

概述

- SharedPreferences保存的数据主要是类似于配置信息格式的数据，因此它保存的数据主要是简单类型的key-value对。
- SharedPreferences接口主要负责读取应用程序的Preferences数据
- SharedPreferences方法

方法	说明
boolean contains(String key)	判断SharedPreferences是否包含特定key的数据。
Map<String, ?> getAll()	获取SharedPreferences数据里全部的key-value对。
getXxx(String key, xxx defValue)	获取SharedPreferences 数据里指定key对应的value。如果该key不存在，则返回默认值defValue。其中xxX可以是boolean、float、int、 long、String等各种基本类型的值。

- SharedPreferences接口并没有提供写入数据的能力，通过SharedPreferences. Editor才允许写入，SharedPreferences调用edit()方法即可获取它所对应的Editor 对象。
    - SharedPreferences. Editor方法

SharedPreferences.Editor clear()	清空SharedPreferences里所有数据。
SharedPreferences.Editor putXxx(String key, xXx value)	向SharedPreferences 存入指定key对应的数据。其中xxx可以是boolean、float、 int、long、String等各种基本类型的值。
SharedPreferences.Editor remove(String key)	删除SharedPreferences里指定key对应的数据项。
boolean apply()	当Editor编辑完成后，调用该方法提交修改。
commit()	与applyO功能类似，但commit()会立即提交修改:而apply()在后台提交修改，不会阻塞前台!

- SharedPreferences本身是一个**接口**，程序无法直接创建SharedPreferences 实例，只能通过Context提供的getSharedPreferences(String name，int mode)方法来获取SharedPreferences实例
    - mode参数可选值
        - Context.MODE PRIVATE:指定该SharedPreferences数据只能被本应用程序读写。

        - Context.MODE WORLD_ READABLE: 指定该SharedPreferences数据能被其他应用程序读，但不能写。
        - ContextMODE WORLD_WRITEABLE:指定该SharedPreferences 数据能被其他应用程序读写。

存储位置和格式

- SharedPreferences数据保存在`/data/data/<package name>/shared _prefs`目录下，SharedPreferences数据以XML格式保存。
    - SharedPreferences 数据文件的根元素是<map..>元素，该元素里每个子元素代表一一个key-value对，当value是整数类型时，使用<int...> 子元素;当value是字符串类型时，使用<string.../> 子.....依此类推。

File

- Context提供了方法来处理应用程序的数据文件夹里的文件IO流。
方法	说明
FileInputStream openFileInput(String name)	打开应用程序的数据文件夹下的name文件对应的输入流。
FileOutputStream openFileOutput(String name, int mode)	打开应用程序的数据文件夹下的name文件对应的输出流。
getDir(String name, int mode)	在应用程序的数据文件夹下获取或创建name对应的子目录。
File getFilesDir()	获取应用程序的数据文件夹的绝对路径。
String[] fileList()	返回应用程序的数据文件夹下的全部文件。
deleteFile(String name)	删除应用程序的数据文件夹下的指定文件。

        - mode:指定打开文件的模式，该模式的可选项为
            - MODE _PRIVATE:该文件只能被当前程序读写。
            - MODE_APPEND:以追加方式打开该文件，应用程序可以向该文件中追加内容。
            - MODE_WORLD READABLE:该文件的内容可以被其他程序读取。
            - MODE_WORLD WRITEABLE:该文件的内容可由其他程序读写。

SD卡

- Secure Digital，缩写为SD，全名为Secure Digital Memory Card
- 读写SD卡步骤
    - 请求动态获取读写SD卡的权限，只有当用户授权读写SD卡时才执行读写。
    - 调用Environment的getExternalStorageDirectory()方法来获取外部存储器，也就是SD卡的目录。
    - 使用FileInputStream、FileOutputStream、 FileReader 或FileWriter读写SD卡里的文件。

SQLite

- Android系统集成了一个轻量级的数据库: SQLite，SQLite 是一个嵌入式的数据库引擎，专门适用于资源有限的设备(如手机、PDA等)上适量数据存取。
- SQLite支持绝大部分SQL92语法，也允许开发者使用SQL语句操作数据库中的数据，SQLite数据只是一个文件
- Android提供了SQLiteDatabase 代表一个数据库(底层就是一个数据库文件)，一旦应用程序获得了代表指定数据库的SQLiteDatabase对象，接下来就可通过SQLiteDatabase对象来管理、操作数据库了。
- 如果真的需要在App中频繁地操作SQLite 数据库，则强烈建议使用ORM工具，比如OrmLite、GreenDao、 LitePal 等。
- SQLiteDatabase方法

	方法	说明
打开或创建	static SQLiteDatabase openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)	打开path文件所代表的SQLite数据库。
	static SQLiteDatabase openOrCreateDatabase(**File** file, SQLiteDatabase.Cursor Factory factory)	打开或创建(如果不存在)file 文件所代表的SQLite数据库。
	static SQLiteDatabase openOrCreateDatabase(**String** path,SQLiteDatabase. Cursor Factoryfactory)	打开或创建(如果不存在)path 文件所代表的SQLite数据库。
操作	execSQL(String sql, Object[] bindArgs)	执行带占位符的SQL语句。
	execSOL(String sal)	执行SOL语句。
	insert(String table, String nullColumnHack, ContentValues values)	向指定表中插入数据。
	update(String table, ContentV alues values, String whereClause, String[] whereArgs)	更新指定表中的特定数据。
	delete(String table, String whereClause, String[] whereArgs)	删除指定表中的特定数据。
	Cursor query(String table, String[] columns, String whereClause, String[] whereArgs, String groupBy, String having, String orderBy)	对指定数据表执行查询。
	Cursor query(String table, String[] columns, String whereClause, String[] whereArgs, String groupBy, String having, String orderBy, String limit)	对指定数据表执行查询。limit 参数控制最多查询几条记录(用于控制分页的参数)。
	Cursor query(boolean distinct, String table, String[] columns, String whereClause, String[]whereArgs, String groupBy, String having, String orderBy, String limit)	对指定数据表执行查询。其中第-一个参数控制是否去除重复值。
	rawQuery(String sql, String] selectionArgs)	执行带占位符的SQL查询
	beginTransaction()	开始事务。
	endTransaction()	结束事务。
- Cursor对象，Android 中的Cursor类似于JDBC的ResultSet, Cursor同样提供了方法来移动查询结果的记录指针。

-

方法	说明
move(int offset)	将记录指针向上或向下移动指定的行数。offset 为正数就是向下移动;为负数就是向上移动。
boolean moveToFirst()	将记录指针移动到第一行，如果移动成功则返回true。
boolean moveToLast()	将记录指针移动到最后一行，如果移动成功则返回true。
boolean moveToNext()	将记录指针移动到下一行，如果移动成功则返回true。
boolean moveToPosition(int position)	将记录指针移动到指定行,如果移动成功则返回true。
boolean moveToPrevious()	将记录指针移动到上一行，如果移动成功则返回true。

  - SQLiteDatabase.CursorFactory:该参数是一个用于返回Cursor的工厂，如果指定该参数为null, 则意味着使用默认的工厂。

SQLiteOpenHelper类

- SQLiteOpenHelper是Android 提供的一个管理数据库的工具类，可用于管理数据库的创建和版本更新。一般的用法是创建SQLiteOpenHelper的子类，并扩展它的onCreate (SQLiteDatabase db)和onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)方法。
-

方法	说明
synchronized sQLiteDatabase getReadableDatabase()	以读写的方式打开数据库对应的SQLiteDatabase对象。
synchronized SQLiteDatabase getWritableDatabase()	以写的方式打开数据库对应的SQLiteDatabase对象。
abstract void onCreate(SQLiteDatabase db)	当第一次创建数据库时回调该方法。
abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	当数据库版本更新时回调该方法。
synchronized void close()	关闭所有打开的SQLiteDatabase对象。

  - onCreate(SQLiteDatabase db):用于初次使用软件时生成数据库表。当调用SQLiteOpenHelper的getWritableDatabase()或getReadableDatabase()方法获取用于操作数据库的SQLiteDatabase实例时，如果数据库不存在，Android系统会自动生成一个数据库，然后调用onCreate()方法，该方法在**初次生成数据库表时才会被调用**。重写onCreate( ) 方法时，可以生成数据库表结构，也可以添加应用使用到的一些初始化数据。
  - onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion):用于升级软件时更新数据库表结构，此方法在数据库的版本发生变化时会被调用，该方法被调用时oldVersion代表数据库之前的版本号，newVersion代表数据库当前的版本号。当程序创建SQLiteOpenHelper对象时，必须指定一个version 参数，该参数就决定了所使用的数据库的版本,只要某次创建

SQLiteOpenHelper时指定的数据库版本号高于之前指定的版本号，系统就会自动触发onUpgrade()方法

- 数据库的磁盘空间满了，数据库就只能读而不能写
  - getWritableDatabase( ) 方法以写的方式打开数据库,倘若使用getWritableDatabase()打开数据库就会出错。
  - getReadableDatabase()方法先以读写的方式打开数据库，如果数据库的磁盘空间满了，就会打开失败，当打开失败后会继续尝试以只读的方式打开数据库。

操作数据库

- SQLiteDatabase 的execSQL()方法可执行任意的SQL语句，包括带占位符的SQL语句。但该方法**没有返回值**.
  - 如果需要执行查询语句，则可调用SQLiteDatabase的rawQuery(String sql, StringD] selectionArgs)方法。
  - ```
    db.execSQL ("insert into news inf values (null, ?,?)", new String[] {title, content});
    ```

- 流程
  1. 通过SQLiteOpenHelper的子类获取SQLiteDatabase对象，它代表了与数据库的连接。
  2. 调用SQLiteDatabase的方法来执行SQL语句。
  3. 操作SQL语句的执行结果，比如用CursorRecyclerViewAdapter封装Cursor。
  4. 关闭SQLiteDatabase,回收资源。

sqlite3工具

- 在Android SDK的platform-tools目录下提供了一个sqlite3.exe文件，它是一个简单的SQLite数据库管理工具，类似于MySQL提供的命令行窗口。
  - ```
    sqlite3 f: /my.db3
    ```
  - SQLite数据库还支持绝大部分常用的SQL语句，可以在命令行窗口中运行各种DDL、DML、查询语句来测试它们。
  - SQLite 内部只支持NULL、INTEGER、REAL (浮点数)、TEXT (文本)和BLOB (大二进制对象)这5种数据类型，但实际上SQLite完全可以接受varchar(n)、 char(n)、decimal(p,s)等数据类型，只不过SQLite会在运算或保存时将它们转换为上面5种数据类型中相应的类型。
  - SQLite允许把各种类型的数据保存到任何类型字段中，开发者可以不用关心声明该字段所使用的数据类型。例如，程序可以把字符串类型的值存入INTEGER类型的字段中，也可以把数值类型的值存入布尔类型的字段....但有种情况例外:定义为INTEGER PRIMARY KEY的字段只能存储64位整数，当向这种字段中保存除整数以外的其他类型的数据时，SQLite 会产生错误。

事务

方法	说明
beginTransaction()	开始事务。
endTransaction()	结束事务。
inTransaction()	如果当前上下文处于事务中，则返回true;否则返回false。

- 当程序执行endTransaction()方法时将会结束事务，提交和回滚取决于SQLiteDatabase是否调用了setTransactionSuccesful()方法来设置事务标志，如果程序在事务执行中调用该方法设置了事务成功则提交事务;否则程序将会回滚事务。

Gesture

- 手势，是指用户手指或触摸笔在触摸屏上的连续触碰行为，手势这种连续的触碰会形成某个方向上的移动趋势，也会形成一个不规则的几何图形。
  - 比如在屏幕上从左至右划出的一个动作，就是手势;
    - Android提供了手势检测，并为手势检测提供了相应的监听器。
  - 再比如在屏幕上画出一个圆圈也是手势。
    - Android允许开发者添加手势，并提供了相应的API识别用户手势。

手势控制

- Android为手势检测提供了一个GestureDetector 类，GestureDetector 实例代表了一个手势检测器，创建GestureDetector时需要传入一 个GestureDetector.OnGestureListener 实例，负责对用户的手势行为提供响应。
- GestureDetector.OnGestureListener 方法

方法	说明
boolean onDown(MotionEvent e)	当触碰事件按下时触发该方法。
boolean onFling(MotionEvent el, MotionEvent e2, float velocityX, float velocityY)	当用户手指在触摸屏上“拖过”时触发该方法。其中velocityX、velocityY 代表“拖过”动作在横向、纵向上的速度。
abstract void onLongPress(MotionEvent e)	当用户手指在屏幕上长按时触发该方法。
boolean onScroll(MotionEvent el, MotionEvent e2, float distanceX, float distanceY)	当用户手指在屏幕上“滚动"时触发该方法。
void onShowPress(MotionEvent e)	当用户手指在触摸屏上按下，而且还未移动和松开时触发该方法。
boolean onSingleTapUp(MotionEvent e)	用户手指在触摸屏上的轻击事件将会触发该方法。

- 使用流程
  1. 创建一个GestureDetector对象。创建该对象时必须实现GestureDetector.OnGestureListener监听器接口。
  2. 为应用程序的Activity (偶尔也可为特定组件)的TouchEvent事件绑定监听器，在事件处理中指定把Activity (或特定组件)上的TouchEvent事件交给GestureDetector处理。
- 多点触碰
  - Android也为多点触碰提供了支持，处理多点触碰也通过重写onTouch方法来实现，通过该方法的MotionEvent参数的getPointerCount0方 法可判断触碰点的数量。
  - 在处理多点触碰事件时，程序要通过MotionEvent的getActionMasked()方法来判断触碰事件的类型(按下、移动、松开等)。
  - 根据两个手指的距离(捏合)来计算缩放比

增加手垫

- Android除提供手势检测之外，还允许应用程序把用户手势(多个持续的触摸事件在屏幕上形成特定的形状)添加到指定文件中，以备以后使用一如果程序需要，当用户下次再次画出该手势时，系统将可识别该手势。
- Android使用GestureLibrary来代表手势库，并提供了GestureLibraries 工具类来创建手势库。
- GestureLibraries方法
方法	说明
static GestureLibrary fromFile(String path)	从path代表的文件中加载手势库。
static GestureLibrary fromFile(File path)	从path代表的文件中加载手势库。
static GestureLibrary fromPrivateFile(Context context, String name)	从指定应用程序的数据文件夹的name文件中加载手势库。
static GestureLibrary fromRawResource(Context context, int resourceId)	从resourceId所代表的资源中加载手势库。
  - 获得了GestureLibrary对象之后，该对象提供了如下方法来添加手势、识别手势。
方法	说明
void addGesture(String entryName, Gesture gesture)	添加一一个名为entryName的手势。
Set getGestureEntries()	获取该手势库中的所有手势的名称。
ArrayList getGestures(String entryName)	获取entryName名称对应的全部手势。
ArrayList recognize(Gesture gesture)	从当前手势库中识别与gesture匹配的全部手势。
void removeEntry(String entryName)	删除手势库中entryName对应的手势。
void removeGesture(String entryName, Gesture gesture)	删除手势库中entryName、gesture对应的手势。
booleansave()	当向手势库中添加手势或从中删除手势后调用该方法保存手势库。
- GestureOverlayView：提供了一个专门的手势编辑组件:, 该组件就像一个“绘图组件”，只是用户在组件上绘制的不是图形，而是手势。
  - 为了监听GestureOverlayView 组件上的手势事件，Android 为GestureOverlayView 提供了OnGestureListener、OnGesturePerformedListener 、OnGesturingListener三个监听器接口
  - android:gestureStrokeType 参数：该参数控制手势是否需要多笔完成。可设置为single和multiple

recognize

- GestureLibrary 提供了recognize(Gesture ges)方法来识别手势，该方法将会返回该手势库中所有与ges匹配的手势一两 个手势的图形越相似，相似度越高。
- recognize(Gesture ges)方法的返回值为ArrayList, 其中Prediction 封装了手势的匹配信息，Prediction对象的name属性代表了匹配的手势名，score属性代表了手势的相似度。

TTS

- TTS:text to speech
- Android提供了自动朗读支持。自动朗读支持可以对指定文本内容进行朗读，从而发出声音不仅如此，Android 的自动朗读支持还允许把文本对应的音频录制成音频文件，方便以后播放。
- 使用流程
    1. 通过TextToSpeech类的构造器TextToSpeech(Context context, TextToSpeech.OnInitListener listener)创建一个TextToSpeech对象
    2. 调用TextToSpeech 的setLanguage(Locale loc)方法来设置该TTS语音引擎应使用的语言、国家选项。
        - 如果调用setLanguage(Locale loc)的返回值是"TextToSpeech.LANG COUNTRY_ AVAILABLE",则说明当前TTS系统可以支持所设置的语言、国家选项。
    3. 通过TexttoSpeech朗读文本
方法	说明
speak(CharSequence text, int queueMode, Bundle params, String utteranceId)	把text 文字内容转换为音频，播放转换的音频
synthesizeToFile (CharSequence text, Bundle params, File file, String utteranceId)	把text 文字内容转换为音频，把转换得到的音频保存成声音文件
        - queueMode：指定TTS的发音队列模式，该参数支持如下两个常量。
            - TextToSpeech.QUEUE_ FLUSH:如果指定该模式，当TTS调用speak()方法时，它会中断当前实例正在运行的任务(也可以理解为清除当前语音任务，转而执行新的语音任务)。
                - TextToSpeech.QUEUE ADD:如果指定为该模式，当TTS调用speak()方法时，会把新的发音任务添加到当前发音任务列队之后,也就是等任务队列中的发音任务执行完成后再来执行speak()方法指定的发音任务。
    4. 当程序用完了TextToSpeech对象之后,在Activity的OnDestroy()方法中调用TextToSpeech的shutdown()来关闭TextToSpeech,释放TextToSpeech所占用的资源。

ContentProvider

- 为了在应用程序之间交换数据，Android 提供了ContentProvider, 它是不同应用程序之间进行数据交换的标准API,当一个应用程序需要把自己的数据暴露给其他程序使用时，该应用程序就可通过提供ContentProvider来实现;其他应用程序就可通过ContentResolver来操作ContentProvider暴露的数据。
    - ContentProvider 以某种Uri的形式对外提供数据,其他应用程序使用ContentResolver 根据Uri去访问操作指定数据。
- ContentProvider也是Android 应用的四大组件之一， 与Activity、 Service 、BroadcastReceiver相似，它们都需要在AndroidManifest.xml文件中进行配置。

- 一旦某个应用程序通过ContentProvider 暴露了自己的数据操作接口，那么不管该应用程序是否启动，其他应用程序都可通过该接口来操作该应用程序的内部数据，包括增加数据、删除数据、修改数据、查询数据等。
- 定义流程
  1. 定义自己的ContentProvider类，该类需要继承Android提供的ContentProvider基类。
  2. 向Android 系统注册这个“网站”， 也就是在AndroidManifest.xml 文件中注册这个ContentProvider,就像注册Activity一样。 注册ContentProvider时需要为它绑定一个Uri。
     - ```
       <!--在Application元素下添加，name属性指定ContentProvider类，authorities就相当于为该ContentProvider指定域名-- >
       <provider
           android:name=".DictProvider"
           android:authorities="org.hello'
           android:exported="true"/>
       ```
方法	说明
boolean onCreate()	该方法在ContentProvider创建后会被调用，当其他应用程序第一次访问ContentProvider时，该ContentProvider会被创建出来，并立即回调该onCreate0方法。
Uri insert(Uri uri, ContentValues values)	根据该Uri插入values对应的数据。
int delete(Uri uri, String selection, String[] selectionArgs)	根据Uri删除selection 条件所匹配的全部记录。
int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)	根据Uri修改selection条件所匹配的全部记录。
Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)	根据Uri查询出selection条件所匹配的全部记录，其中projection就是一个列名列表，表明只选择出指定的数据列。
String getType(Uri uri)	该方法用于返回当前Uri所代表的数据的MIME类型。如果该Uri对应的数据可能包括多条记录,那么MIME类型字符串应该以vnd.android.cursor dir/开头;如果该Uri对应的数据只包含一条记录，那么MIME类型字符串应该以vnd. android.cursor.item/开头。

URI

- 统一资源标志符（英语：Uniform Resource Identifier，缩写：URI）在电脑术语中是用于标志某一互联网资源名称的字符串。
  - 统一资源定位符（英语：Uniform Resource Locator，缩写：URL，或称统一资源定位器、定位地址、URL地址）
- Uri是ContentResolver和ContentProvider进行数据交换的标识。
- ContentProvider要求的Uri
       ```

    - `content://org.hello/words`
        - content://:这个部分是Android的ContentProvider 规定的，就像上网的协议默认是http://一样。 暴露ContentProvider、访问ContentProvider的协议默认是content://。
        - org.hello:这个部分就是ContentProvider 的authorities。系统就是由这个部分来找到操作哪个ContentProvider的。只要访问指定的ContentProvider,这个部分就是固定的。
        - words:资源部分(或者说数据部分)。当访问者需要访问不同资源时，这个部分是动态改变的。
- 为了将一个字符串转换成Uri, Uri 工具类提供了parse()静态方法。
    - `Uri uri = Uri.parse("content://org.hello/word/2") ;`

Uri的有效性判断

- 为了确定该ContentProvider实际能处理的Uri, 以及确定每个方法中Uri参数所操作的数据，Android系统提供了**UriMatcher**工具类。
- UriMatcher工具类方法。

方法	说明
void addURI(String authority, String path, int code)	该方法用于向UriMatcher对象注册Uri。其中authority和path组合成一个Uri,而code则代表该Uri对应的标识码。
int match(Uri uri)	根据前面注册的Uri来判断指定Uri对应的标识码。如果找不到匹配的标识码，该方法将会返回-1。

操作Uri字符串

- Android 提供了一个**ContentUris**工具类，它是一个操作Uri字符串的工具类
- ContentUris方法

方法	说明
withAppendedId(uri, id)	用于为路径加上ID部分。
parseId(uri)	用于从指定Uri中解析出所包含的ID值。

ContentResolver

- Context提供了getContentResolver()方法获取该应用默认的ContentResolver对象
- ContentResolver方法

方法	说明
Uri insert(Uri uri, ContentValues values)	根据该Uri对应的ContentProvider插入values对应的数据。
int delete(Uri uri, String selection, String[] selectionArgs)	根据Uri对应的ContentProvider删除selection 条件所匹配的全部记录。
int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)	根据Uri对应的ContentProvider修改selection条件所匹配的全部记录。
Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)	根据Uri对应的ContentProvider查询出selection条件所匹配的全部记录，其中projection 就是一个列名列表，表明只选择出指定的数据列。

- 一般来说，ContentProvider 是单实例模式的，当多个应用程序通过ContentResolver 来操作ContentProvider提供的数据时, ContentResolver 调用的数据操作将会委托给同一个ContentProvider处理。

开发ContentProvider

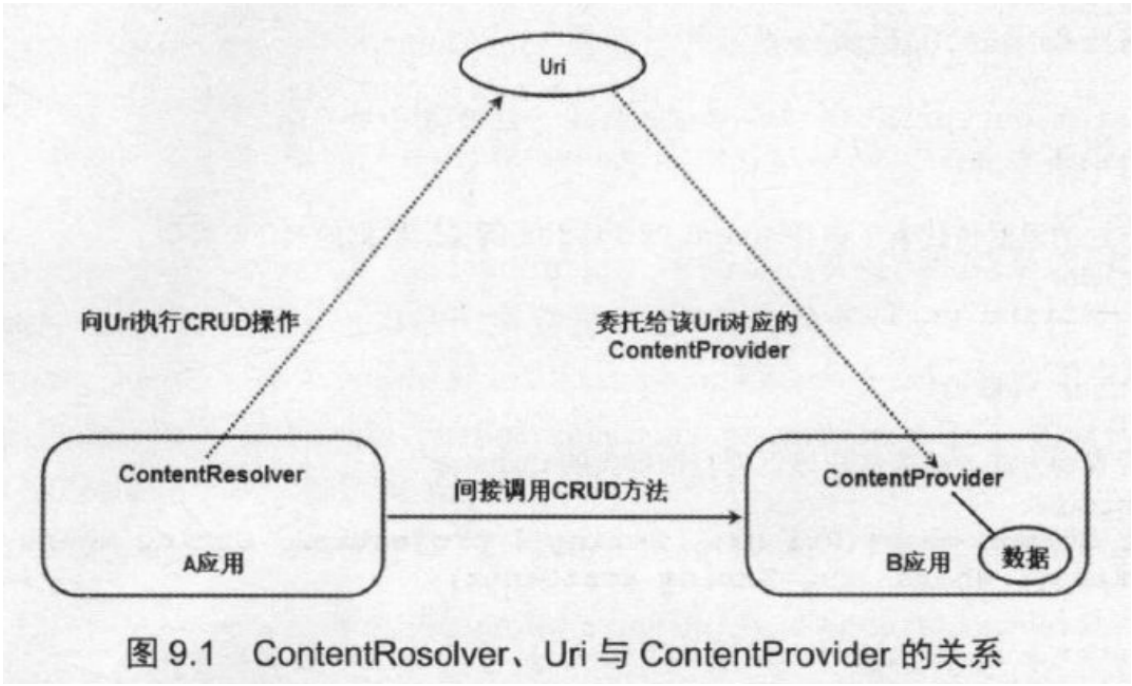
- 

图 9.1 ContentRosolver、Uri 与 ContentProvider 的关系

- 步骤
  1. 开发一个ContentProvider子类，该子类需要实现query0、insert()、 update()和 delete)等方法。
     - 实现的方法并不是给该应用本身调用的，而是供其他应用来调用的。
     - 如何实现方法，完全由程序员决定
  2. 在AndroidManifest.xml文件中注册该ContentProvider,指定android:authorities属性。

配置ContentProvider

- Android应用要求所有应用程序组件(Activity、Service、 ContentProvider 、BroadcastReceiver)都必须显式进行配置。
- 只要为<applcation..>元素添加<provier../>子元素即可配置ContentProvider，provier子元素属性

属性	说明
name	指定该ContentProvider的实现类的类名。
authorities	指定该ContentProvider对应的Uri(相当于为该ContentProvider分配一个域名)。
android:exported	指定该ContentProvider是否允许其他应用调用。如果将该属性设为false,那么该ContentProvider将不允许其他应用调用。
readPermission	指定读取该ContentProvider所需要的权限。也就是调用ContentProvider的query()方法所需要的权限
writePermission	指定写入该ContentProvider 所需要的权限。也就是调用ContentProvider的insert()、delete()、 update()方法所需要的权限。
permission	该属性相当于同时配置readPermission和writePermission两个权限。

  - 如果不配置上面的readPermission、 writePermission、 permission 权限，则表明没有权限限制.那意味着该ContentProvider可以被所有App访问。
  - 如果配置了permission，为了让Android系统知道该权限，还必须在AndroidManifest.xml 文件的根元素下(与<applcatio...>元素同级)增加如下配置。
    - ```
      <!--指定该应用暴露了一个权限-->
      <permission android:name="permission属性的属性值"
      android:protectionLevel="normal" />
      ```
    - ContentProvider声明了使用需要权限，因此在要使用到该ContentProvider的应用中必须声明本应用所需的权限。
      - ```
        <uses-permission android: name= "permission属性的属性值"/>
        ```

Android的ContentProvider

- Android 系统本身提供了大量的ContentProvider, 例如联系人信息、系统的多媒体信息等
- 为了操作系统提供的ContentProvider, 需要了解该ContentProvider 的Uri,以及该ContentProvider所操作的数据列的列名，可以通过查阅Android官方文档来获取这些信息。

Contacts的ContentProvider

- Uri
Uri	说明
ContactsContract.Contacts.CONTENT_URI	管理联系人的Uri。
ContactsContract.CommonDataKinds.Phone.CONTENT_URI	管理联系人的电话的Uri。
ContactsContract.CommonDataKinds.Email.CONTENT URI	管理联系人的E-mail的Uri

Camera的ContentProvider

- Android提供了Camera 程序来支持拍照、拍摄视频，用户拍摄的照片、视频都将存放在固定的位置。有些时候，其他应用程序可能需要直接访问Camera所拍摄的照片、视频等，为了处理这种需求，Android 同样为这些多媒体内容提供了ContentProvider。

ContentObserver

- 有些时候，应用程序需要实时监听ContentProvider所共享数据的改变，并随着ContentProvider的数据的改变而提供响应,这就需要利用ContentObserver了。
- 只要导致ContentProvider数据的改变，程序就调用context. getContentResolver () . notifyChange (uri, null) ，这行代码可用于通知所有注册在该Uri上的监听者:该ContentProvider所共享的数据发生了改变。
    - 监听ContentProvider数据改变的监听器需要继承ContentObserver类，并重写该基类所定义的onChange(boolean selfChange)方法一当 它所监听的ContentProvider 数据发生改变时，该onChange()方法将会被触发。
    - 为了监听指定ContentProvider 的数据变化，需要通过ContentResolver 的 registerContentObserver (Uri uri, boolean notifyForDescendents, ContentObserver observer)向指定Uri 注册ContentObserver监听器
        - uri: 该监听器所监听的ContentProvider的Uri。
        - notifyForDescendents: 如果该参数设为true,假如注册监听的Uri为content://abc,那么Uri为contet:/abc/xyz、content:/abc/xyz/foo 的数据改变时也会触发该监听器;如果该参数设为false,假如注册监听的Uri为content://abc,那么只有content://abc的数据发生改变时才会触发该监听器。
        - observer: 监听器实例。

Service和BroadcastReceiver

- Service 一直在后台运行，它没有用户界面。一旦Service被启动起来之后，它就与Activity一样，它完全具有自己的生命周期。
- BroadcastReceiver组件就像一个全局的事件监听器，只不过它用于监听系统发出的Broadcast。通过使用BroadcastReceiver, 即可在不同应用程序
之间通信。

Service

创建、配置Service

- 创建
    1. 定义一个继承Service的子类。
    2. 在AndroidManifest.xml文件中配置该Service。
- Service与Activity都是从Context派生出来的，因此它们都可调用Context里定义的如getResources()、 getContentResolver()等方法。
- 生命周期

方法	说明
IBinder onBind(Intent intent)	该方法是Service 子类必须实现的方法。该方法返回一个IBinder对象，应用程序可通过该对象与Service组件通信。
void onCreate()	在该Service第一次被创建后将立即回调该方法。
void onDestroy()	在该Service 被关闭之前将会回调该方法。
void onStartCommand(Intent intent, int flags, int startId)	该方法的早期版本是void onStart(Intent intent, Int startId)，每次客户端调用startService(Intent)方 法启动该Service时都会回调该方法。
boolean onUnbind(Intent intent)	当该Service上绑定的所有客户端都断开连接时将会回调该方法

- 配置:定义了Service之后，在AndroidManifest.xml文件中配置该Service, 配置Service使用<service...>元素。与配置Activity相似的是，配置Service时也可为<service.../>元素配置<intent-filter../>子元素，用于说明该Service可被哪些Intent 启动。
  - <service.../>元素属性

属性	说明
name	指定该Service的实现类类名。
exported	指定该 Service是否能被其他 App启动。如果在配置该Service时指定了<intent-filter...> 子元素，则该属性默认为true。
permission	指定启动该Service所需的权限。
process	指定该Service所处的进程，该Service组件默认处于该App所在的进程中。实际上，Android的四大组件都可通过该属性指定进程。

启动和停止Service

- 启动Service的方式
  - 通过Context的startService(Intent service)方法:通过该方法启动Service,访问者与Service 之间没有关联，即使访问者退出了，Service 也仍然运行。
  - 通过Context的bindService(Intent service, ServiceConnection conn, int flags)方法:使用该方法启动Service, 访问者与Service绑定在一起，访问者一旦退出，Service 也就终止了。用于Service 和访问者之间需要进行方法调用或交换数据的情况。
    - **service**:该参数通过Intent指定要启动的Service。
    - **conn**:该参数是一个ServiceConnection对象，该对象用于监听访问者与Service 之间的连接情况。
      - 当访问者与Service 之间连接成功时将回调该ServiceConnection 对象的onServiceConnected(ComponentName name, IBinder service)方法
        - IBinder对象:该对象即可实现与被绑定Service之间的通信。

        - 当Service所在的宿主进程由于异常中止或其他原因终止，导致该Service 与访问者之间断开连接时回调该ServiceConnection对象的onServiceDisconnected(ComponentName)方法。
            - 注意：当调用者主动通过unBindService()方法断开与Service的连接时，ServiceConnection对象的onServiceDisconnected (ComponentName)方法并不会被调用。
    - **flags**:指定绑定时是否自动创建Service (如果Service还未创建)。该参数可指定为0 (不自动创建)或BIND_ AUTO_CREATE (自动创建)。
- 关闭Service的方式
    - startService()对应stopService()
    - bindService(Intent service, ServiceConnection conn, int flags)对应unbindService()
- 当开发Service类时，该Service类必须提供一个IBinder onBind(Intent intent)方法，在绑定本地Service的情况下，onBind(Intent intent)方法所返回的IBinder对象将会传给ServiceConnection对象里onServiceConnected(ComponentName name, IBinder service)方法的service 参数，这样访问者就可通过该IBinder对象与Service进行通信了。可以继承Binder (IBinder的实现类)的方式实现自己的IBinder对象。

生命周期

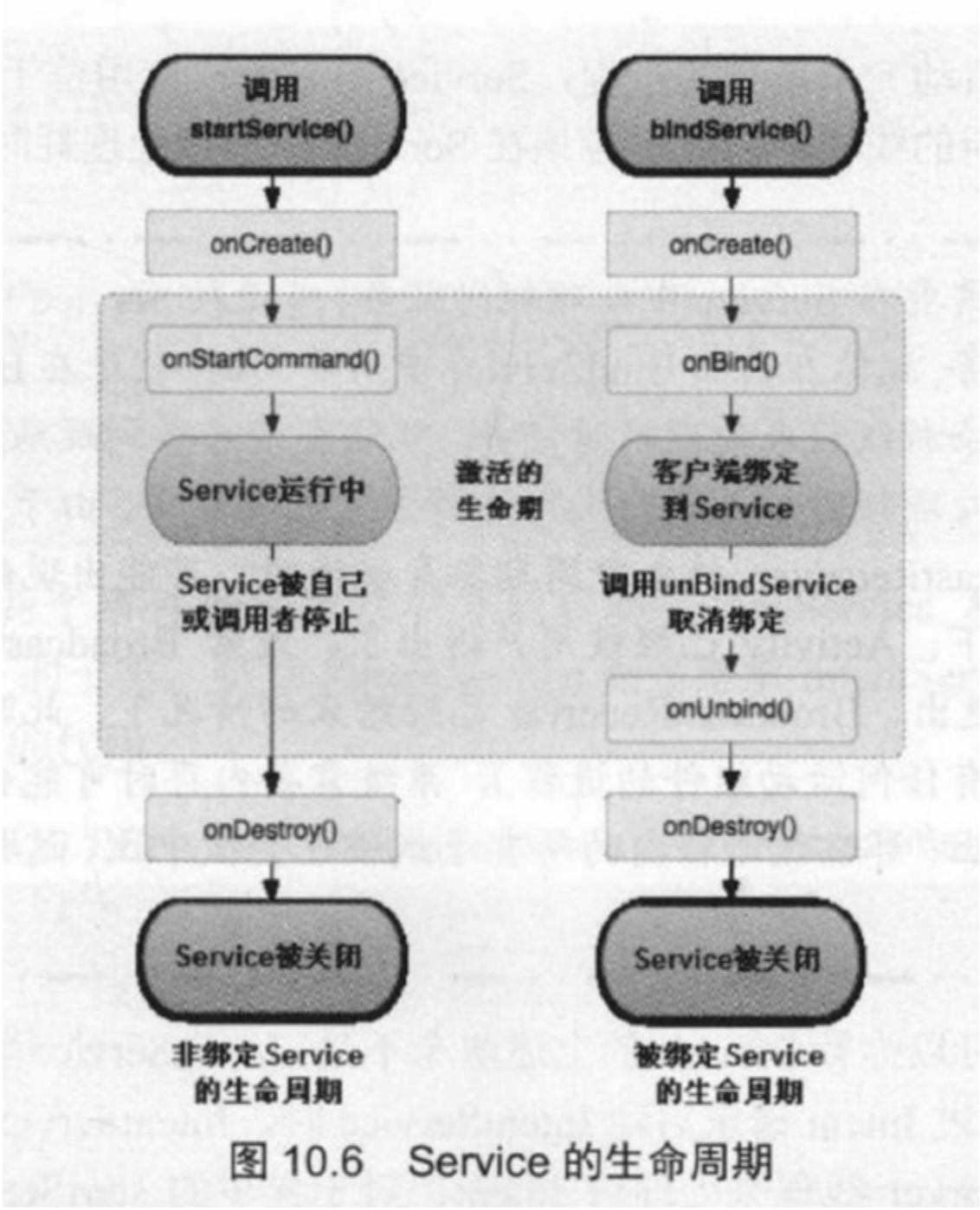
- 

图 10.6　Service 的生命周期

IntentService

- 背景
  - Service不会专门启动一个单独的进程，Service 与它所在应用位于同一个进程中。
  - Service不是一条新的线程，因此不应该在Service中直接处理耗时的任务。
- IntentService是Service 的子类。IntentService将会使用队列来管理请求Intent,每当客户端代码通过Intent 请求启动IntentService 时，IntentService 会将该Intent加入队列中，然后开启一条新的worker线程来处理该Intent。 对于异步的startService()请求， IntentService会按次序依次处理队列中的Intent,该线程保证同一时刻只处理一个Intent。由于IntentService 使用新的worker线程处理Intent请求，因此IntentService 不会阻塞主线程，所以IntentService自己就可以处理耗时任务。
- 特征
  - IntentService会创建单独的worker线程来处理所有的Intent 请求。
  - IntentService会创建单独的worker线程来处理onHandleIntent()方法实现的代码，因此开发者无须处理多线程问题。

- 当所有请求处理完成后，IntentService 会自动停止，因此开发者无须调用stopSelf()方法来停止该Service。为Service的onBind()方法提供了默认实现，默认实现的onBind()方法返回null。
  为Service的onStartCommand0方法提供了默认实现，该实现会将请求Intent添加到队列中。

AIDL Service

- AIDL :Android Interface Definition Language (AIDL)
- 在Android系统中，各应用程序都运行在自己的进程中，进程之间一般无法直接进行数据交换。为了实现跨进程通信(Interprocess Communication,简称IPC)，Android 提供了AIDL Service。
- Android的远程Service调用与Java的RMI基本相似，都是先定义一个远程调用接口，然后为该接口提供一个实现类即可。但Android并不是直接返回Service对象给客户端Service 只是将它的代理对象(IBinder对象)通过onBind()方法返回给客户端。因此，Android 的AIDL远程接口的实现类就是那个IBinder实现类。
  与绑定本地Service不同的是，本地Service的onBind()方法会直接把IBinder对象本身传给客户端的ServiceConnection的onServiceConnected方法的第二个参数;而远程Service的onBind()方法只是将IBinder对象的代理传给客户端的ServiceConnection的onServiceConnected方法的第二个参数。

创建AIDL

- Android需要AIDL (Android Interface Definition Language, Android 接口定义语言)来定义远程接口。
- 规则
  - AIDL定义接口的源代码必须以.aidl结尾。
  - 在AIDL接口中用到的数据类型，除基本类型、String、 List、 Map、CharSequence之外，其他类型全都需要导包，即使它们在同一个包中也需要导包。
- 开发人员定义的AIDL接口只是定义了进程之间的通信接口，Service 端、客户端都需要使用Android SDK安装目录下的build-tools子目录下的aidl.exe工具为该接口提供实现。如果开发人员使用Android Studio工具进行开发，那么Android Studio工具会自动为该AIDL接口生成实现。
- 使用：定义一个Service实现类了,该Service的onBind（ ）方法所返回的IBinder对象应该是ADT所生成的类的子类实例。
  - 在绑定本地Service的情况下，该对象会直接传给客户端的ServiceConnection对象的onServiceConnected()方法的第二个参数
  - 在绑定远程Service的情况下，只将该对象的代理传给客户端的ServiceConnection对象的onServiceConnected()方法的第二个参数

客户端访问AIDL Service

- AIDL接口定义了两个进程之间的通信接口,不仅服务器端需要AIDL接口，客户端也同样需要前面定义的AIDL接口。所以开发客户端的第一步就是将Service端的AIDL接口文件复制到客户端应用中，然后ADT工具会为AIDL接口生成相应的实现。
- 绑定远程Service流程
  - 创建ServiceConnection对象。
  - 以ServiceConnection对象作为参数，调用Context的bindService()方法绑定远程Service即可。
    - 绑定远程Service的ServiceConnection并不能直接获取Service的onBind()方法所返回的对象，它只能获取onBind()方法所返回的对象的代理，因此在 ServiceConnection的onServiceConnected方法中需要进行处理。
      - ```
        catService = ICat.Stub.asInterface（service）
        ```
        ```

- Android要求调用远程Service 的参数和返回值都必须实现Parcelable 接口。实现Parcelable 接口不仅要求实现该接口中定义的方法，而且要求在实现类中定义一个名为CREATOR、类型为Parcelable.Creator的静态常量。除此之外，还要求使用AIDL代码来定义这些自定义类型。
    - 定义了一个实现Parcelable 接口的类，要就是实现writerToParcel(Parceldest, int flags)方法，该方法负责把Person对象的数据写入Parcel中。与此同时，该类必须定义一个类型为Parcelable.Creator、名为CREATOR的静态常量，该静态常量的值负责从Parcel数据包中恢复Person对象，因此该对象定义的createFromPerson()方法用于恢复Person 对象。

系统Service

- Android系统本身提供了大量的系统Service,开发者只要在程序中调用Context的getSystemService(String name)即可获取对应的系统Service。
    - getSystemService(String name):根据Service名称来获取系统Service。

电话管理器(TelephonyManager)

- TelephonyManager是一个管理手机通话状态、电话网络信息的服务类，该类提供了大量的getXxx()方法来获取电话网络的相关信息。还提供
  了一个listen(PhoneStateListener listener, int events)方法来监听通话状态。

短信管理器(SmsManager)

- SmsManager提供了一系列sendXxxMessage()方法用于发送短信

音频管理器(AudioManager)

- 在某些时候，程序需要管理系统音量，或者直接让系统静音，这就可借助于Android 提供的AudioManager来实现。
- AudioManager方法
    - adjustStreamVolume(int streamType, int direction, int flags):调整手机指定类型的声音。
        - streamType指定声音类型
        - direction：指定对声音进行增大、减小还是静音等;
        - flags：调整声音时的标志，例如指定FLAG SHOW_ UI,则调整声音时显示音量进度条。
    - setMicrophoneMute(boolean on):设置是否让麦克风静音。
    - setMode(int mode):设置声音模式，可设置的值有NORMAL、RINGTONE和IN CALL。
    - setRingerMode(int ringerMode):设置手机的电话铃声模式。

振动器(Vibrator)

- 系统获取Vibrator 也是调用Context的getSystemService0方法即可，接下来就可调用Vibrator的方法来控制手机振动了。
- Vibrator方法
方法	说明
vibrate(VibrationEffect vibe)	控制手机按VibrationEffect效果执行振动。
vibrate(VibrationEffect vibe, AudioAttributes attributes)	控制手机按VibrationEffect效果执行振动，并执行AudioAttributes指定的声音效果。
cancel()	关闭手机振动。
    - VibrationEffect方法

方法	说明
createOneShot(long milliseconds, int amplitude)	创建只振动一次的振动效果。其中milliseconds指定振动时间，amplitude 指定振动幅度，该值可以是0~255之间的幅度。
createWaveform(long[] timings, int repeat)	创建波形振动的振动效果。其中timings指定振动停止、开始的时间，比如[400, 800, 1200], 就是指定在400ms、800ms、 1200ms 这些时间点交替关闭、启动振动。

闹钟服务（AlarmManager）

- AlarmManager不仅可用于开发闹钟应用，还可作为一个全局定时器使用，在Android应用程序中也是通过Context 的getSystemServiceQ方法来获取AlarmManager对象的。
- AlarmManager方法

方法	说明
set(int type, long triggerAtTime, PendingIntent operation)	设置在triggerAtTime时间启动由operation参数指定的组件。其中第一个参数指定定时服务的类型
setInexactRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)	设置一个非精确的周期性任务。例如，我们设置Alarm每个小时启动一次，但系统并不一定总在每个小时的开始启动Alarm服务。
setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)	设置一个周期性执行的定时服务
cancel(PendingIntent operation)	取消AlarmManager 的定时服务。

BroadcastReceiver

- 本质上就是一个全局监听器，用于监听系统全局的广播消息。它可以非常方便地实现系统中不同组件之间的通信。
  - 各种OnXxxListener只是程序级别的监听器，这些监听器运行在指定程序所在进程中，当程序退出时，OnXxxListener 监听器也就随之关闭了。但BroadcastReceiver属于系统级的监听器，它拥有自己的进程，只要存在与之匹配的Intent被广播出来，BroadcastReceiver 就会被激发。
- BroadcastReceiver用于接收程序(包括用户开发的程序和系统内建的程序)所发出的BroadcastIntent
- 使用
  1. 创建需要启动的BroadcastReceiver的Intent。
     - 实现BroadcastReceiver只要重写BroadcastReceiver的onReceive(Context context, Intent intent)方法即可。
     - 接下来就应该指定该BroadcastReceiver能匹配的Intent
       - 使用代码进行指定，调用BroadcastReceiver的Context的registerReceiver(BroadcastReceiver receiver, IntentFilter filter)方法指定。
       - 在AndroidManifest.xml文件中配置。

2. 调用Context 的sendBroadcast() 或sendOrderedBroadcast0方法来启动指定的BroadcastReceiver.
    - 当应用程序发出一个BroadcastIntent之后,所有匹配该Intent 的BroadcastReceiver都有可能被启动。

- 每次系统Broadcast事件发生后，系统都会创建对应的BroadcastReceiver实例，并自动触发它的onReceive()方法，onReceive()方 法执行完后，BroadcastReceiver 实例就会被销毁。
    - 与Activity组件不同的是，当系统通过Intent 启动指定了Activity 组件时，如果系统没有找到合适的Activity 组件，则会导致程序异常中止;但系统通过Intent激发BroadcastReceiver时，如果找不到合适的BroadcastReceiver组件，应用不会有任何问题。
    - 如果BroadcastReceiver 的onReceive()方法不能在10 秒内执行完成，Android 会认为该程序无响应。如果确实需要根据Broadcast来完成一项比较耗时的操作，则可以考虑通过Intent 启动一个Service来完成该操作。

发送广播

- 调用Context的sendBroadcast(Intent intent)方法， 这条广播将会启动intent参数所对应的BroadcastReceiver。
- 分类
    - Normal Broadcast (普通广播,sendBroadcast()) : Normal Broadcast是完全异步的，可以在同一时刻(逻辑上)被所有接收者接收到，消息传递的效率比较高。但缺点是接收者不能将处理结果传递给下一个接收者，并且无法终止Broadcast Intent的传播。
    - Ordered Broadcast (有序广播,sendOrderedBroadcast()) : Ordered Broadcast的接收者将按预先声明的优先级依次接收Broadcasto比如A的级别高于B、B的级别高于C,那么Broadcast先传给A,再传给B,最后传给C。优先级别声明在<intent fiter...元素的android:priority属性中，数越大优先级别越高，取值范围为-1000~ 1000，也可以调用IntentFilter 对象的setPriority()设置优先级别。Ordered Broadcast接收者可以终止Broadcast Intent 的传播，Broadcast Intent 的传播一旦终止，后面的接收者就无法接收到Broadcast。另外，Ordered Broadcast的接收者可以将数据传递给下一个接收者，比如A得到Broadcast后，可以往它的结果对象中存入数据，当Broadcast传给B时，B可以从A的结果对象中得到A存入的数据。
        - 优先接收到Broadcast的接收者可以通过setResultExtras(Bundle)方法将处理结果存入Broadcast 中，然后传给下一个接收者，下一个接收者通过代码`Bundle bundle = getResultExtras(true)`可以获取上一个接收者存入的数据。

系统广播消息

- 除接收用户发送的广播之外，BroadcastReceiver还可以接收系统广播。如果应用需要在系统特定时刻执行某些操作，就可以通过监听系统广播来实现。Android的大量系统事件都会对外发送标准广播。
    - 广播Action常量：参考Android API文档中关于Intent的说明

网络应用

- Android完全支持JDK本身的TCP、UDP网络通信API,也可以使用ServerSocket、Socket 来建立基于TCP/IP协议的网络通信还可以使用DatagramSocket、Datagrampacket、MulticastSocket 来建立基于UDP协议的网络通信。Android也支持JDK提供的URL、URLConnection等网络通信API。
- OkHttp:使用OkHttp取代原来的Apache HttpClient,依然可以非常方便地发送HTTP请求，并获取HTTP响应，从而简化网络编程。

基于TCP协议的网络通信

- TCP
    - 传输控制协议（英语：Transmission Control Protocol，缩写：TCP）是一种面向连接的、可靠的、基于字节流的传输层通信协议，由IETF的RFC 793定义。在简化的计算机网络OSI模型中，它完成第四层传输层所指定的功能。用户数据报协议（UDP）是同一层内另一个重要的传输协议。
    - 在因特网协议族（Internet protocol suite）中，TCP层是位于IP层之上，应用层之下的中间层。不同主机的应用层之间经常需要可靠的、像管道一样的连接，但是IP层不提供这样的流机制，而是提供不可靠的包交换。
    - 应用层向TCP层发送用于网间传输的、用8位字节表示的数据流，然后TCP把数据流分割成适当长度的报文段（通常受该计算机连接的网络的数据链路层的最大传输单元（MTU）的限制）。之后TCP把结果包传给IP层，由它来透过网络将包传送给接收端实体的TCP层。TCP为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认信息（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失并进行重传。TCP用一个校验和函数来检验数据是否有错误，在发送和接收时都要计算校验和。
    - 在不可靠的网络环境中，提供可靠的网络连接
- Java对基于TCP协议的网络通信提供了良好的封装，程序使用Socket对象来代表两端的通信接口，并通过Socket产生IO流来进行网络通信。

使用ServerSocket创建TCP服务器端

能接收其他通信实体连接请求的类是ServerSocket, ServerSocket 对象用于监听来自客户端的Socket连接，如果没有连接，它将一直处于 等待状态。ServerSocket 包含一个监听来自客户端连接请求的方法。

ServerSocket方法

方法	说明
Socket accept()	如果接收到一个客户端Socket的连接请求，该方法将返回一个与连接客户端Socket对应的Socket，否则该方法直处于等待状态，线程也被阻塞。
ServerSocket(int port)	用指定的端口port 来创建一个 ServerSocket。该端口应该有一个有效的端口整数值0~-65535。
ServerSocket(int port,int backlog)	增加一个用来改变连接队列长度的参数backlog。
ServerSocket(int port,int backlog,InetAddress localAddr)	在机器存在多个IP 地址的情况下，允许通过localAddr这个参数来指定将ServerSocket绑定到指定的IP地址。
close()	关闭该ServerSocket

    - 服务器不应该只接收一个客户端请求，而应该不断地接收来自客户端的所有请求，所以程序通常会通过循环不断地调用ServerSocket的accept()方法
    - 创建ServerSocket时没有指定IP地址，ServerSocket将会绑定到本机默认的IP地址。

使用Socket进行通信

- 客户端通常可以使用Socket的构造器来连接到指定服务器
- Socket方法
方法	说明
Socket(InetAddress/String remoteAddress, int port)	创建连接到指定远程主机、远程端口的 Socket,该构造器没有指定本地地址、本地端口，默认使用本地主机的默认IP地址，默认使用系统动态分配的端口。
Socket(InetAddress/String remoteAddress, int port, InetAddress localAddr, int localPort)	创建连接到指定远程主机、远程端口的Socket,并指定本地IP地址和本地端口，适用于本地主
InputStream getInputStream()	返回该Socket 对象对应的输入流，让程序通过该输入流从Socket中取出数据。
OutputStream getOutputStream()	返回该Socket对象对应的输出流，让程序通过该输出流向Socket中输出数据。
setSoTimeout(int timeout)	当网络连接、读取操作超过timeout之后，系统自动认为该操作失败
  - 多线程
    - C/S聊天室
      - 服务器：服务器端应该包含多条线程，每个Socket对应一条线程，该线程负责读取Socket对应输入流的数据(从客户端发送过来的数据)，并将读到的数据向每个Socket 输出流发送一遍(将一个客户端发送的数据“广播”给其他客户端)，因此需要在服务器端使用List来保存所有的Socket。
      - 客户端：每个客户端应该包含两条线程:一条负责生成主界面，响应用户动作，并将用户输入的数据写入Socket对应的输出流中;另一条负责读取Socket对应输入流中的数据(从服务器发送过来的数据)，并负责将这些数据在程序界面上显示出来。

使用URL访问网络资源

- URL (Uniform Resource Locator)对象代表统一资 源定位器，它是指向互联网“资源”的指针。资源可以是简单的文件或目录，也可以是对更复杂的对象的引用，例如对数据库或搜索引擎的查询。就通常情况而言，URL可以由协议名、主机、端口和资源组成
- URL类方法

方法	说明
String getFile()	获取此URL的资源名。
String getHost()	获取此URL的主机名。
String getPath()	获取此URL的路径部分。
int getPort()	获取此URL的端口号。
String getProtocol()	获取此URL的协议名称。
String getQuery()	获取此URL的查询字符串部分。
URLConnection openConnection()	返回一个URLConnection对象，它表示到URL所引用的远程对象的连接。
InputStream openStream()	打开与此URL的连接，并返回一个用于读取该URL资源的InputStream。

- 从Android 9开始，Android 默认要求使用加密连接,需要使用传输层安全协议(TransportLayer Security)。 但如果目标网站就是使用HTTP协议，那么App确实需要使用HTTP协议与目标网站通信，可以在AndroidManifest.xml文件配置
  - 在<application.../>元素 中通过android:networkSecurityConfig属性指定网络安全配置，通过该配置文件将目标网站加入白名单中。
  - 将<application.../>元素的android:usesCleartextTraffic 属性指定为true,该App将完全不受加密连接的限制。

使用URLConnection提交请求

- URL的openConnection0方法将返回一个URLConnection对象，该对象表示应用程序和URL之间的通信连接。程序可以通过URLConnection实例向该URL发送请求，读取URL引用的资源。通常创建一个和URL的连接，并发送请求、读取此URL引用的资源需要如下几个步骤。
  1. 通过调用URL对象的openConnection()方法来创建URLConnection对象。
  2. 设置URLConnection的参数和普通请求属性。
     - setRequestProperty(String key, String value):设置该URLConnection的key请求头字段的值为value
  3. 如果只是发送GET方式的请求，那么使用connect方法建立和远程资源之间的实际连接即可;如果需要发送POST方式的请求，则需要获取URLConnection实例对应的输出流来发送请求参数。
  4. 远程资源变为可用，程序可以访问远程资源的头字段，或通过输入流读取远程资源的数据。

方法	说明
Object getContent()	获取该URLConnection的内容。
String getHeaderField(String name)	获取指定响应头字段的值。
getInputStream()	返回该URLConnection对应的输入流，用于获取URLConnection响应的内容。
getOutputStream()	返回该URLConnection对应的输出流，用于向URLConnection发送请求参数。

使用HttpURLConection访问网络

- HttpURLConection继承自URLConnection，HttpURLConnection 在URLConnection的基础上做了进一步改进， 增加了一些用于操作HTTP资源的便捷方法。
- HttpURLConection方法
方法	说明
int getResponseCode()	获取服务器的响应代码。
String getResponseMessage()	获取服务器的响应消息。
String getRequestMethod()	获取发送请求的方法。
setRequestMethod(String method)	设置发送请求的方法。
- 多线程下载步骤
  1. 创建URL对象。
  2. 获取指定URL对象所指向资源的大小(由getContentLength()方 法实现)，此处用到了HttpURLConnection类。
  3. 在本地磁盘上创建一个与网络资源相同大小的空文件。
  4. 计算每条线程应该下载网络资源的哪个部分(从哪个字节开始，到哪个字节结束)。
  5. 依次创建、启动多条线程来下载网络资源的指定部分。

OkHttp

- 为了更好地处理向Web站点请求，包括处理Session、Cookie等细节问题，可以使用OkHttp用于发送HTTP请求，接收HTTP响应。
  - Android开发中的网络框架Retrofit 就是基于OkHttp 做的封装，Retrofit封装之后更符合RESTful风格，但Retrofit也丢失了部分灵活性。
- 使用OkHttp流程
  1. 创建OkHttpClient对象，如果只是发送简单的请求，则使用默认构造器创建即可;如果需要更有效地设置OkHttpClient,则应通过OkHttpClient. Builder对象。
  2. 通过Request. Builder构建Request对象。Request 代表一次请求， 所有和请求有关的信息都通过Request.Builder进行设置。Request.Builder对象方法

-

方法	说明
url(String url)	设置请求的URL。该方法有三个重载版本，该方法的参数可以是String、URL、HttpUrl。
addHeader(String name, String value)	设置请求头。
removeHeader(String name)	删除请求头。
cacheControl(CacheControl cacheControl)	设置Cache-Control请求头，用于控制缓存。
method(String method, RequestBody body)	设置请求方法和请求参数。其中RequestBody代表请求参数。
get()	method(method, body)方法的简化版本，用来发送GET请求。默认就是发送GET请求的，所以这个方法通常无须执行。
delete/post/put/patch(RequestBody body)	这几个方法都是method(method, body)方法的简化版本，分别代表发送DELETE、POST、PUT、PATCH请求，这些请求对于RESTful服务很常用。

3. 调用OkHttpClient的newCall（ ）方法，以Request对象为参数创建Call对象。
4. 如果要发送同步请求，则直接调用Call对象的execute()方法即可:如果要发送异步请求，则调用Call对象的enqueue()方法，在调用该方法时要传入一个Callback回调对象，该回调对象将会负责处理服务器响应成功和响应出错的情况。

WebView

- WebView组件本身就是一个浏览器实现，WebView 基于Chromium内核实现，直接支持WebRTC、WebAudio 和WebGL等。WebView 也允许执行JavaScript。
- Chromium也包括对Web组件规范的原生支持，如自定义元素、阴影DOM、HTML导入和模板等，这意味着开发者可以直接在WebView中使用聚合(Polymer) 和Material设计。
- 混合开发方式: Android + HTML 5混合开发。对于一些偏重展示、广告，尤其是需要经常更新的页面内容，用WebView嵌入一个HTML5页面是比较常用的做法，这样AndroidApp不需要更新，运营商只要更新服务器端的网页，WebView中显示的内容就会改变。而且不需要受制于应用商店的审核。
- WebView方法

方法	说明
goBack()	后退。
goForward()	前进。
loadUrl(String url)	加载指定URL对应的网页。
boolean zoomIn()	放大网页。
boolean zoomOut()	缩小网页。
loadData(String data, String mime Type, String encoding)	用于加载并显示data(HTML)代码。
loadDataWithBaseURL(String baseUrl, String data, String mimeType,String encoding, String historyUrl)	用于加载并显示data(HTML)代码。

    - data:指定需要加载的HTML代码。

WebView中的JavaScript调用Android方法

- 使用流程
    1. 调用WebView的getSettings获得WebSettings对象。
    2. 调用WebSettings的setJavaScriptEnabled(true)启用JavaScript调用功能。
    3. 调用WebView的addJavascriptInterface(Object object, String name)方法将object 对象暴露给JavaScript脚本。
    4. 在JavaScript脚本中通过刚才暴露的name对象调用Android方法。