概述

- 疯狂Android讲义（168页-229页，第四章深入了解Activity和Fragment）
  - 第二章
    1. 对话框组件的剩余部分
    2. 菜单
    3. 活动条
  - 第三章
    1. 基于监听的事件处理
    2. 基于回调的事件处理
    3. 系统设置响应
    4. 消息传递机制
    5. 异步任务
  - 第四章
    - Activity的配置、启动、关闭
    - Activity间的数据交换，Intent的bundle

疯狂Android第二章

对话框dialog剩余部分

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

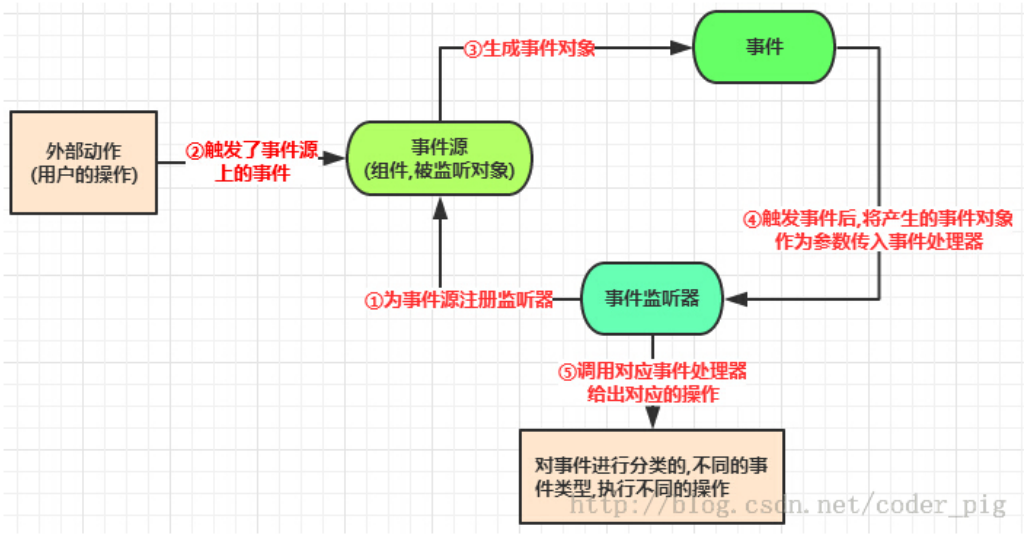
- ActionBar位于屏幕的顶部。ActionBar 可显示应用的图标和Activity 标题，ActionBar 的右边还可以显示活动项(Action Item)。
- 功能
    - 显示选项菜单的菜单项(将菜单项显示成Action Item)。
    - 使用程序图标作为返回Home主屏或向上的导航操作。
    - 提供交互式View作为Action View。
    - 提供基于Tab的导航方式，可用于切换多个Fragment。
    - 提供基于下拉的导航方式。
- ActionBar关闭：安卓不低于3.0的版本都自动打开ActionBar，关闭
    - ```
      <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- -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。

深入了解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)。

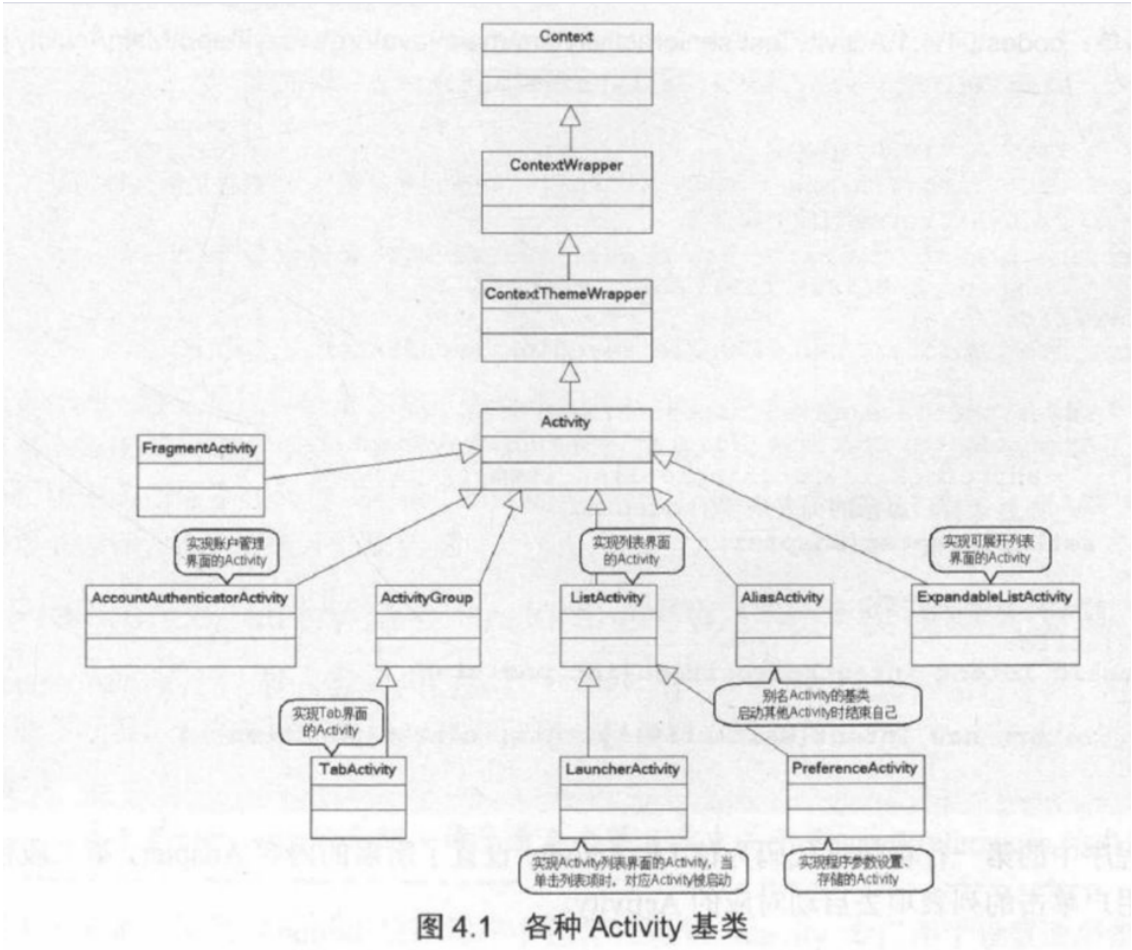
- 

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。