第4章 Android用户界面

杨刚
中国人民大学

本章学习目标：

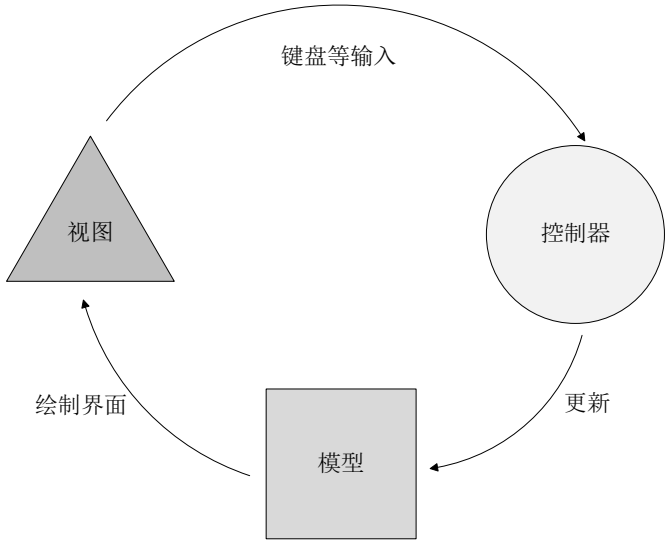
- 了解各种界面控件的使用方法
- 掌握几种基本界面布局的特点和使用方法
- 掌握Material Design设计理念与方法

4.1 用户界面基础

- 随着智能手机不断更新及其应用软件多样化发展，智能手机界面设计也趋于多样化，而良好的用户体验是界面设计的关键所在
- 手机界面的设计必须基于手机设备的物理特性和系统应用的特性进行合理的设计
- 优秀用户界面设计的一些基本原则
  - 美学完整性，功能与界面的匹配
  - 风格一致性
  - 操作直接化
  - 拟物化
  - 用户控制

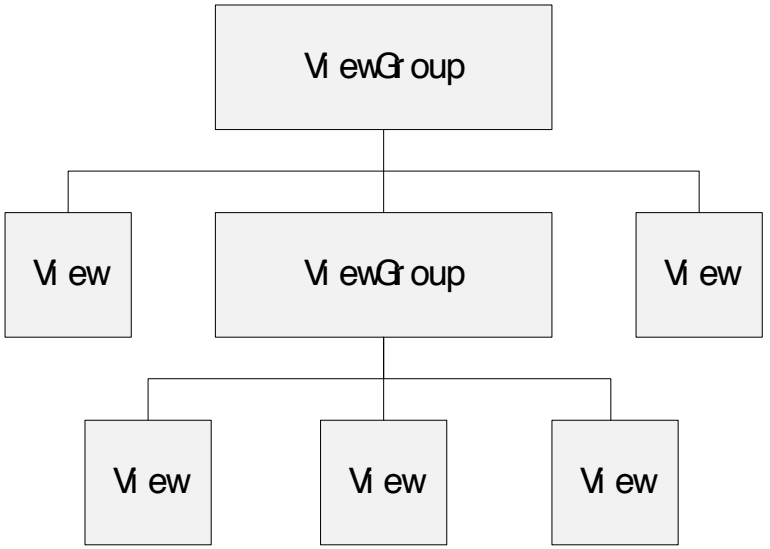
4.1 用户界面基础

- Android用户界面框架
  - Android用户界面框架采用MVC（Model-View-Controller）模型
    - 控制器（Controller）处理用户输入
    - 视图（View）显示用户界面和图像
    - 模型（Model）保存数据和代码
  - 扩展框架 MVP



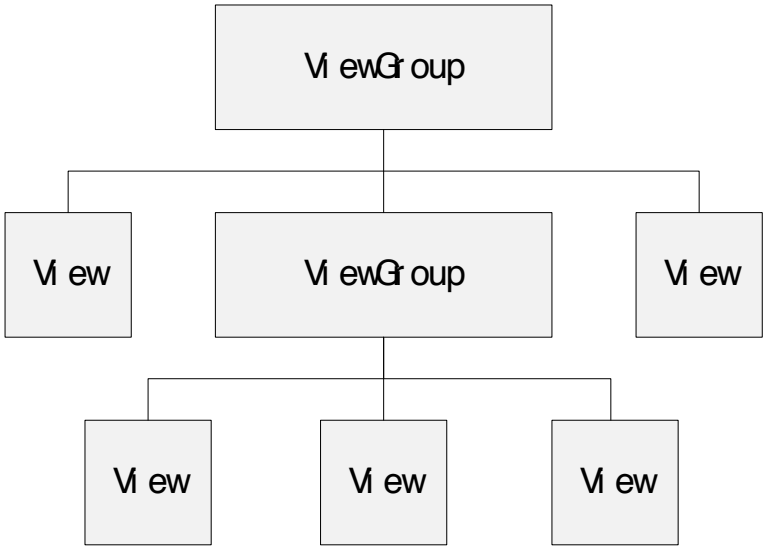
4.1 用户界面基础

- Android用户界面框架
  - Android用户界面框架采用视图树（View Tree）模型
    - 由View和ViewGroup构成
    - View是最基本的可视单元
    - ViewGroup是一种能够承载含多个View的显示单元



4.1 用户界面基础

- Android用户界面框架
  - Android用户界面框架采用视图树（View Tree）模型
    - Android系统会依据视图树的结构从上至下绘制每一个界面元素
    - 每个元素负责对自身的绘制，如果元素包含子元素，该元素会通知其下所有子元素进行绘制



4.1 用户界面基础

- Android用户界面框架
  - 单线程用户界面
    - 控制器从队列中获取事件和视图在屏幕上绘制用户界面，使用的都是同一个线程
    - 当应用程序启动时，系统会为应用程序创建一个主线程（或者叫UI线程），它负责分发事件到不同的组件，包括绘画事件
    - 特点：设计简单，处理函数具有顺序性，能够降低应用程序的复杂程度，同时也能减低开发的难度
    - 缺点：如果事件处理函数过于复杂，可能会导致用户界面失去响应

4.1 用户界面基础

- Android用户界面框架
  - 单线程用户界面
    - 确保只在UI线程中访问Android用户界面工具包中的组件;
    - 不能阻塞UI线程

4.2 界面控件

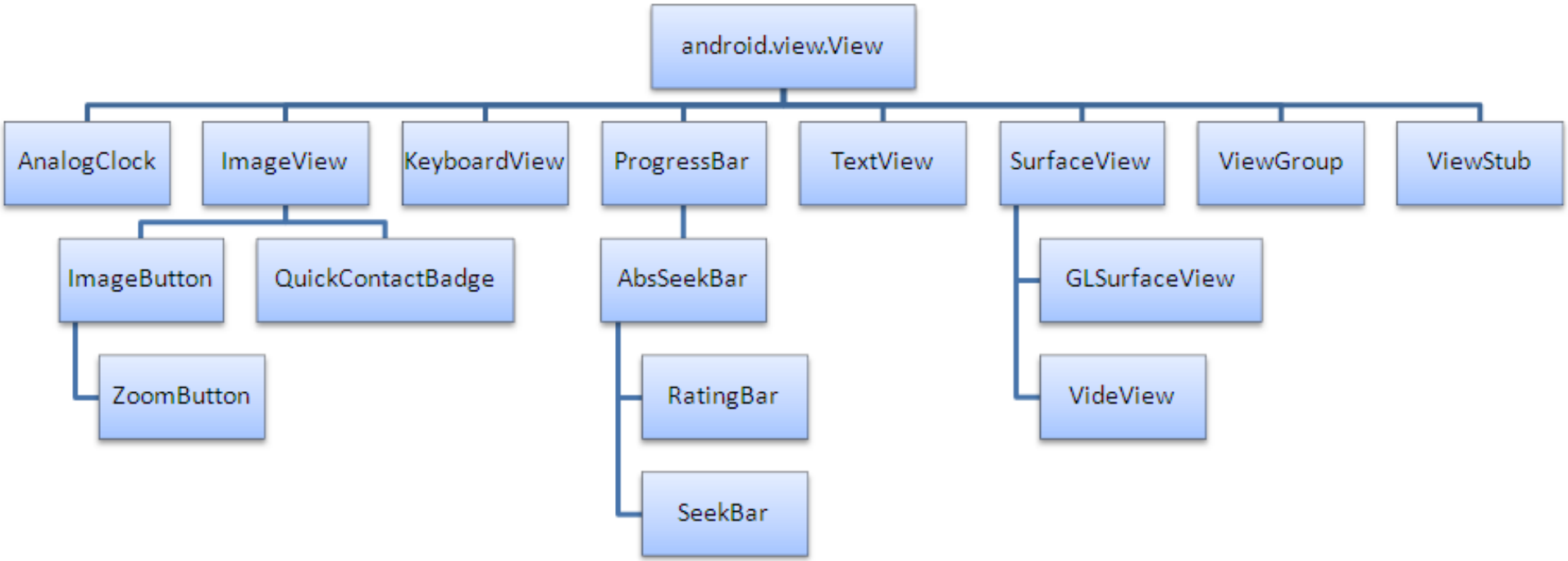
- 大多数的界面控件都在android.view和android.widget包中
- 若干常见的系统控件：
  - TextView，EditText，Button，ImageButton，Checkbox，RadioButton，Spinner，ListView，TabHost

4.2 界面控件

- Android的原生控件，一般是在res/layout下的xml文件中声明。Activity通过使用super.setContentView(R.layout.布局layout文件名)来加载layout。在Activity中获取控件的引用需要使用super.findViewById(R.id.控件ID)，就可以使用这个引用对控件进行操作，例如添加监听，设置内容等。当然也可以通过代码动态的使用控件

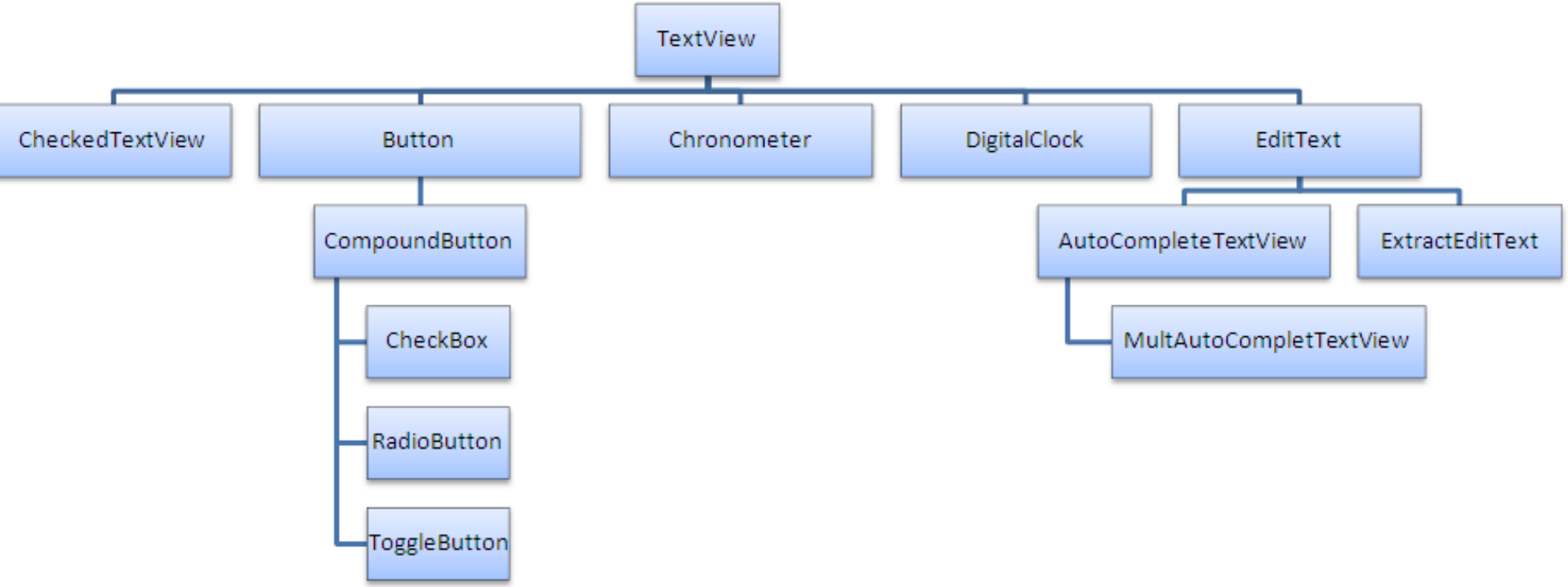
4.2 界面控件

- View子类结构图



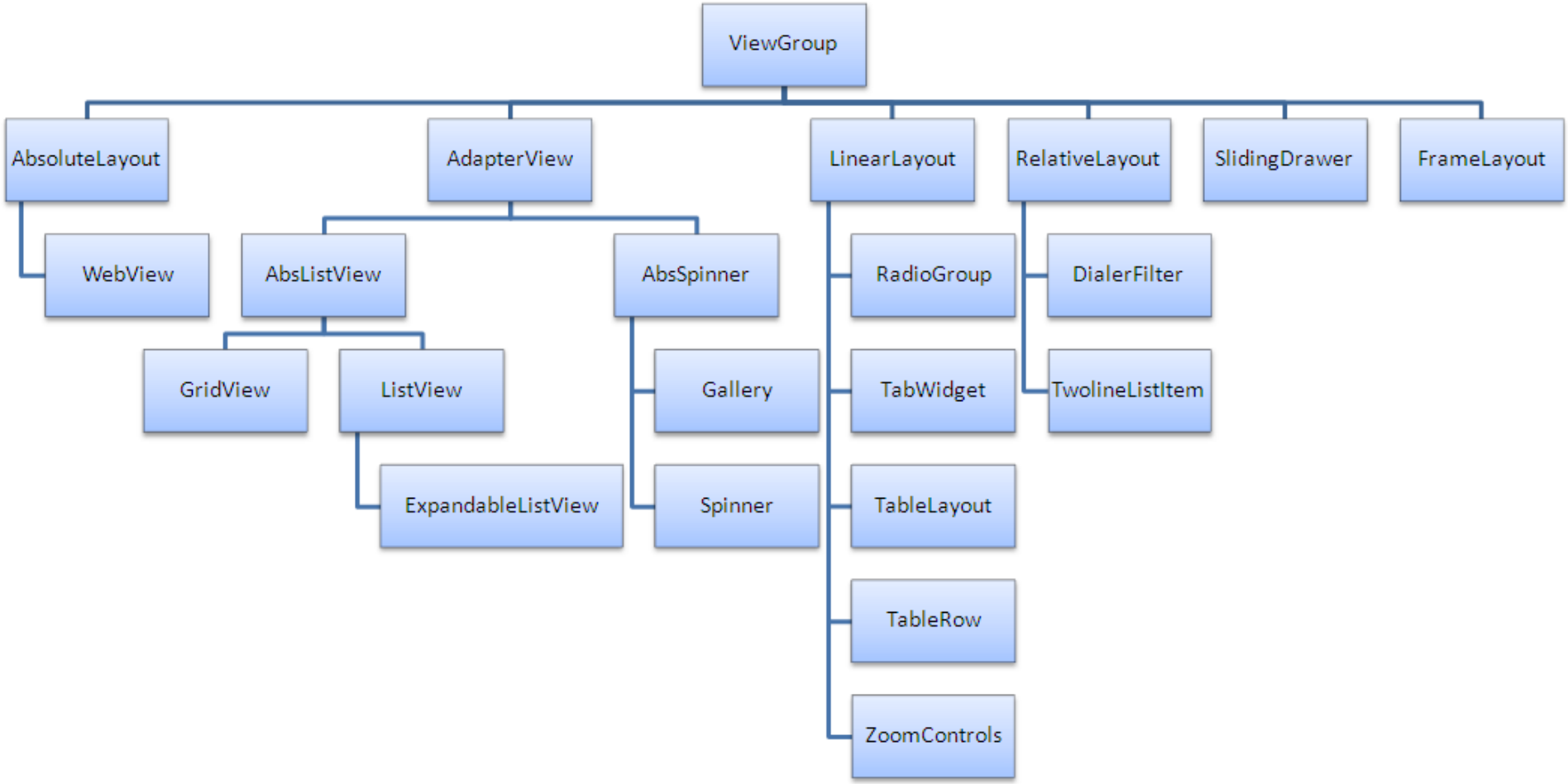
4.2 界面控件

- TextView子类结构图



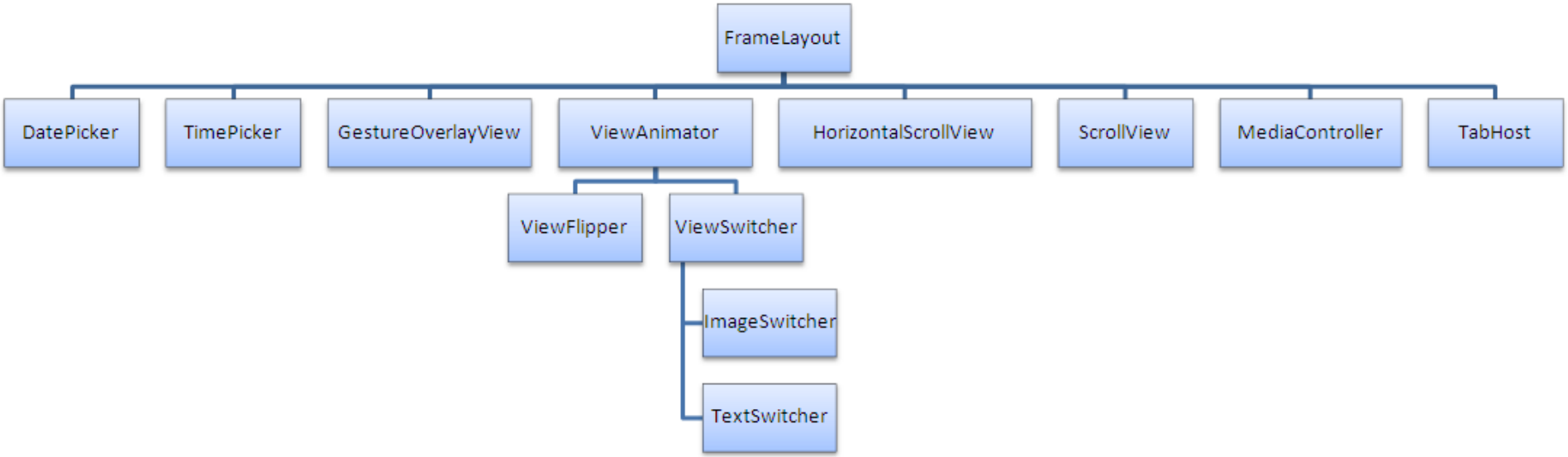
4.2 界面控件

- ViewGroup子类结构图



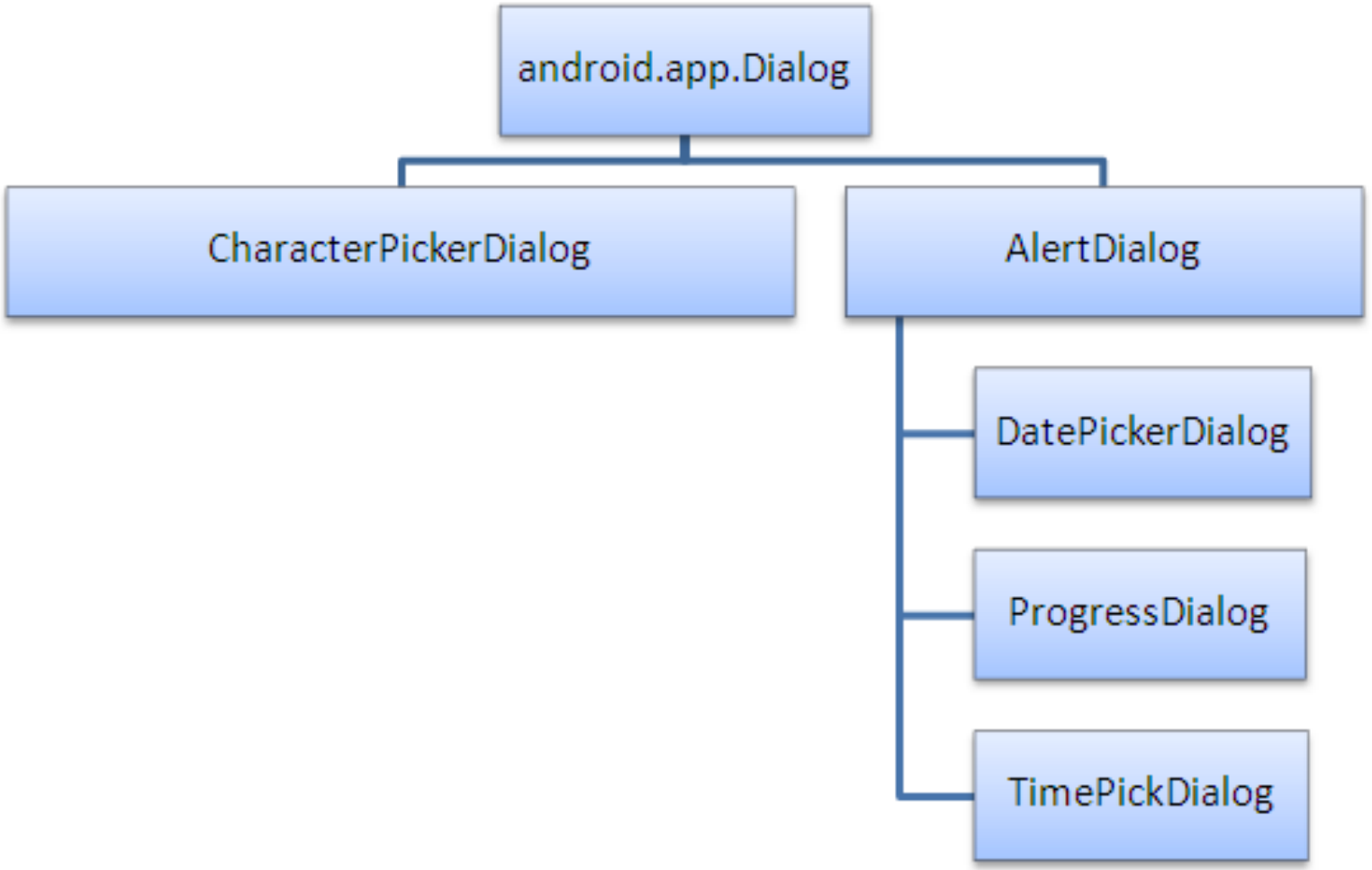
4.2 界面控件

- FrameLayout子类结构图



4.2 界面控件

- android.app.Dialog子类结构图



4.2 界面控件

- 4.2.1 TextView和EditText
  - TextView是一种用于显示字符串的控件
    - 定义：
    - java.lang.Object
      - Android.view.View
        - Android.widget.TextView
  - EditText则是用来输入和编辑字符串的控件
    - EditText是一个具有编辑功能的TextView
    - 定义：
    - java.lang.Object
      - Android.view.View
        - Android.widget.TextView
          Android.widget.EditText

4.2 界面控件

- 4.2.1 TextView和EditText
  - TextView组件的常用属性及对应方法

配置属性名称	对应方法	描述
android:text	setText(CharSequence text)	定义组件的显示文字
android:MaxLength	setFilters(InputFilter[] filters)	设置最大允许长度
android:textColor	setTextColor(int)	设置文本颜色
android:textSize	setTextSize(int,float)	设置显示文字大小
android:textStyle	setTypeface(Typeface)	设置文字显示的样式，粗体，斜体等
android:selectAllOnFocus	setSelectAllOnFocus(boolean)	默认选中并获得焦点
android:password	setTransformationMethod(TransformationMethod)	按密文方式显示文本

4.2 界面控件

- 4.2.2 Button和ImageButton
  - Button是一种按钮控件，用户能够在该控件上点击，并后引发相应的事件处理函数
    - 定义：
    - java.lang.Object
      - android.view.View
        - android.widget.TextView
          - android.widget.Button
  - ImageButton用以实现能够显示图像功能的控件按钮
    - 定义：
    - java.lang.Object
      - android.view.View
        - android.widget.ImageView
          - android.widget.ImageButton

4.2 界面控件

- 4.2.2 Button和ImageButton
  - View.OnClickListener()
    - View.OnClickListener()是View定义的点击事件的监听器接口，并在接口中仅定义了onClick()函数
    - 当Button从Android界面框架中接收到事件后，首先检查这个事件是否是点击事件，如果是点击事件，同时Button又注册了监听器，则会调用该监听器中的onClick()函数
    - 每个View仅可以注册一个点击事件的监听器，如果使用setOnClickListener()函数注册第二个点击事件的监听器，之前注册的监听器将被自动注销

4.2 界面控件

- 4.2.3 CheckBox和RadioButton
  - CheckBox同时可以选择多个选项的控件
    - 定义：
    - java.lang.Object
      - android.view.View
        - android.widget.TextView
          - android.widget.Button
            - android.widget.CompoundButton
              - android.widget.CheckBox

4.2 界面控件

- 4.2.3 CheckBox和RadioButton
  - RadioButton则是仅可以选择一个选项的控件, RadioGroup是RadioButton的承载体，程序运行时不可见。
    - 定义：
    - java.lang.Object
      - android.view.View
        - android.widget.TextView
          - android.widget.Button
            - android.widget.CompoundButton
              - android.widget.RadioButton

4.2 界面控件

4.2.4 Spinner

- 一种能够从多个选项中选择一个选项的控件，使用浮动菜单为用户提供选择
  - 定义：
  - java.lang.Object
    - android.view.View
      - android.ViewGroup
        - android.widget.AdapterView<T extends android.widget.Adapter>
          - android.widget.AbsSpinner
            - android.widget.Spinner

4.2 界面控件

- 4.2.4 Spinner
  - Spinner类的常用方法

No.	方法	类型	描述
1	public CharSequence getPrompt()	普通	取得提示文字
2	public void setPrompt(CharSequence prompt)	普通	设置组件的提示文字
3	public void setAdapter (SpinnerAdapter adapter)	普通	设置下拉列表项

4.2 界面控件

- 4.2.5 ImageView
  - 为图片展示提供一个容器
    - 定义：
    - java.lang.Object
      - android.view.View
        - android.widget.ImageView
  - 属性及操作方法

配置属性名称	对应方法	描述
android:maxHeight	setMaxHeight(int)	定义图片的最大高度
android:MaxWidth	setMaxWidth(int)	定义图片的最大宽度
android:src	setImageResource(int)	定义图片的ID

4.3 界面布局

- 界面布局
  - 界面布局（Layout）是用户界面结构的描述，定义了界面中所有的元素、结构和相互关系
  - 声明Android程序的界面布局有两种方法
    - 使用XML文件描述界面布局（推荐使用）
    - 在程序运行时动态添加或修改界面布局
  - 既可以独立使用任何一种声明界面布局的方式，也可以同时使用两种方式

4.3 界面布局

- 常用的6种界面布局
  - 线性布局
  - 框架布局
  - 表格布局
  - 相对布局
  - 绝对布局
  - 网格布局

4.3 界面布局

- 4.3.1 线性布局
  - 在线性布局（LinearLayout）中，所有的子元素都按照垂直或水平的顺序在界面上排列
    - 如果垂直排列，则每行仅包含一个界面元素
    - 如果水平排列，则每列仅包含一个界面元素

4.3 界面布局

- 4.3.1 线性布局
  - 修改界面控件的属性

编号	类型	属性	值
1	TextView	Id	@+id/label
		Text	用户名:
2	EditText	Id	@+id/entry
		Layout width	fill_parent
		Text	[null]
3	Button	Id	@+id/ok
		Text	确认
4	Button	Id	@+id/cancel
		Text	取消

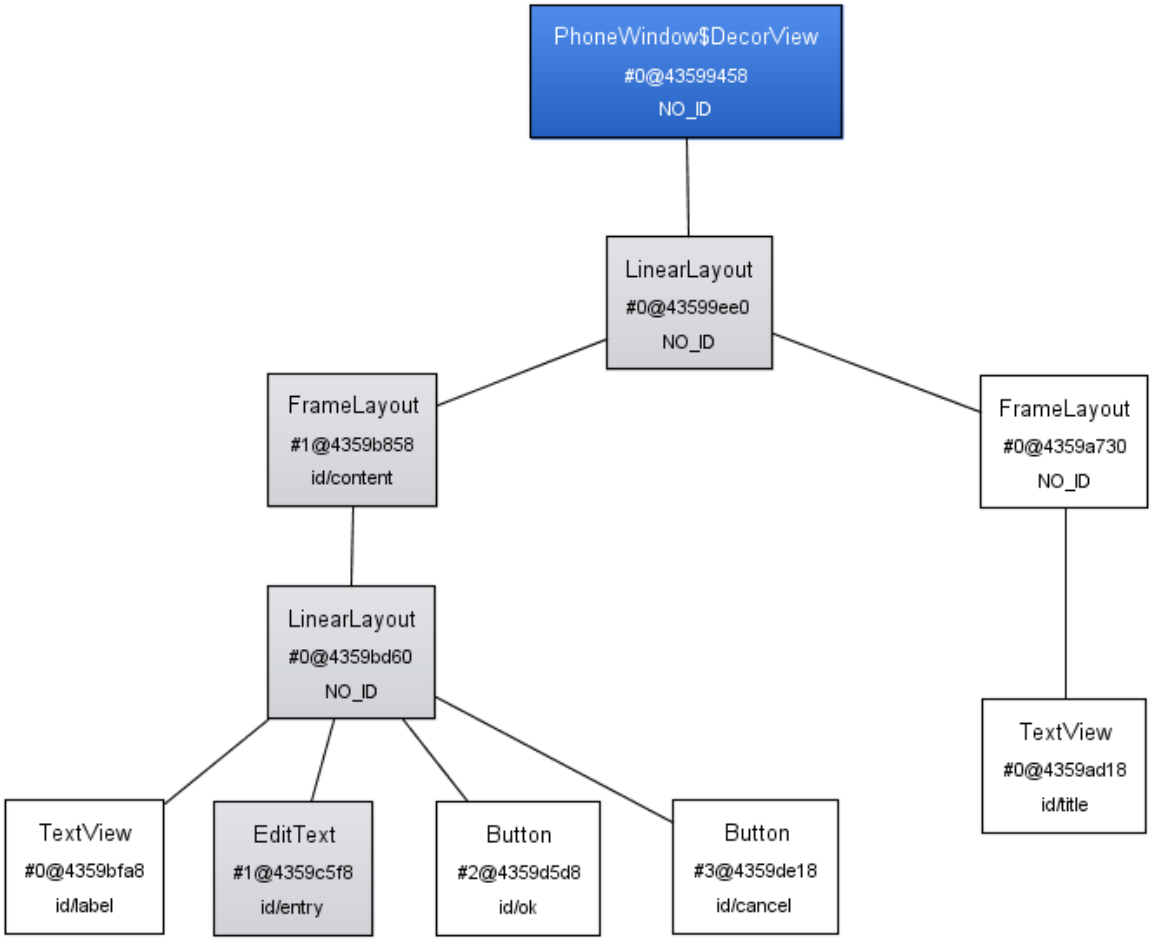
- ID是一个字符串，编译时被转换为整数，可以用来在代码中引用界面元素
- 一般仅在代码中需要动态修改的界面元素，才界面元素设置ID，反之则不需要设置ID

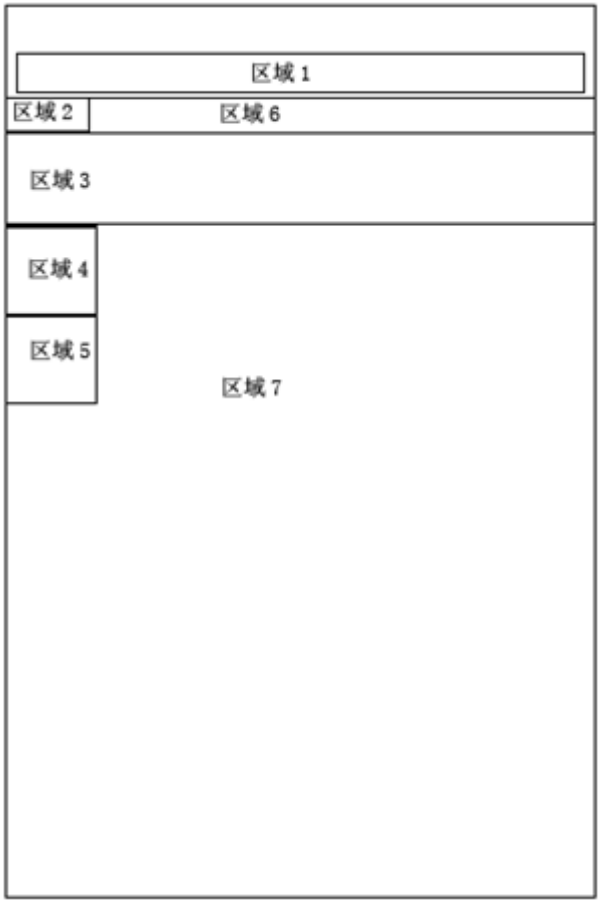
4.3 界面布局

- 4.3.2 框架布局
  - 框架布局（FrameLayout）是最简单的界面布局，是用来存放一个元素的空白空间，且子元素的位置是不能够指定的，只能够放置在空白空间的左上角
  - 如果有多个子元素，后放置的子元素将遮挡先放置的子元素
  - 使用Android SDK中提供的层级观察器（Hierarchy Viewer）进一步分析界面布局

4.3 界面布局

- 4.3.2 框架布局
  - 树形结构图和界面示意图





4.3 界面布局

4.3.2 框架布局

- 结合界面布局的树形结构图和示意图，分析不同界面布局和界面控件的区域边界
  - 用户界面的根节点（#0@43599ee0）是线性布局，其边界是整个界面，也就是示意图的最外层的实心线
  - 根节点右侧的子节点（#0@43599a730）是框架布局，仅有一个节点元素（#0@4359ad18），这个子元素是TextView控件，用来显示Android应用程序名称，其边界是示意图中的区域1。因此框架布局元素#0@43599a730的边界是同区域1的高度相同，宽度充满整个根节点的区域。这两个界面元素是系统自动生成的，一般情况下用户不能够修改和编辑
  - 根节点左侧的子节点（#1@4359b858）也是框架布局，边界是区域2到区域7的全部空间

4.3 界面布局

- 4.3.2 框架布局
  - 子节点（#1@4359b858）下仅有一个子节点（#0@4359bd60）元素是线性布局，因为线性布局的Layout width属性设置为fill_parent，Layout height属性设置为wrap_content，因此该线性布局的宽度就是其父节点#1@4359b858的宽带，高度等于所有子节点元素的高度之和
  - 线性布局#0@4359bd60的四个子节点元素#0@4359bfa8、#1@4359c5f8、#2@4359d5d8和#3@4359de18的边界，分别是界面布局示意图中的区域2、区域3、区域4和区域5

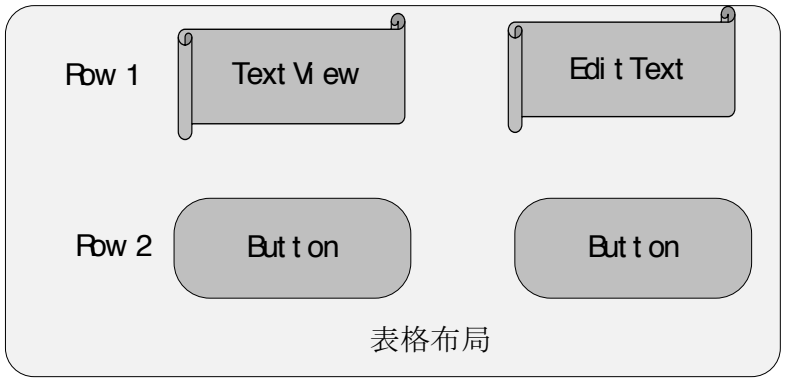
4.3 界面布局

- 4.3.3 表格布局
  - 表格布局（TableLayout）是一种常用的界面布局，通过指定行和列将界面元素添加到表格中
    - 网格的边界对用户是不可见的
    - 表格布局支持嵌套
      - 可以将表格布局放置在表格布局的表格中
      - 可以在表格布局中添加其他界面布局，例如线性布局、相对布局等

4.3 界面布局

4.3.3 表格布局

- 表格布局示意图



- 表格布局效果图

4.3 界面布局

4.3.3 表格布局

- 建立表格布局要注意以下几点
  - 在界面可视化编辑器上，再向TableRow02中拖拽两个Button；
  - 参考下表设置TableRow中四个界面控件的属性值；

编号	类型	属性	值
1	TextView	Id	@+id/label
		Text	用户名：
		Gravity	right
		Padding	3dip
		Layout width	160dip
2	EditText	Id	@+id/entry
		Text	[null]
		Padding	3dip
		Layout width	160dip
3	Button	Id	@+id/ok
		Text	确认
		Padding	3dip
4	Button	Id	@+id/cancel
		Text	取消
		Padding	3dip

4.3 界面布局

- 4.3.4 相对布局
  - 相对布局（RelativeLayout）是一种非常灵活的布局方式，能够通过指定界面元素与其他元素的相对位置关系，确定界面中所有元素的布局位置
  - 特点：能够最大程度保证在各种屏幕尺寸的手机上正确显示界面布局

4.3 界面布局

- 4.3.5 绝对布局
  - 绝对布局（AbsoluteLayout）能通过指定界面元素的坐标位置，来确定用户界面的整体布局
  - 绝对布局是一种不推荐使用的界面布局，因为通过X轴和Y轴确定界面元素位置后，Android系统不能够根据不同屏幕对界面元素的位置进行调整，降低了界面布局对不同类型和尺寸屏幕的适应能力

4.3 界面布局

- 4.3.6 网格布局
  - 网格布局（GridLayout）
    - Android SDK 4.0版本以上所支持的布局方式
    - 将用户界面划分为网格，界面元素可随意摆放在网格中
    - 网格布局比表格布局（TableLayout）在界面设计上更加灵活，在网格布局中界面元素可以占用多个网格的，而在表格布局却无法实现，只能将界面元素指定在一个表格行（TableRow）中，不能跨越多个表格行。

4.4 菜单

- 菜单
  - 应用程序中非常重要的组成部分
  - 在不占用界面空间的前提下，为应用程序提供了统一的功能和设置界面
  - 为程序开发人员提供了易于使用的编程接口
- Android系统支持三种菜单
  - 选项菜单（Option Menu）
  - 子菜单（Submenu）
  - 快捷菜单（Context Menu）

4.4 菜单

- 4.4.1菜单资源
  - Android程序的菜单
    - 可以在代码中动态生成
    - 使用XML文件制作菜单资源
      - 使用XML描述菜单是较好的选择
      - 可以将菜单的内容与代码分离
      - 有利于分析和调整菜单结构

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - 选项菜单是一种**经常**被使用的Android系统菜单
  - 打开方式：通过“**菜单键**”（MENU key）打开
  - 选项菜单分类
    - 图标菜单(Icon Menu) –添加图片，美观漂亮
    - 扩展菜单(Expanded Menu) -显示不全时，扩展出来

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - **图标菜单**
    - 能够同时显示**文字和图标**的菜单，最多支持**六**个子项
    - 不支持**单选框和复选框**

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - 扩展菜单
    - 在图标菜单子项多于**六**个时才出现，通过点击图标菜单最后的**子项“More”**才能打开。
    - 显示为垂直的列表型菜单
      - **不能够显示图标**
      - **可支持单选框和复选框**

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - 使用方法
    - 重载**Activity**的**onCreateOptionMenu()**函数，才能够在Android应用程序中使用选项菜单。
    - 初次使用选项菜单时，会调用**onCreateOptionMenu**()函数，用来初始化菜单子项的相关内容。
      - 设置菜单子项自身的ID和组ID；
      - 设置菜单子项显示的文字和图片等；
      - 关联文字和图片数组，可利用adapter。

菜单

- 选项菜单(Option Menu)—最基本的菜单

```
1.          final static int MENU_DOWNLOAD = Menu.FIRST;
2.          final static int MENU_UPLOAD = Menu.FIRST+1;
3.      @Override
4.      public boolean onCreateOptionsMenu(Menu menu){
5.                          menu.add(0,MENU_DOWNLOAD,0,"下载设置");
6.                          menu.add(0,MENU_UPLOAD,1,"上传设置");
7.                          return true;
8.      }
```

- 第1行和第2行代码将菜单子项**ID定义成静态常量**，并使用**静态常量Menu.FIRST（整数类型，值为1）**定义第一个菜单子项，以后的菜单子项仅需在Menu.FIRST增加相应的数值即可。**标识菜单使用**
- 第7行代码是**onCreateOptionsMenu()**函数返回值，函数的返回值类型为布尔型。
  - **返回true将显示在函数中设置的菜单，否则不能够显示菜单**

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - 第4行代码**Menu对象**作为一个参数被传递到函数内部，因此在onCreateOptionsMenu()函数中，用户可以使用Menu对象的add()函数添加菜单子项
  - add()函数的语法

```
MenuItem android.view.Menu.add(int groupId, int itemId, int order, CharSequence title)
```

    - 第1个参数groupId是组ID，用以批量的对菜单子项进行处理和排序；
    - 第2关参数itemId是子项ID，是每一个菜单子项的唯一标识，通过子项ID使应用程序能够定位到用户所选择的菜单子项；
    - 第3个参数order是定义菜单子项在选项菜单中的排列顺序；
    - 第4个参数title是菜单子项所显示的标题；

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - 添加**菜单子项的图标和快捷键**：使用setIcon()函数和setShortcut()函数

```
1.          menu.add(0,MENU_DOWNLOAD,0,"下载设置")
2.                                  .setIcon(R.drawable.download)
3.                                                  .setShortcut('2','d');
```

  - MENU_DOWNLOAD菜单设置图标和快捷键的代码
  - 第2行代码中使用了新的图像资源，用户将需要使用的图像文件拷贝到**/res/drawable**目录下
  - setShortcut()函数第一个参数是为**数字键盘**设定的快捷键

  第二个参数是为**全键盘设定**的快捷键，且**不区分字母的大小写**。

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - 动态菜单
    - 重载onPrepareOptionsMenu()函数，能够动态的添加、删除菜单子项，或修改菜单的标题、图标和可见性等内容
    - onPrepareOptionsMenu()函数的返回值的含义与onCreateOptionsMenu()函数相同
      - 返回true则显示菜单；
      - 返回false则不显示菜单；

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - 下面的代码是在用户每次打开选项菜单时，在菜单子项中显示用户打开该子项的次数

```
1.    static int MenuUploadCounter = 0;
2.    @Override
3.    public boolean onPrepareOptionsMenu(Menu menu){
4.                    MenuItem uploadItem = menu.findItem(MENU_UPLOAD);
5.                    uploadItem.setTitle("上传设置:" +String.valueOf(MenuUploadCounter));
6.                    return true;
7.    }
```

- 第1行代码设置一个菜单子项的计数器，用来统计用户打开“上传设置”子项的次数
- 第4行代码是通过将菜单子项的ID传递给menu.findItem()函数，获取到菜单子项的对象
- 第5行代码是通过MenuItem的setTitle()函数修改菜单标题

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - onOptionsItemSelected ()函数能够处理**菜单选择**事件，且该函数在每次点击**菜单子项**时都会被调用；
  - 下面的代码说明了**如何通过菜单子项的子项ID执行不同的操作。 根据getItemId获得具体子项，做对应处理。**

```
1.    @Override
2.    public boolean onOptionsItemSelected(MenuItem item){
3.        switch(item.getItemId()){
4.                case MENU_DOWNLOAD:
5.                        MenuDownlaodCounter++;
6.                        return true;
7.                case MENU_UPLOAD:
8.                        MenuUploadCounter++;
9.                        return true;
10.       }
11.       return false;
12.   }
```

菜单

- 选项菜单(Option Menu)—最基本的菜单
  - onOptionsItemSelected ()的返回值表示是否对菜单的选择事件进行处理
    - 如果已经处理过则返回true，否则返回false
  - 第2行的**MenuItem.getItemId()**函数可以获取到被选择菜单子项的ID
  - 程序运行后，通过点击“菜单键”可以调出程序设计的两个菜单子项

菜单

- 子菜单—扩展菜单项
  - 子菜单是能够显示更加详细信息的菜单子项
  - 菜单子项使用了浮动窗体的显示形式，能够更好适应小屏幕的显示方式

菜单

- 子菜单—扩展菜单项
  - Android系统的子菜单使用非常灵活，可以在选项菜单或快捷菜单中使用子菜单，有利于将相同或相似的菜单子项组织在一起，便于显示和分类
  - 子菜单不支持嵌套
  - 子菜单的添加是使用addSubMenu()函数实现

```
1. SubMenu uploadMenu = (SubMenu) menu.addSubMenu(0,MENU_UPLOAD,1,"上传设置")
                                    .setIcon(R.drawable.upload);
2. uploadMenu.setHeaderIcon(R.drawable.upload);
3. uploadMenu.setHeaderTitle("上传参数设置");
4. uploadMenu.add(0,SUB_MENU_UPLOAD_A,0,"上传参数A");
5. uploadMenu.add(0,SUB_MENU_UPLOAD_B,0,"上传参数B");
```

菜单

- 子菜单—扩展菜单项
  - 第1行代码在onCreateOptionsMenu()函数传递的menu对象上调用addSubMenu()函数，在选项菜单中添加一个菜单子项，用户点击后可以打开子菜单
  - addSubMenu()函数与选项菜单中使用过的add()函数支持相同的参数，同样可以指定菜单子项的ID、组ID和标题等参数，并且能够通过setIcon()函数菜单所显示的图标
  - 第2行代码使用setHeaderIcon ()函数，定义子菜单的图标
  - 第3行定义子菜单的标题，若不规定子菜单的标题，子菜单将显示父菜单子项标题，即第1行代码中“上传设置”
  - 第4行和第5行在子菜单中添加了两个菜单子项，菜单子项的更新函数和选择事件处理函数，仍然使用onPrepareOptionsMenu()函数和onOptionsItemSelected()函数

菜单

- 子菜单—扩展菜单项
  - 以上小节的代码为基础，将“上传设置”改为子菜单，并在子菜单中添加“上传参数A”和“上传参数B”两个菜单子项。

4.4 菜单

4.4.2快捷菜单

- 当用户点击界面元素超过2秒后，将启动注册到该界面元素的快捷菜单
- 类似于计算机程序中的“右键菜单”
- 与“选项菜单”的方法非常相似，需要重载：
  - onCreateContextMenu()函数
  - onContextItemSelected()函数
- 快捷菜单注册到界面中的某个控件上
  - registerForContextMenu()函数

4.4 菜单

4.4.2 快捷菜单

- 使用registerForContextMenu()函数，将快捷菜单注册到TextView控件上

```
TextView LabelView = null;
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LabelView = (TextView)findViewById(R.id.label);
        registerForContextMenu(LabelView);
}
```

菜单

- 快捷菜单—上下文菜单
  - 选项菜单中的onCreateOptionsMenu()函数仅在选项菜单**第一次启动**时被调用一次
  - 快捷菜单的onCreateContextMenu()函数**每次启动**时都会被调用一次

```
1.  final static int CONTEXT_MENU_1 = Menu.FIRST;
2.  final static int CONTEXT_MENU_2 = Menu.FIRST+1;
3.  final static int CONTEXT_MENU_3 = Menu.FIRST+2;
4.  @Override
5.  public void onCreateContextMenu(ContextMenu menu, View v,
                                    ContextMenuInfo menuInfo){
6.          menu.setHeaderTitle("快捷菜单标题");
7.          menu.add(0, CONTEXT_MENU_1, 0,"菜单子项1");
8.          menu.add(0, CONTEXT_MENU_2, 1,"菜单子项2");
9.          menu.add(0, CONTEXT_MENU_3, 2,"菜单子项3");
10. }
```

菜单

- 快捷菜单—上下文菜单
  - ContextMenu类支持**add()函数**（代码第7行）和**addSubMenu()函数**，可以在快捷菜单中添加**菜单子项和子菜单**
  - 第5行代码的**onCreateContextMenu**()函数中的参数
    - 第1个参数menu是需要显示的快捷菜单
    - 第2个参数v是用户选择的界面元素
    - 第3个参数menuInfo是所选择界面元素的额外信息

菜单

- 快捷菜单—上下文菜单
  - 菜单选择事件的处理需要重载**onContextItemSelected()**函数，该函数在用户选择快捷菜单中的菜单子项后被调用，与**onOptionsItemSelected ()函数的使用方法**基本相同

菜单

- 快捷菜单—上下文菜单

```
1.      @Override
2.      public boolean onContextItemSelected(MenuItem item){
3.              switch(item.getItemId()){
4.                      case CONTEXT_MENU_1:
5.                              LabelView.setText("菜单子项1");
6.                              return true;
7.                      case CONTEXT_MENU_2:
8.                              LabelView.setText("菜单子项2");
9.                              return true;
10.                     case CONTEXT_MENU_3:
11.                             LabelView.setText("菜单子项3");
12.             return true;
13.             }
14.             return false;
15.     }
```

菜单

- 快捷菜单—上下文菜单
  - 使用**registerForContextMenu()函数，将快捷菜单注册到界面控件**上（下方代码第7行）。这样，用户在长时间点击该界面控件时，便会启动快捷菜单
  - 为了能够在界面上直接显示用户所选择快捷菜单的菜单子项，在代码中引用了界面元素TextView（下方代码第6行），通过更改TextView的显示内容（上方代码第5、8和11行），显示用户所选择的菜单子项

```
1.  TextView LabelView = null;
2.  @Override
3.  public void onCreate(Bundle savedInstanceState) {
4.          super.onCreate(savedInstanceState);
5.          setContentView(R.layout.main);
6.          LabelView = (TextView)findViewById(R.id.label);
7.          registerForContextMenu(LabelView);
8.  }
```

菜单

- 快捷菜单
  - 下方代码是/src/layout/main.xml文件的部分内容，第1行声明了TextView的ID为label，在上方代码的第6行中，通过R.id.label将ID传递给findViewById()函数，这样用户便能够引用该界面元素，并能够修改该界面元素的显示内容

```
1.  <TextView   android:id="@+id/label"
2.      android:layout_width="fill_parent"
3.      android:layout_height="fill_parent"
4.      android:text="@string/hello"
5.   />
```

菜单

- 快捷菜单—上下文菜单
  - 与交互相关
    - 需要注意的一点，将TextView的android:layout_width设置为fill_parent，这样TextView将填充满父节点的所有剩余屏幕空间，用户点击屏幕TextView下方任何位置都可以启动快捷菜单。
    - 如果将android:layout_width设置为wrap_content，则用户必须准确点击TextView才能启动快捷菜单。
    - 需要用户交互是定位准确

菜单

- 快捷菜单
  - 完整代码参考MyContextMenu程序，运行结果如图所示

菜单

- 快捷菜单—上下文菜单
  - 在Android系统中，菜单不仅能够在代码中定义，而且可以像界面布局一样**在XML文件中**定义菜单；
  - 使用XML文件定义界面菜单，将代码与界面设计分类，有助于**简化代码的复杂程度**，并且更**有利于界面的可视化**；

菜单

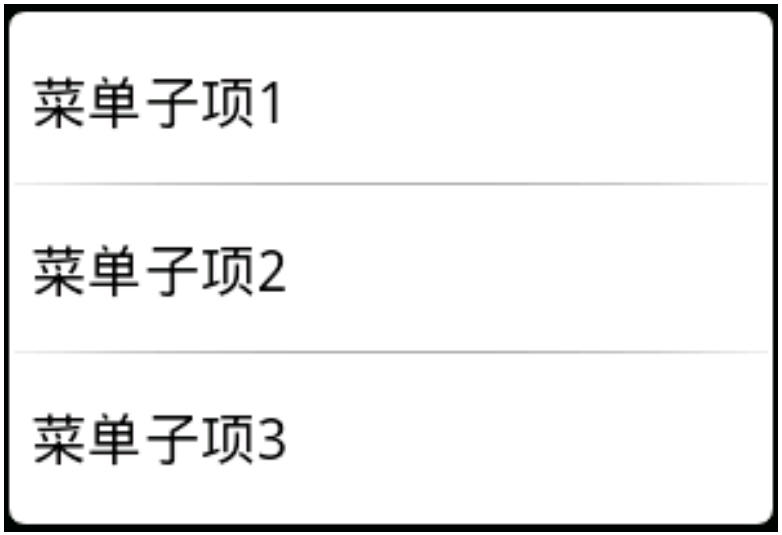
- 快捷菜单—上下文菜单
  - 首先需要创建保存菜单内容的XML文件
  - 在/src目录下建立子目录menu，并在menu下建立context_menu.xml文件，代码如下（专门的Menu标签）

```
1.  <menu xmlns:android="http://schemas.android.com/apk/res/android">
2.          <item android:id="@+id/contextMenu1"
3.                  android:title="菜单子项1"/>
4.          <item android:id="@+id/contextMenu2"
5.                  android:title="菜单子项2"/>
6.          <item android:id="@+id/contextMenu3"
7.                  android:title="菜单子项3"/>
8.  </menu>
```

    - 在描述菜单的XML文件中，必须以**<menu>标签**（代码第1行）作为根节点，<item>标签（代码第2行）用来描述菜单中的子项，<item>标签可以通过嵌套实现子菜单的功能。

菜单

- 快捷菜单—上下文菜单
  - XML菜单的显示结果如图所示



菜单

- 快捷菜单—上下文菜单
  - 在XML文件中定义菜单后，在onCreateContextMenu()函数中调用**inflater.inflate()**方法，将XML资源文件传递给菜单对象；

```
1.      @Override
2.      public void onCreateContextMenu(ContextMenu menu,
3.                                      View v, ContextMenuInfo menuInfo){
4.                  MenuInflater inflater = getMenuInflater();
5.                  inflater.inflate(R.menu.context_menu, menu);
6.  }
```

  - 第4行代码中的**getMenuInflater()为当前的Activity返回MenuInflater**
  - 第5行代码将XML资源文件R.menu.context_menu，传递给menu这个快捷菜单对象
  - **Inflater是标准的填充转换器，将XML文件转换为布局。**

界面事件—多数应用由事件驱动

- 在Android系统中，存在多种界面事件，如**点击事件、触摸事件、焦点事件和菜单事件**等等
- 在这些界面事件发生时，Android界面框架调用界面控件的事件处理函数对事件进行处理;
- 事件处理函数定义，并建立具体**事件**的关联。

界面事件

- 按键事件
  - 在MVC模型中，控制器根据界面事件（UI Event）类型不同，将事件传递给界面控件不同的事件处理函数。
    - 按键事件（KeyEvent）将传递给onKey()函数进行处理
    - 触摸事件（TouchEvent）将传递给onTouch()函数进行处理

界面事件

- 按键事件
  - Android系统界面事件的传递和处理遵循一的规则
    - 如果界面控件设置了事件监听器，则事件将先传递给事件监听器
    - 如果界面控件没有设置事件监听器，界面事件则会直接传递给界面控件的其他事件处理函数
    - 即使界面控件设置了事件监听器，界面事件也可以再次传递给其他事件处理函数
      - 是否继续传递事件给其他处理函数是由事件监听器处理函数的返回值决定的
      - 如果监听器处理函数的返回值为true，表示该事件已经完成处理过程，不需要其他处理函数参与处理过程，这样事件就不会再继续进行传递
      - 如果监听器处理函数的返回值为false，则表示该事件没有完成处理过程，或需要其他处理函数捕获到该事件，事件会被传递给其他的事件处理函数

界面事件

- 按键事件
  - 以EditText控件中的按键事件为例，说明Android系统界面事件传递和处理过程，假设EditText控件已经设置了按键事件监听器
    - 当用户按下键盘上的某个按键时，控制器将产生KeyEvent按键事件
    - Android系统会首先判断EditText控件是否设置了按键事件监听器，因为EditText控件已经设置按键事件监听器OnKeyListener，所以按键事件先传递到监听器的事件处理函数onKey()中

界面事件

- 按键事件
  - 事件能够继续传递给EditText控件的其他事件处理函数，完全根据onKey()函数的返回值来确定
  - 如果onKey()函数返回false，事件将继续传递，**这样EditText控件就可以捕获到该事件，将按键的内容显示在EditText控件中（这是EditText固有的事件响应操作）；**
  - 如果onKey()函数返回true，将阻止按键事件的继续传递，这样EditText控件就不能够捕获到按键事件，也就不能够将按键内容显示在EditText控件中。

界面事件

- 按键事件
  - Android界面框架支持对按键事件的监听，并能够将**按键事件的详细信息**传递给处理函数
  - 为了处理控件的按键事件，先需要设置按键事件的监听器，并重载onKey()函数

界面事件

- 按键事件

```
1.  entryText.setOnKeyListener(new OnKeyListener(){
2.          @Override
3.          public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4.                  //过程代码……
5.                  return true/false;
6.          }
```

  - 第1行代码是设置控件的按键事件监听器
  - 第3行代码的onKey ()函数中的参数
    - 第1个参数**view表示产生按键事件的界面控件**
    - 第2个参数keyCode表示按键代码
    - 第3个参数**keyEvent则包含了事件的详细信息**，如按键的重复次数、硬件编码和按键标志等
  - 第5行代码是onKey()函数的返回值
    - 返回true，阻止事件传递
    - 返回false，允许继续传递按键事件

界面事件

- 按键事件
  - KeyEventDemo是一个说明如何处理按键事件的示例
  - 完成此实例
  - KeyEventDemo用户界面
    - 最上方的EditText控件是输入字符的区域
    - 中间的CheckBox控件用来控制onKey()函数的返回值
    - 最下方的TextView控件用来显示按键事件的详细信息，包括按键动作、按键代码、按键字符、Unicode编码、重复次数、功能键状态、硬件编码和按键标志

界面事件

- 按键事件
  - 界面的XML文件的代码如下

```
1.  <EditText android:id="@+id/entry"
2.          android:layout_width="fill_parent"
3.          android:layout_height="wrap_content">
4.  </EditText>
5.  <CheckBox android:id="@+id/block"
6.          android:layout_width="wrap_content"
7.          android:layout_height="wrap_content"
8.          android:text="返回true，阻止将按键事件传递给界面元素" >
9.  </CheckBox>
10. <TextView android:id="@+id/label"
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:text="按键事件信息" >
14. </TextView>
```

界面事件

- 按键事件
  - 在EditText中，每当任何一个键子按下或抬起时，都会引发按键事件；
  - 为了能够使EditText处理按键事件，需要使用setOnKeyListener()函数在代码中设置按键事件监听器，并在onKey()函数添加按键事件的处理过程

界面事件

- 按键事件

```
1.  entryText.setOnKeyListener(new OnKeyListener(){
2.          @Override
3.          public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4.                  int metaState = keyEvent.getMetaState();
5.                  int unicodeChar = keyEvent.getUnicodeChar();
6.                  String msg = "";
7.                  msg +="按键动作:" + String.valueOf(keyEvent.getAction())+"\n";
8.                  msg +="按键代码:" + String.valueOf(keyCode)+"\n";
9.                  msg +="按键字符:" + (char)unicodeChar+"\n";
10.                 msg +="UNICODE:" + String.valueOf(unicodeChar)+"\n";
11.                 msg +="重复次数:" + String.valueOf(keyEvent.getRepeatCount())+"\n";
12.                 msg +="功能键状态:" + String.valueOf(metaState)+"\n";
13.                 msg +="硬件编码:" + String.valueOf(keyEvent.getScanCode())+"\n";
14.                 msg +="按键标志:" + String.valueOf(keyEvent.getFlags())+"\n";
15.                 labelView.setText(msg);
16.                 if (checkBox.isChecked())
17.                         return true;
18.                 else
19.                         return false;
20.                 }
```

界面事件

- 按键事件
  - 第4行代码用来获取功能键状态。**功能键包括左Alt键、右Alt键和Shift键**，当这三个功能键被按下时，**功能键代码metaState值分别为18、34和65；但没有功能键被按下时，功能键代码metaState值为0**
  - 第5行代码获取了按键的**Unicode值**，在第9行中，将Unicode转换为字符，显示在TextView中
  - 第7行代码获取了**按键动作，0表示按下按键，1表示抬起按键**。
  - 第11行代码获取按键的重复次数，但按键被长时间按下时，则会产生这个属性值
  - 第13行代码获取了按键的硬件编码，不同硬件设备的按键硬件编码都不相同，因此该值一般用于调试
  - 第14行获取了按键事件的标志符

界面事件

- **触摸**事件
  - Android界面框架支持对触摸事件的监听，并能够将触摸事件的详细信息传递给处理函数
  - 需要设置触摸事件的监听器，并重载onTouch ()函数

```
1.  touchView.setOnTouchListener(new View.OnTouchListener(){
2.          @Override
3.          public boolean onTouch(View v, MotionEvent event) {
4.                  //过程代码……
5.                  return true/false;
6.          }
```

    - 第1行代码是设置控件的触摸事件监听器
    - 在代码第3行的onTouch()函数中，第1个参数View表示产生触摸事件的界面控件；第2个参数**MontionEvent表示触摸事件的详细信息**，如产生时间、坐标和触点压力等
    - 第5行是onTouch()函数的返回值

界面事件

- 触摸事件
  - TouchEventDemo是一个说明如何处理触摸事件的示例
  - TouchEventDemo用户界面
    - 浅蓝色区域是可以接受触摸事件的区域，用户可以在Android模拟器中使用鼠标点击屏幕，用以模拟触摸手机屏幕
    - 下方黑色区域是显示区域，用来显示触摸事件的类型、相对坐标、绝对坐标、触点压力、触点尺寸和历史数据量等信息

界面事件

- 触摸事件
  - 在用户界面中使用了线性布局，并加入了3个TextView控件
    - 第1个TextView（ID为touch_area）用来标识触摸事件的测试区域
    - 第2个TextView（ID为history_label）用来显示触摸事件的历史数据量
    - 第3个TextView（ID为event_label）用来触摸事件的详细信息，包括类型、相对坐标、绝对坐标、触点压力和触点尺寸

界面事件

- 触摸事件
  - XML文件的代码如下

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.          android:orientation="vertical"
4.          android:layout_width="fill_parent"
5.          android:layout_height="fill_parent">
6.          <TextView  android:id="@+id/touch_area"
7.                  android:layout_width="fill_parent"
8.                  android:layout_height="300dip"
9.                  android:background="#0FF"
10.                 android:textColor="#FFFFFF"
11.                 android:text="触摸事件测试区域">
12.         </TextView>
```

  - 第9行代码定义了TextView的背景颜色，#80A0FF是颜色代码
  - 第10行代码定义了TextView的字体颜色

界面事件

- 触摸事件

```
13.         <TextView android:id="@+id/history_label"
14.                   android:layout_width="wrap_content"
15.                   android:layout_height="wrap_content"
16.                   android:text="历史数据量：" >
17.         </TextView>
18.         <TextView android:id="@+id/event_label"
19.                   android:layout_width="wrap_content"
20.                   android:layout_height="wrap_content"
21.                   android:text="触摸事件：" >
22.         </TextView>
23.   </LinearLayout>
```

界面事件

- 触摸事件
  - 在代码中为了能够引用XML文件中声明的界面元素，使用了下面的代码

```
1. TextView labelView = null;
2. labelView = (TextView)findViewById(R.id.event_label);
3. TextView touchView = (TextView)findViewById(R.id.touch_area);
4. final TextView historyView = (TextView)findViewById(R.id.history_label);
```

界面事件

- 触摸事件
  - 当手指**接触到触摸屏、在触摸屏上移动或离开触摸屏**时，分别会引发**ACTION_DOWN**、**ACTION_MOVE**和**ACTION_UP**触摸事件，而无论是哪种触摸事件，都会调用**onTouch()**函数进行处理
  - 事件类型包含在onTouch()函数的**MotionEvent**参数中，可以通过**getAction**()函数获取到触摸事件的类型，然后根据触摸事件的不同类型进行不同的处理
  - 为了能够使屏幕最上方的TextView处理触摸事件，需要使用setOnTouchListener()函数在代码中设置触摸事件监听器，并在onTouch()函数添加触摸事件的处理过程

界面事件

- 触摸事件

```
1.  touchView.setOnTouchListener(new View.OnTouchListener(){
2.          @Override
3.          public boolean onTouch(View v, MotionEvent event) {
4.              int action = event.getAction();
5.              switch (action) {
6.                case (MotionEvent.ACTION_DOWN):
7.                      Display("ACTION_DOWN",event);
8.                      break;
9.                case (MotionEvent.ACTION_UP):
10.                     int historySize = ProcessHistory(event);
11.                     historyView.setText("历史数据量："+historySize);
12.                     Display("ACTION_UP",event);
13.                     break;
14.               case (MotionEvent.ACTION_MOVE):
15.                     Display("ACTION_MOVE",event);
16.                     break;
17.             }
```

界面事件

- 触摸事件

```
18.                 return true;
19.             }
20.         });
```

  - 第7行代码的Display()是一个自定义函数，主要用来显示触摸事件的详细信息，函数的代码和含义将在后面进行介绍
  - 第10行代码的ProcessHistory()也是一个自定义函数，用来处理触摸事件的历史数据，后面进行介绍
  - 第11行代码是使用TextView显示历史数据的数量

界面事件

- 触摸事件
  - MotionEvent参数中不仅有触摸事件的类型信息，还触点的坐标信息，获取方法是使用getX()和getY()函数，这两个函数获取到的是触点相对于父界面元素的坐标信息。如果需要获取绝对坐标信息，则可使用getRawX()和getRawY()函数
  - 触点压力是一个介于0和1之间的浮点数，用来表示用户对触摸屏施加压力的大小，接近0表示压力较小，接近1表示压力较大，获取触摸事件触点压力的方式是调用getPressure()函数
  - 触点尺寸指用户接触触摸屏的接触点大小，也是一个介于0和1之间的浮点数，接近0表示尺寸较小，接近1表示尺寸较大，可以使用getSize()函数获取

界面事件

- 触摸事件
  - Display()将MotionEvent参数参数中的事件信息提取出来，并显示在用户界面上

```
1.  private void Display(String eventType, MotionEvent event){
2.          int x = (int)event.getX();
3.          int y = (int)event.getY();
4.          float pressure = event.getPressure();
5.          float size = event.getSize();
6.          int RawX = (int)event.getRawX();
7.          int RawY = (int)event.getRawY();
8.
9.          String msg = "";
10.         msg += "事件类型：" + eventType + "\n";
11.         msg += "相对坐标："+String.valueOf(x)+","+String.valueOf(y)+"\n";
12.         msg += "绝对坐标："+String.valueOf(RawX)+","+String.valueOf(RawY)+"\n";
13.         msg += "触点压力："+String.valueOf(pressure)+"，  ";
14.         msg += "触点尺寸："+String.valueOf(size)+"\n";
15.         labelView.setText(msg);
16.   }
```

界面事件

- 触摸事件
  - 两类情况
    - 一般情况下，如果用户将手指放在触摸屏上，但不移动，然后抬起手指，应先后产生ACTION_DOWN和ACTION_UP两个触摸事件
    - 但如果用户在屏幕上移动手指，然后再抬起手指，则会产生这样的事件序列：ACTION_DOWN → ACTION_MOVE → ACTION_MOVE → ACTION_MOVE → ……→ ACTION_UP

界面事件

- 触摸事件
  - 在手机上运行的应用程序，效率是非常重要的。如果Android界面框架不能产生足够多的触摸事件，则应用程序就不能够很精确的描绘触摸屏上的触摸轨迹
  - 如果Android界面框架产生了**过多**的触摸事件，虽然能够满足精度的要求，但却**降低了应用程序效率**
  - Android界面框架使用了**“打包”的解决方法**。在触点移动速度较快时会产生大量的数据，每经过一定的时间间隔便会产生一个ACTION_MOVE事件，在这个事件中，除了有当前触点的相关信息外，还包含这段时间间隔内触点轨迹的历史数据信息，这样既能够保持精度，又不至于产生过多的触摸事件。

界面事件

- 触摸事件
  - 通常情况下，在ACTION_MOVE的事件处理函数中，都先处理历史数据，然后再处理当前数据

```
1.  private int ProcessHistory(MotionEvent event)
2.      {
3.          int historySize = event.getHistorySize();
4.          for (int i = 0; i < historySize; i++) {
5.                  long time = event.getHistoricalEventTime(i);
6.                  float pressure = event.getHistoricalPressure(i);
7.                  float x = event.getHistoricalX(i);
8.                  float y = event.getHistoricalY(i);
9.                  float size = event.getHistoricalSize(i);
10.
11.                 // 处理过程.....
12.         }
13.         return historySize;
14.     }
```

界面事件

- 触摸事件
  - 第3行代码获取了历史数据的数量
  - 然后在第4行至12行中循环处理这些历史数据
  - 第5行代码获取了历史事件的发生时间
  - 第6行代码获取历史事件的触点压力
  - 第7行和第8行代码获取历史事件的相对坐标
  - 第9行获取历史事件的触点尺寸
  - 在第14行返回历史数据的数量，主要是用于界面显示
- **Android模拟器并不支持触点压力和触点尺寸**的模拟，所有触点压力恒为1.0，触点尺寸恒为0.0
- 同时Android模拟器上无法产生历史数据，因此历史数据量一直显示为**0**

触屏事件的真机处理

```
public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        Log.v("Himi", "ACTION_DOWN");
    } else if (event.getAction() == MotionEvent.ACTION_UP) {
        Log.v("Himi", "ACTION_UP");
    } else if (event.getAction() == MotionEvent.ACTION_MOVE) {
        Log.v("Himi", "ACTION_MOVE");
    }
    synchronized (object) {  //给出一个对象做排他延迟使用，防止过贫
            //乏的响应
    try {
        object.wait(TIME);
    } catch (InterruptedException e) {
    e.printStackTrace();
    }
}
    return true
}
```

UI用户界面设计：Activity间跳转

- 两种不同的跳转方式
  - 通过Intent跳转

```
Intent intent = new Intent();
intent.setClass(MainActivity.this, SpinnerActivity.class);
startActivity(intent);
```

  - 需要有返回结果的跳转

```
Intent bintent = new Intent(A.this, B.class);
String bsayHello = "Hello, this is B speaking";
bintent.putExtra("listenB", bsayHello)
startActivityForResult(bintent,10);
```

回传处理：

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
}
```

两种打开Activity的区别

区别：

1.startActivity( )

仅仅是跳转到目标页面，若是想跳回当前页面，则必须再使用一次startActivity( )。

2. startActivityForResult( )

可以一次性完成这项任务，当程序执行到这段代码的时候，页面会跳转到下一个Activity，而当这个Activity被关闭以后(this.finish())，程序会自动跳转会第一个Activity，并调用前一个Activity的onActivityResult( )方法。

4.5 Material Design

- 概述
  - 2014年Google I/O大会隆重发布，把系统内的各种设计都规范成一种变形的纸片，并套用现实中纸墨的物理模型进行交互
  - Material Design提出了平面像素的Z轴概念，通过纸片在物理世界中形态的抽象和提炼，定义了各种信息层级和常用状态的表达方式，并详细讲解了各个细节的处理方法

4.5 Material Design

- 概述
  - 核心思想，就是把物理世界的体验带进屏幕。去掉现实中的杂质和随机性，保留其最原始纯净的形态、空间关系、变化与过渡，配合虚拟世界的灵活特性，还原最贴近真实的体验，达到简洁与直观的效果。
  - Material design是最重视跨平台体验的一套设计语言。由于规范严格细致，保证它在各个平台使用体验高度一致。不过目前还只有Google自家的服务这么做，毕竟其他平台有自己的规范与风格。

4.5 Material Design

- 属性
  - Material是一种在现实中不存在的**"magical material"**。Material的灵感来源于纸片，却不是纸片，而是作为信息载体的魔法纸片。
  - 魔法纸片层叠、合并、分离，拥有现实中的厚度、惯性和反馈，同时拥有液体的一些特性，能够自由伸展变形

4.5 Material Design

- 空间
  - 引入了z轴的概念，z轴垂直于屏幕，用来表现元素的层叠关系。z值（海拔高度）越高，元素离界面底层（水平面）越远，投影越重。这里有一个前提，所有的元素的厚度都是1dp。
  - 所有元素都有默认的海拔高度，对它进行操作会抬升它的海拔高度，操作结束后，它应该落回默认海拔高度。同一种元素，同样的操作，抬升的高度是一致的。
  - 注意：这不止是设计中的概念，开发人员确实可以通过一个值来控制元素的海拔高度和投影。

4.5 Material Design

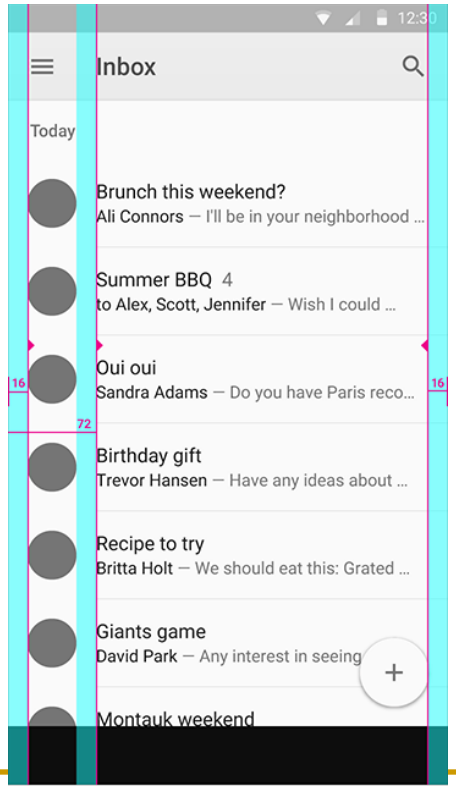
- 高度即层级
  - 所有对象都是以“父-子”关系描述的层级体系的一部分。
  - “父-子”元素说明：
    - 每一个对象只有一个“父”元素。
    - 每一个对象可能会有任意数量的“子”元素。
    - “子”元素继承来自“父”元素的可以转移的属性，比如位置、循环、刻度和高度。
    - “兄弟”元素是指与某一对象处在同一层级的对象。
  - 例外：项目以根元素为父元素，比如主 UI 元素，它们相比于其他对象来说会自主移动。比如说，浮动动作按钮不会与内容一起转动。

4.5 Material Design

- 动画
  - 动画不只是装饰，它有含义，能表达元素、界面之间的关系，具备功能上的作用。
  - 动画要贴近真实世界，就要重视easing。物理世界中的运动和变化都是有加速和减速过程的，忽然开始、忽然停止的匀速动画显得机械而不真实。考虑动画的easing，要先考虑它在现实世界中的运动规律。

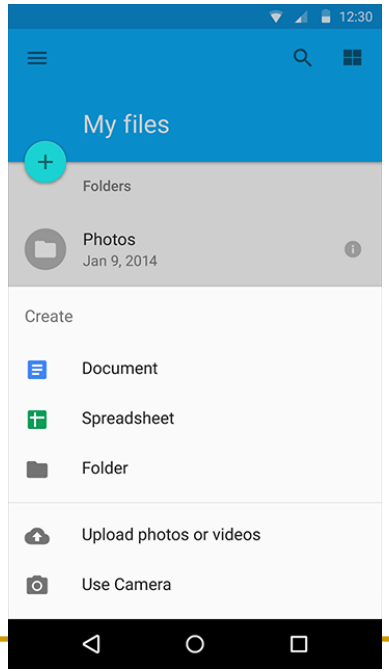
4.5 Material Design

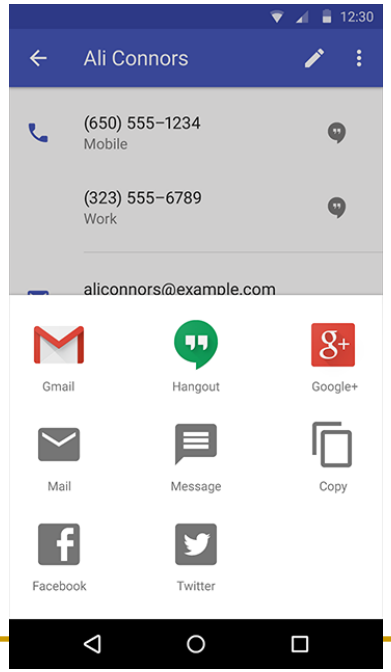
- 布局
  - 所有可操作元素最小点击区域尺寸：48dp X 48dp
  - 栅格系统的最小单位是8dp，一切距离、尺寸都应该是8dp的整数倍。以下是一些常见的尺寸与距离
    - 顶部状态栏高度：24dp
    - Appbar最小高度：56dp
    - 底部导航栏高度：48dp
    - 悬浮按钮尺寸：56x56dp/40x40dp
    - 用户头像尺寸：64x64dp/40x40dp
    - 小图标点击区域：48x48dp
    - 侧边抽屉到屏幕右边的距离：56dp
    - 卡片间距：8dp
    - 分隔线上下留白：8dp
    - 大多元素的留白距离：16dp
    - 屏幕左右对齐基线：16dp
    - 文字左侧对齐基线：72dp

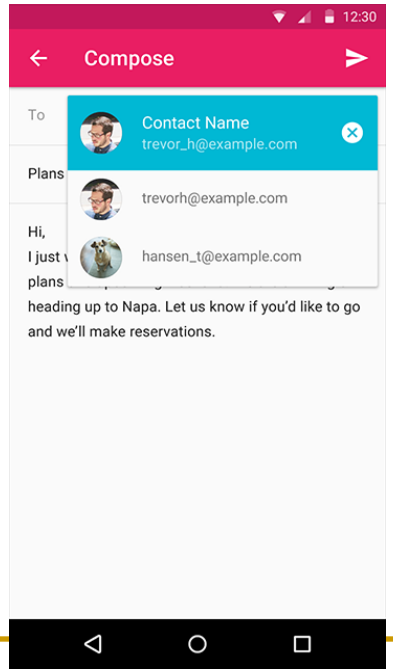


4.5 Material Design

- Bottom sheets
  - 通常以列表形式出现，支持上下滚动。也可以是网格式的。
  - Chips -- 狭小空间内表现复杂信息的一个组件，比如日期、联系人选择器

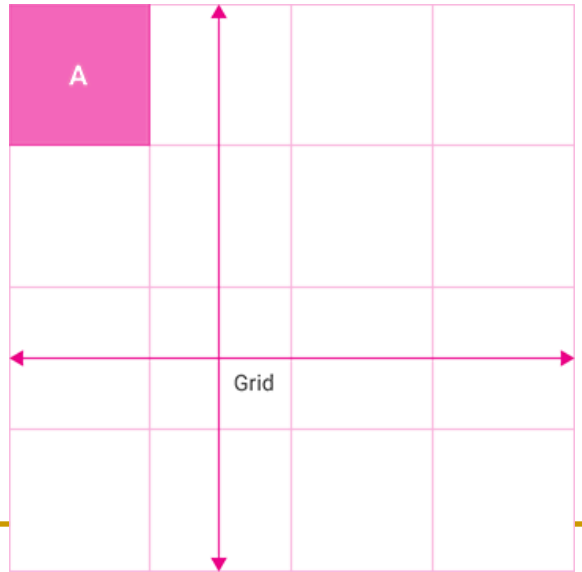


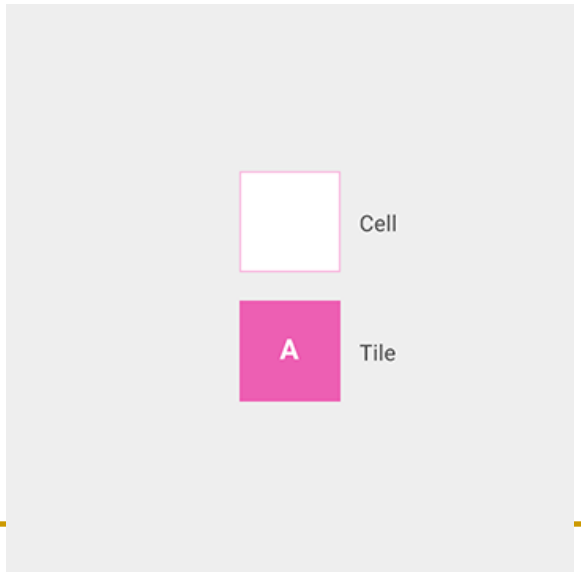




4.5 Material Design

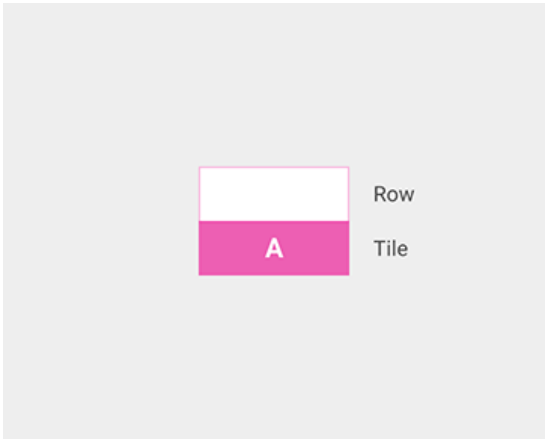
- Grides
  - Grides由单元格构成，单元格中的瓦片用来承载内容。
  - 瓦片包含主操作区和副操作区，副操作区的位置可以在上下左右4个角落。在同一个网格中，主、副操作区的内容与位置要保持一致。两者的操作都应该直接生效，不能触发菜单。
  - Grides只能垂直滚动。单个瓦片不支持滑动手势，也不鼓励使用拖放操作。
  - Grides中的单元格间距是2dp或8dp





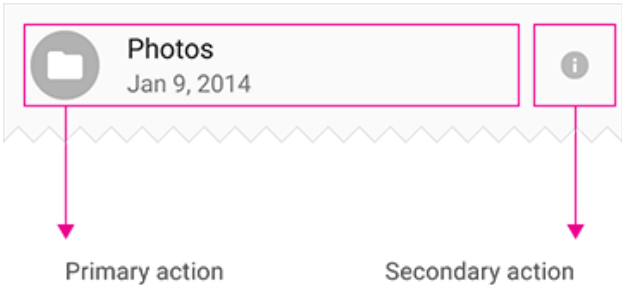
4.5 Material Design

- Lists



- 列表由行构成，行内包含瓦片。如果列表项内容文字超过3行，请改用卡片。如果列表项的主要区别在于图片，请改用网格。

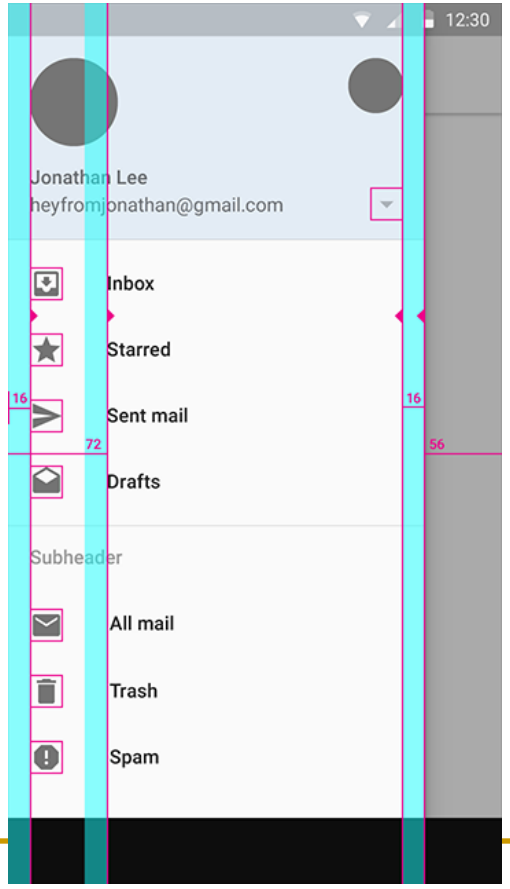
- 列表包含主操作区与副操作区。副操作区位于列表右侧，其余都是主操作区。在同一个列表中，主、副操作区的内容与位置要保持一致。



4.5 Material Design

- Navigation drawer

侧边抽屉从左侧滑出，占据整个屏幕高度，遵循普通列表的布局规则。手机端的侧边抽屉距离屏幕右侧56dp。



侧边抽屉支持滚动。如果内容过长，设置和帮助反馈可以固定在底部。抽屉收起时，会保留之前的滚动位置。列表较短不需要滚动时，设置和帮助反馈跟随在列表后面。

4.5 Material Design

- 详细内容请参考官方文档：
- http://www.google.com/design/spec/material-design/