移动应用软件开发

本次课内容：用户交互

1. 打开雨课堂签到
2. 播放的视频是有声音的，判断你的是不是能听到。
3. 通过视频判断声音和画面时间差（有字幕）。
4. 在右边消息框输入消息，判断你的画面延迟。
5. 结合3和4判断声音延迟。

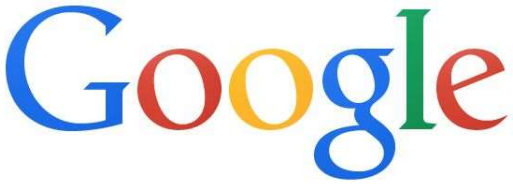
目录

用户交互

- 用户希望可以与应用进行交互
  - 轻触或点击，输入文本，使用手势或者语音
  - 按钮执行操作
  - 以及其他的交互组件来实现数据输入和导航等

用户交互

- 用户交互设计
  - 设计原则：明显，简单，一致
    - 考虑到用户如何使用应用程序
    - 最少化用户需要操作的步骤
    - 使用用户易于接触、理解和使用的UI元素
    - 遵循Android best practices
    - 满足用户的期望



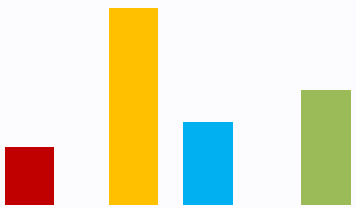
This helps us make the design pixel perfect everywhere it's used, and it allows us to optimize these assets for size and latency, including building a special variant of our full-color logo that is only 305 bytes, compared to our existing logo at ~14,000 bytes. The old logo, with its intricate serifs and larger file size, required that we serve a text-based approximation of the logo for low bandwidth connections. The new logo’s reduced file size avoids this workaround and the consistency has tremendous impact when you consider our goal of making Google more accessible and useful to users around the world, including the next billion.

用户交互

- 用户交互设计
  - 设计原则：明显，简单，一致
    - 考虑到用户如何使用应用程序
    - 最少化用户需要操作的步骤
    - 使用用户易于接触、理解和使用的UI元素
    - 遵循An满足用户的期望
    - droid best practices
  - 这个设计原则适合大部分其他表达场景
    - Simple is elegant
    - KISS
    - 例如写论文

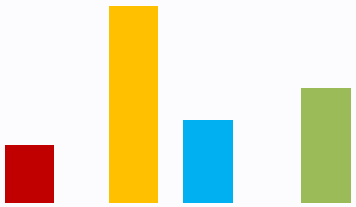
用户交互

- 用户交互设计
  - 设计原则：明显，简单，一致
    - 考虑到用户如何使用应用程序
    - 最少化用户需要操作的步骤
    - 使用用户易于接触、理解和使用的UI元素
    - 遵循满足用户的期望
    - Android best practices
  - 我们不一定能画出最漂亮的图标，但是交互设计不只是图标
    - 先来看一些最简单的交互操作控件



按钮和可点击图像

- 用户交互
- 按钮
- 可点击图片
- 悬浮按钮
- 共同特征



理解用户界面

- 理解用户界面的重要基础是理解事件
  - 按钮点击
  - 手指滑动
  - 双击
  - …
- 这些是如何产生的，如何处理的?

理解事件是理解各种类型编程前提

- 大多是平台上的界面都是以事件为基础
  - Windows界面、各种平台开发也是以事件为基础的
- 不同的操作系统、不同的模式，是对事件处理的方法不同
- 先来看最原始的事件处理方法

按钮

- 按钮
  - 响应用户点击或者按压操作的视图（View）
  - 通常通过文本或图像来告诉用户点击之后的操作
  - 状态：normal，focused，disabled，pressed，on/off

事件处理的流程

- 中断信号产生
- 中断向量
- 中断处理函数
- 找到对应的应用，从上到下来处理这一中断
  - 点击了一个按钮
  - 产生了一个点击事件（中断），
  - 窗口接收中断并处理
  - 窗口里面有没有子窗口来接收
  - 子窗口里面有没有控件来接收
- 例如一个点击事件中断产生是知道对应位置，通过位置可以判断最终是哪个控件来处理这一中断

不同的方式相同点和不同点

- MFC，嵌入式，移动应用开发

```
while(true){
    e = wait_event();
    process_event(e);
}
```

不同的方式相同点和不同点

- MFC，嵌入式，移动应用开发

```
while(true){
    e = wait_event(); //阻塞或者非阻塞
    process_event(e); //多任务？多线程？preemptive
}
```

不同的方式相同点和不同点

- MFC，嵌入式，移动应用开发

```
show_view_elements(); //inflate layout.
while(true){
        e = wait_event(); //阻塞或者非阻塞
        switch(e)
                case e1: process_event1(e);
                case e2: process_event2(e);
                case e3: process_event3(e);
                case e4: process_event4(e);
                …
}
```

不同的方式相同点和不同点

- MFC，嵌入式，移动应用开发

```
show_view_elements(); //inflate layout.
 process_event1(e);
 process_event2(e);
 process_event3(e);
 process_event4(e);
 …
```

如何设计一个按钮

- 画这个按钮的样子

- 增加事件处理
- 对于屏幕上的点击事件
  - 首先判断是不是我这个窗口的事件（判断哪个窗口在前，哪个窗口在后）

如何设计一个按钮

- 画这个按钮的样子

- 增加事件处理
- 对于屏幕上的点击事件
  - 首先判断是不是我这个窗口的事件（判断哪个窗口在前，哪个窗口在后）
  - 然后判断是不是落到我这个空间范围里面
  - 还要判断么？
  - 判断我是不是窗口里面接收事件的最下层控件

按钮图标设置

- 右键点击app/res/drawable
- 选择 **New > Image Asset**
- 从下拉菜单中选择
  Action Bar and Tab Items
- 点击**Clipart:** image
  (the Android logo)

练习:
2. 点击 **New > Vector Asset**

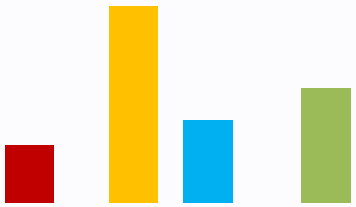
添加点击事件

- Java代码中：使用onClickListener事件监听
- XML中：使用android:onClick属性

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

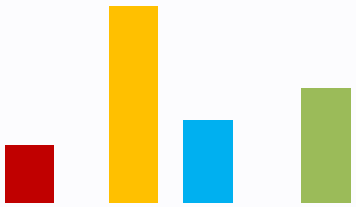
设置事件监听

```
Button button = findViewById(R.id.button);

button.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```



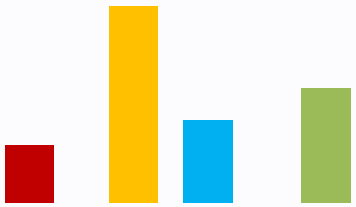
这个代码并不直观

```
button.setOnClickListener(new
View.OnClickListener() {
     public void onClick(View v) {
    // Do something in response to button click
     }
});
```



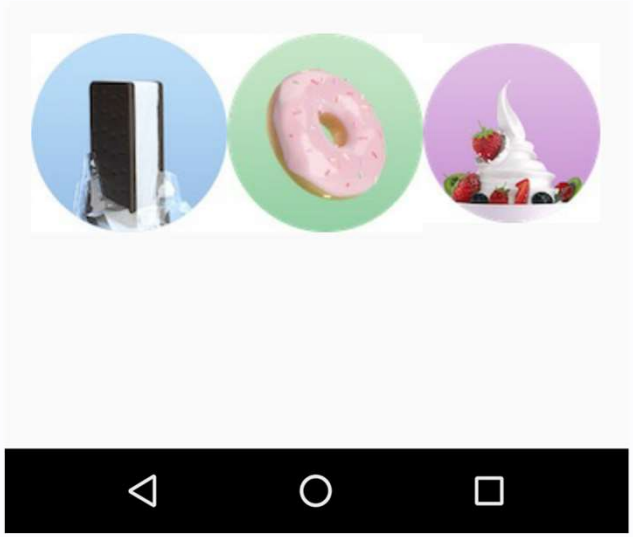
Android常用的控件

- Image View
- FAB
- 输入控件



图像视图（ImageView）

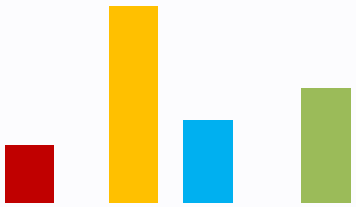
- 图像视图可以添加android:onClick属性
- 图像视图的图像在项目中的位置：
  - app > src > main > res > drawable



为图像视图添加点击响应

- Java代码中：使用onClickListener事件监听
- XML中：使用android:onClick属性

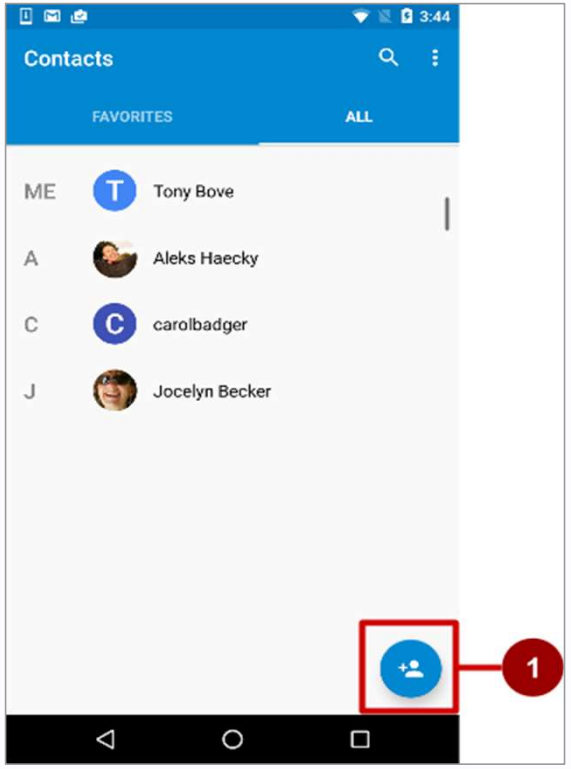
```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/donut_circle"
        android:onClick="orderDonut"/>
```



悬浮按钮（FAB）

- 一般是凸起圆形，漂浮在布局上方
- 屏幕上的主要/进阶操作

例如：添加联系人按钮



使用悬浮按钮

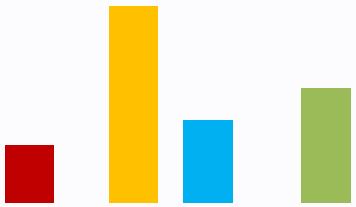
- 创建新项目：Basic Activity template
- 布局：

```
<android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin”
        android:src="@drawable/ic_fab_chat_button_white"
        .../>
```

悬浮按钮大小

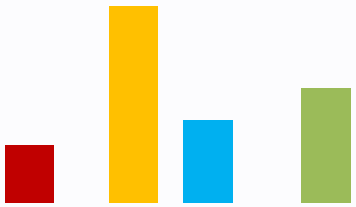
- 默认大小为 56 × 56 dp

- 通过app:fabSize属性设置大小
  - 最小大小（30 × 40 dp）：app:fabSize=“mini”
  - 默认大小（56 × 56 dp）：app:fabSize=“normal”



触屏操作

- 长按（long touch）
- 双击（double-tap）
- 滑动（fling）
- 拖拽（drag）
- 滚屏（scroll）
- 捏合（pinch）

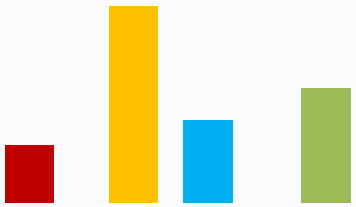


如何检测这些操作?

- onTap
- onTouch
- onUnTouch
- 屏幕知道手指按了没有，按在哪里了。

事件类型

- 长按（long touch）
- 双击（double-tap）
- 滑动（fling）
- 拖拽（drag）
- 滚屏（scroll）
- 这些在Windows编程里面都能够实现



检测手势

- Android提供类和方法来帮助开发者处理各种手势
  - GestureDetectorCompat类来处理各种手势
  - MotionEvent类处理移动事件

检测所有类型的手势

- 将所有触屏事件的数据汇集
- 解释数据，看看它是否符合应用程序支持的任何手势的标准。
  - Android developer documentation

了解更多（官方文档）

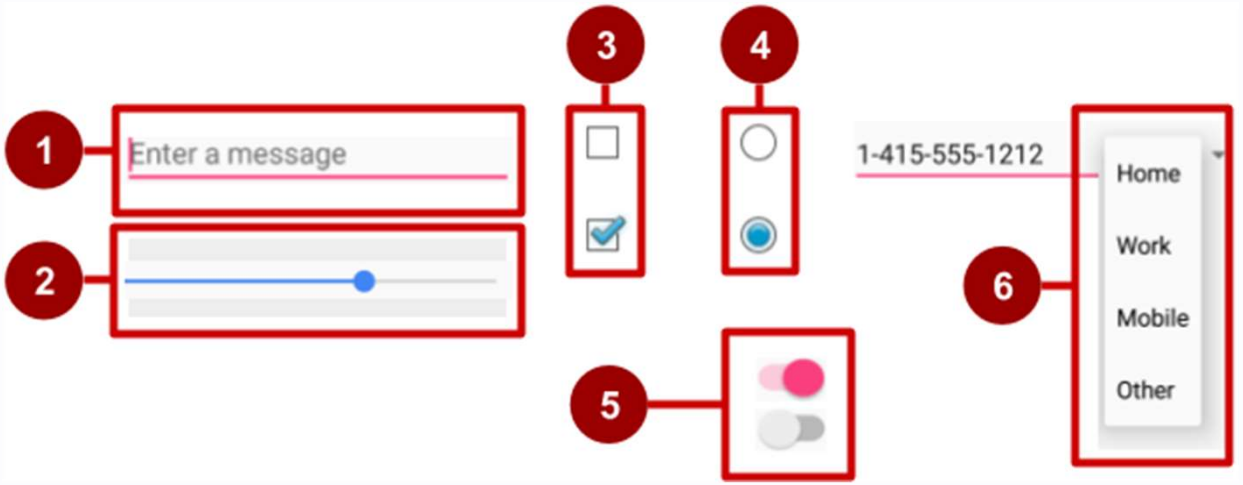
- Input Controls
- Drawable Resources
- Floating Action Button
- Radio Buttons
- Specifying the Input Method Type
- Handling Keyboard Input
- Text Fields
- Buttons
- Spinners
- Dialogs
- Fragments
- Input Events
- Pickers
- Using Touch Gestures
- Gestures design guide

输入控件

- 输入控件概览
- 视图焦点
- 文本和数字
- 提供选项

输入控件示例

1. EditText
2. SeekBar
3. CheckBox
4. RadioButton
5. Switch
6. Spinner



视图（View）是输入控制的基础类

- View类是所有UI组件（包括输入控件）的基本构建块
- View是提供交互式UI组件的类的基类
- View提供基础交互的方式android:onClick

接受用户输入

- 自定义文本和数字输入：EditText（使用键盘输入）
- 提供选项：CheckBox，RadioButton，Spinner
- 打开/关闭：Toggle，Switch
- 从范围中选择值：SeekBar

输入控件（课后学习）

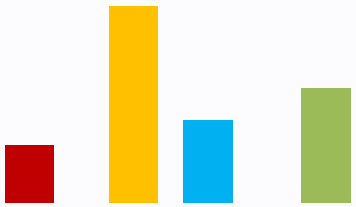
- EditText允许用户通过键盘进行输入
- SeekBar允许用户左右滑动来进行设值
- CheckBox允许用户从多个选择中做出多个选择
- RadioButton结合RadioGroup对选项进行bool选择
- Switch允许用户进行开关切换
- Spinner允许用户从列表中选择单个选项

焦点（Focus）

- 接受用户输入的视图有“Focus（焦点）”
- 只有一个视图具有焦点
- 焦点明确哪一个视图接受输入
- 焦点的指定：
  - 用户点击一个视图
  - APP通过Return，Tab或箭头等引导用户从一个文本输入到下一个
  - 在可以焦点化的视图上调用requestFocus()

可点击 V.S. 可焦点

- 可点击（clickable）：视图可以对点击等事件进行相应
- 可焦点（focusable）：视图可以获得焦点以接受输入
- 输入控件（如键盘）将输入发送给具有焦点的视图



下一个获得焦点的视图?

- 触屏/点击的最上层的视图
- 用户提交输入之后，焦点移到最近的相邻视图，优先级：从左到右，从上到下
- 当用户和方向控件交互式可以改变焦点

用户引导

- 直观地指示哪个视图具有焦点，以便用户知道其输入的位置
- 直观地指示用户哪个视图可以具有焦点，以便用户知道焦点导航
- 可以预测且具有逻辑性——没有惊喜！

焦点引导（课后学习）

- 按照你规划的焦点移动在一个布局中从左到右，从上到下来安排输入控件
- 在布局中，将输入控件放在一个视图组中
- 在XML中指定焦点顺序

```
android:id="@+id/top"
android:focusable="true"
android:nextFocusDown="@+id/bottom"
```

精确设置焦点（课后学习）

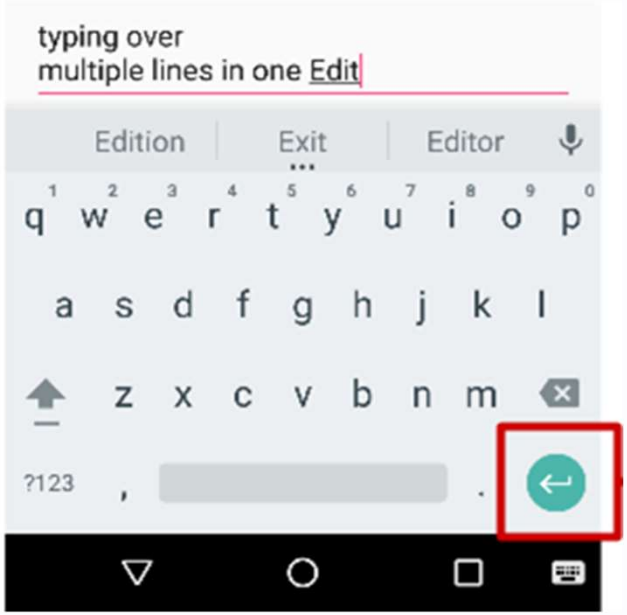
- 使用View类提供的方法来设置焦点
  - setFocusable()设置一个视图是否可以获得焦点
  - requestFocus()将焦点赋予一个指定的视图
  - setOnFocusChangeListener()焦点发生变化时（赋予/失去焦点）的事件监听
  - onFocusChanged()当某一个视图的焦点发生变化时调用

获取具有焦点的视图（课后学习）

- ViewGroup.getFocusedChild()
- Activity.getCurrentFocus()

多行自定义文本

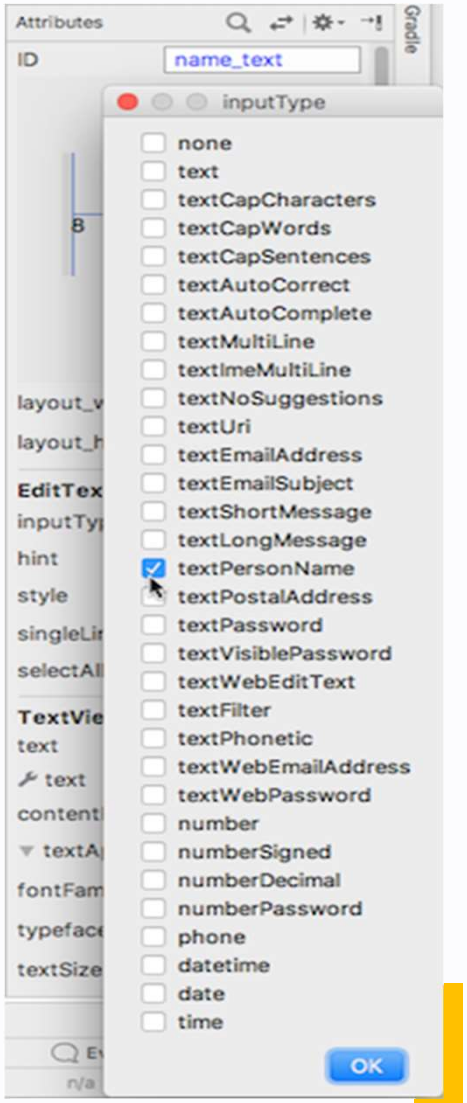
- EditText默认
- 键盘（字母数字键盘）
- 点按回车键（Return/Enter）换行



自定义输入设置（课后学习）

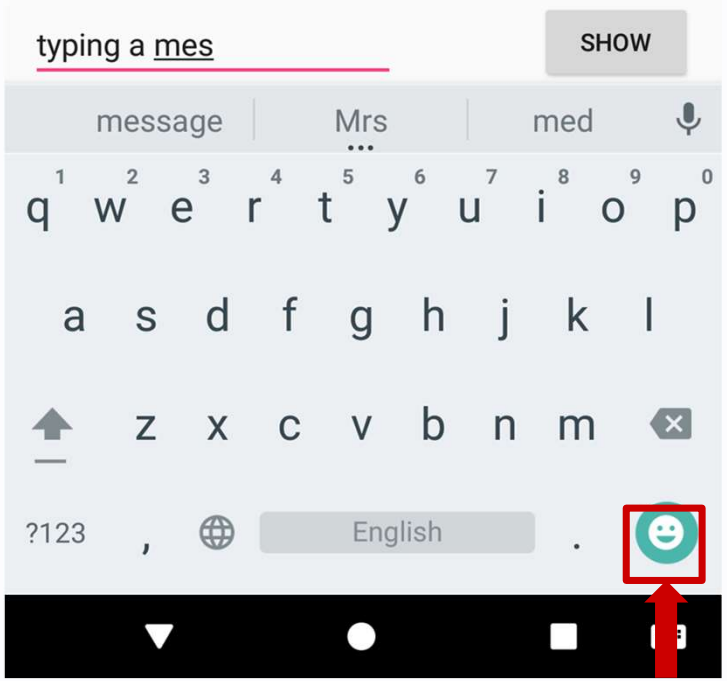
- 在布局编辑器的属性窗口中设置
- 在XML中设置EditText的属性

```
<EditText
      android:id="@+id/name_field"
      android:inputType =
                        "textPersonName"
      ...
```



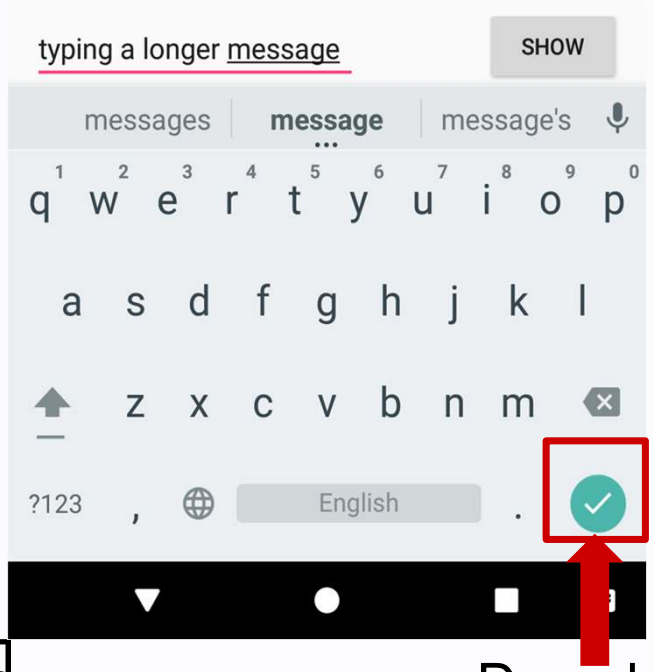
编辑消息（课后学习）

- `android:inputType="textShortMessage"`
- 单行消息
- 点击表情键切换到表情输入



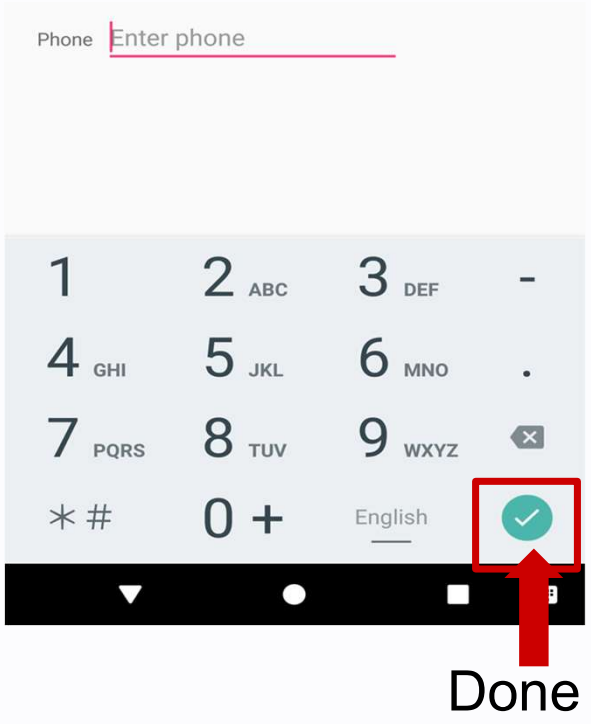
单行编辑（课后学习）

- 两种方式都可以
  - android:inputType
    ="textLongMessage"
  - android:inputType
    ="textPersonName"
- 单行的文本
- 点击完成键使焦点移动到下一个视图



编辑电话号码输入（课后学习）

- `android:inputType ="phone"`
- 数字键盘（只有数字）
- 点击完成按键使焦点移动到下一视图



获取文本（课后学习）

- 获取编辑的编辑文本对象
  - `EditText simpleEditText = findViewById(R.id.edit_simple);`
- 检索CharSequence并将其转换为字符串
  - `String strValue = simpleEditText.getText().toString();`

常用的输入类型（课后学习）

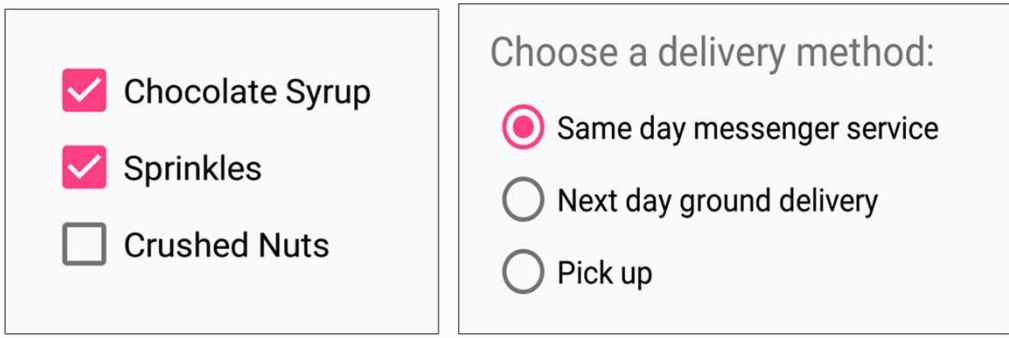
- textCapCharacters：大写字母
- textCapSentences：首字母大写
- textPassword：隐藏显示密码
- number：限制输入数字

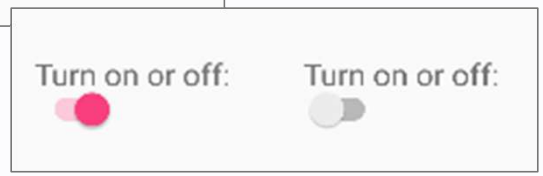
常用的输入类型（课后学习）

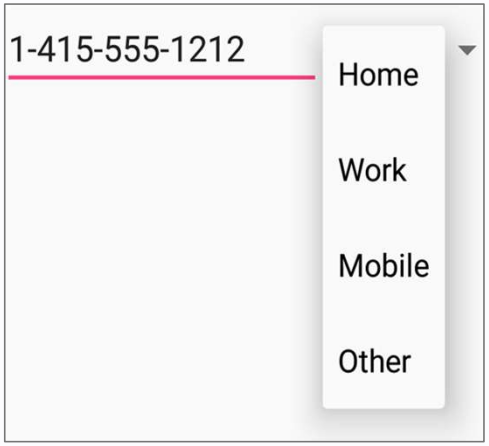
- textEmailAddress: 显示具有@便捷键的键盘
- phone: 显示数字电话键盘
- datetime:显示带有斜线和冒号的数字键盘以输入日期和时间

提供选择的UI控件

- CheckBox和RadioButton
- ToggleButton和Switch
- Spinner



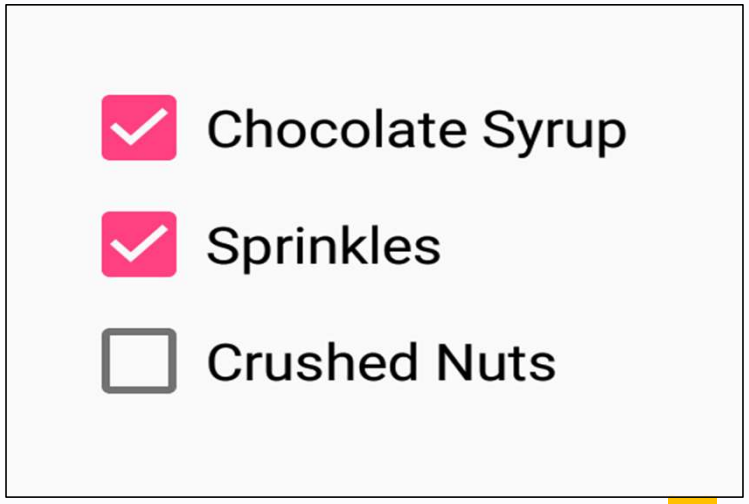




CheckBox （课后学习）

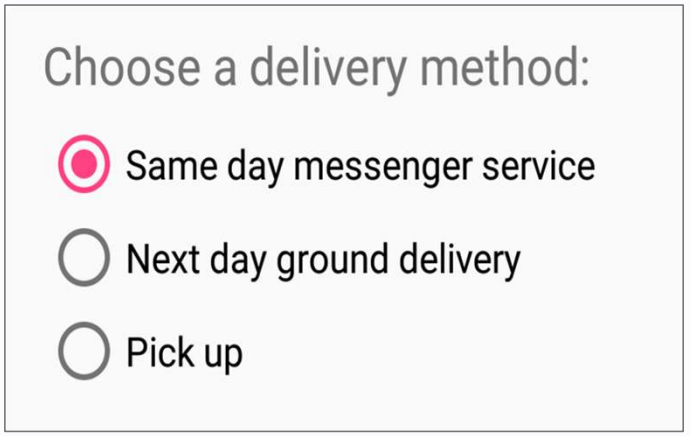
- 用户可以选择多个选项
- 选择一个选项不会改变其他的选项
- 表现形式是垂直列表
- 经常和提交按钮一起使用
- 每个CheckBox都是一个View

可以为其添加onClick监听



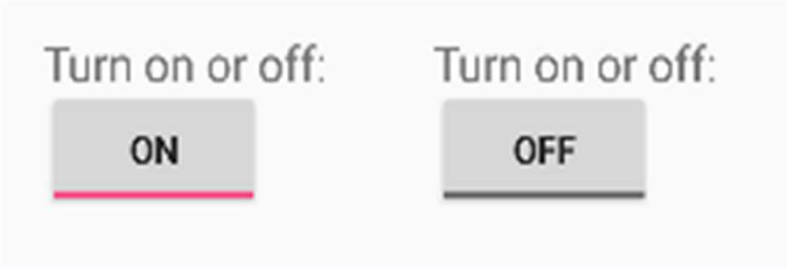
RadioButton （课后学习）

- 将RadioButton放在一个垂直的RadioGroup中（也可以水平放置）
- 用户只可以选择一个选项
- 勾选一个会取消其他选项的勾选
- 通常会和提交按钮一起使用

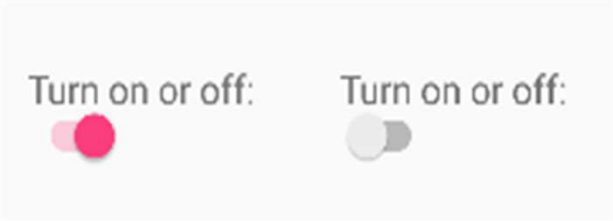


Toggle button和Switch

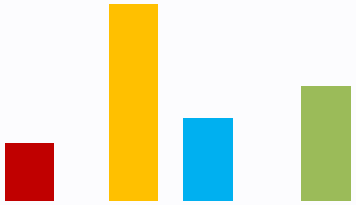
- 用户可以在on或off状态切换
- 用android:onClick添加点击监听



Toggle buttons



Switches



了解更多

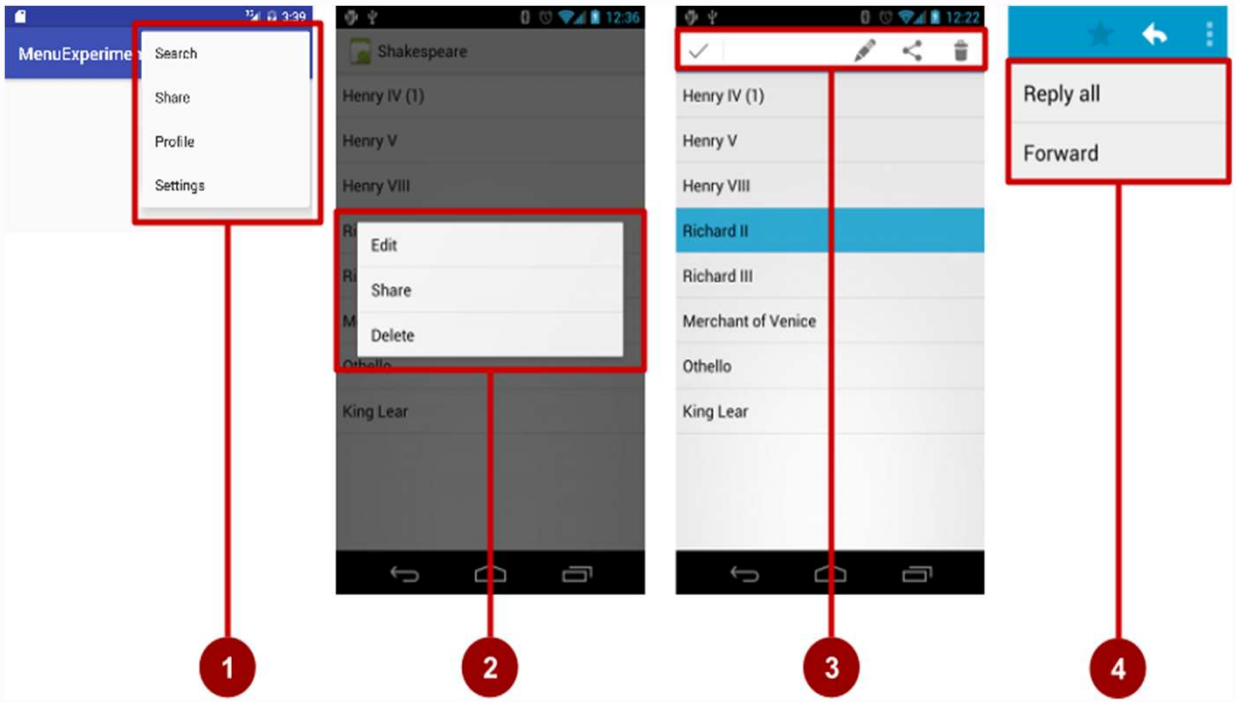
- Input Controls
- Radio Buttons
- Specifying the Input Method Type
- Handling Keyboard Input
- Text Fields
- Spinners

菜单和选择器

- 各种不同菜单
- 对话框
- 选择器

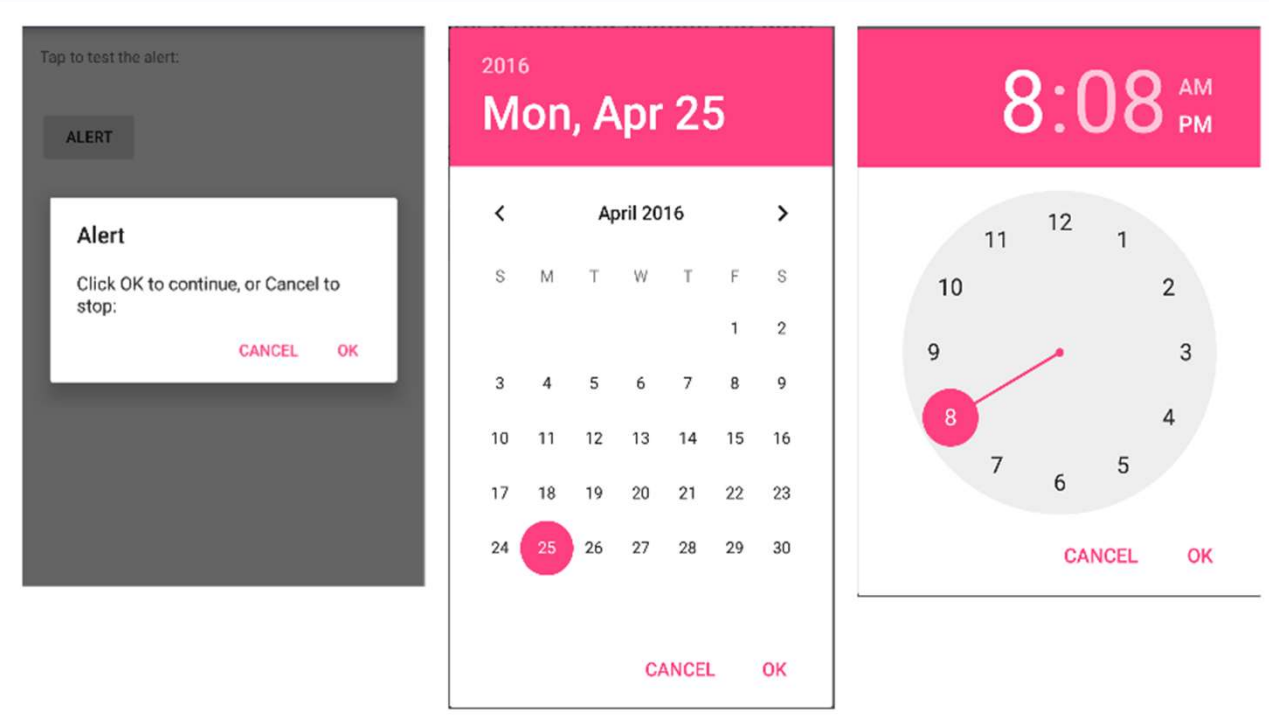
菜单类型

- 1. 带选择的应用栏（App bar with options menu）
- 2. 上下文菜单（context menu）
- 3. 上下文操作栏（contextual action bar）
- 4. 弹出菜单（popup menu）



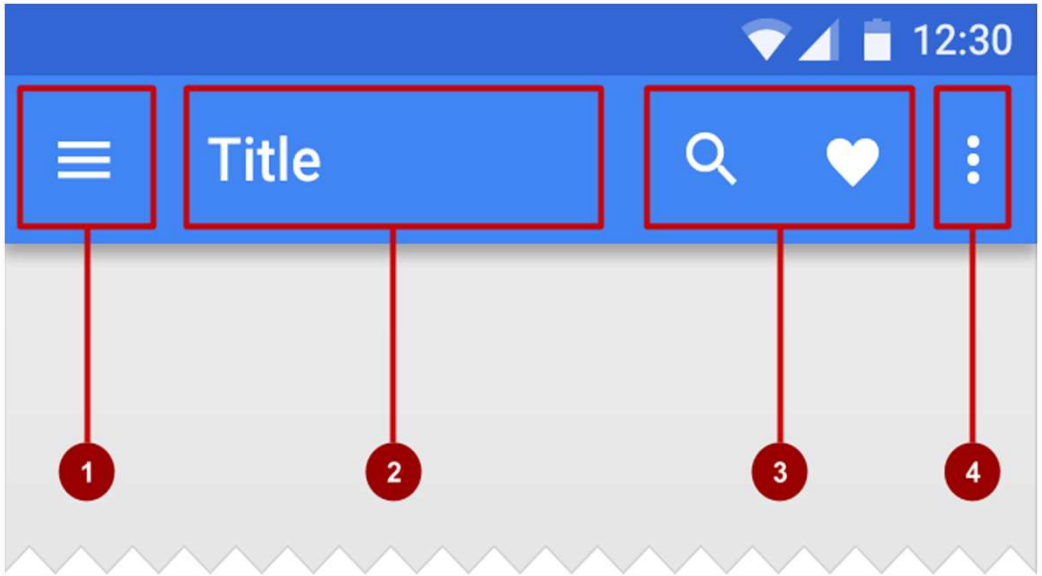
对话框和选择器

- 1. 弹出对话框
- 2. 日期选择器
- 3. 时间选择器



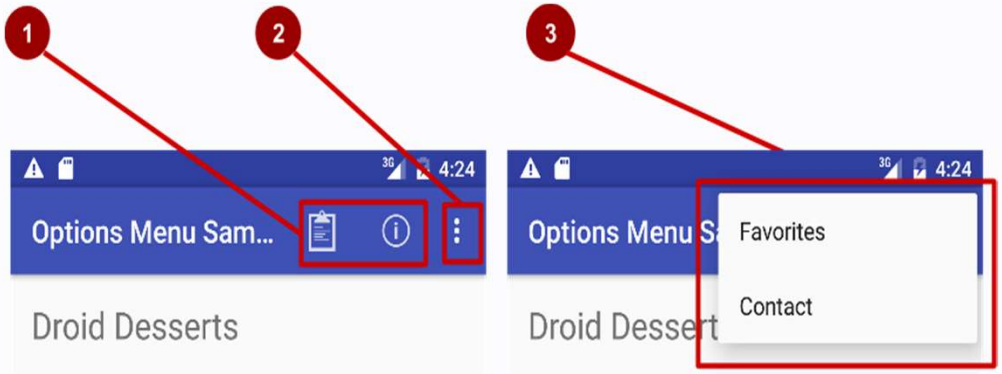
什么是应用栏

- 应用中屏幕最上方的栏
  - 1. 打开导航的导航按钮
  - 2. 当前Activity的名称
  - 3. 选项菜单项的图标
  - 4. 其他图标的选项菜单



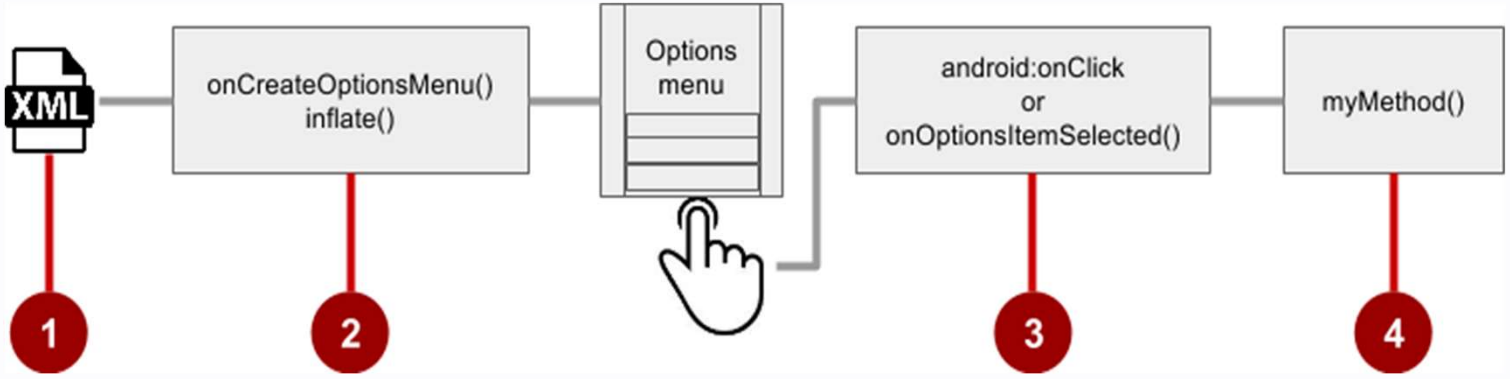
什么是选项菜单

- 应用程序栏中重要项目的操作图标（1）
- 点击三个点的按钮，打开菜单（2）
- 在应用栏的右上角显示
- 导航到其他Activity或者编辑APP设置



实现选项菜单的步骤

- 1. XML菜单资源（menu_main.xml）
- 2. onCreateOptionsMenu()中加载（inflate）菜单选项
- 3. onClick或者onOptionsItemSelected()
- 4. 处理选项点击



创建菜单资源

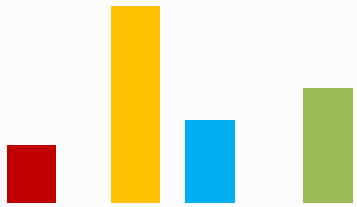
- 1. 创建菜单资源路径
- 2. 创建XML文件(menu_main.xml)
- 3. 添加菜单选项（例：Settings 和 Favorites）

```
<item  android:id="@+id/option_settings"
       android:title="Settings" />
<item  android:id="@+id/option_favorites"
       android:title="Favorites" />
```

加载选项菜单

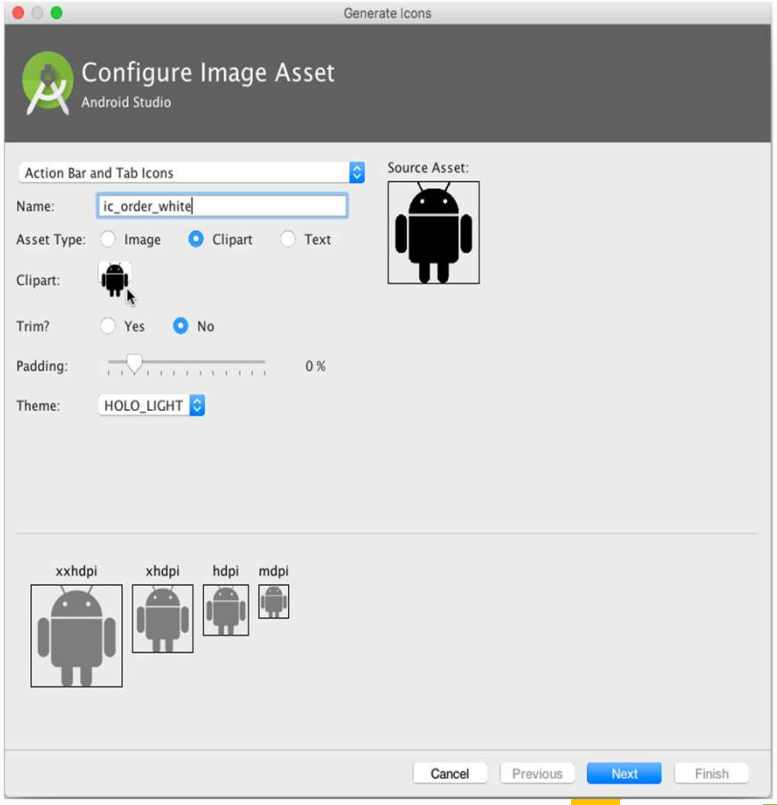
- 在Activity中重写onCreateOptionsMenu()

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    getMenuInflater().inflate(R.menu.menu_main,
menu);
    return true;
}
```



给菜单选项添加图标

- 1. 右击drawable
- 2. 选择New > Image Asset
- 3. 选择Action Bar and Tab Items
- 4. 编辑图标名称
- 点击clipart image，点击图标
- 点击Next > Finish

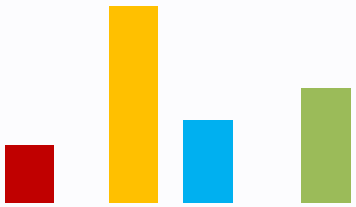


添加菜单选项的属性

```
<item
        android:id="@+id/action_favorites"
        android:icon="@drawable/ic_favorite"
        android:orderInCategory="30"
        android:title="@string/action_favorites"
        app:showAsAction="ifRoom" />
```

重写onOptionsItemSelected()

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            showSettings();
            return true;
        case R.id.action_favorites:
            showFavorites();
            return true;
        default:
                return super.onOptionsItemSelected(item);
        }
}
```

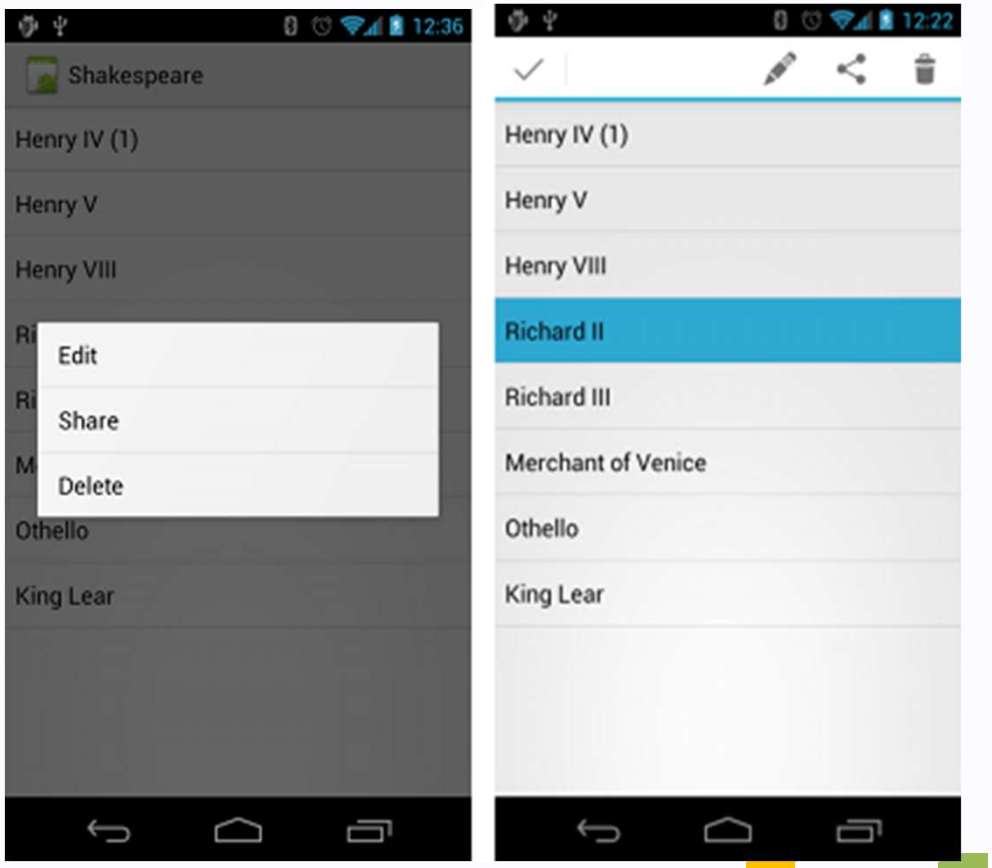


什么是上下文菜单

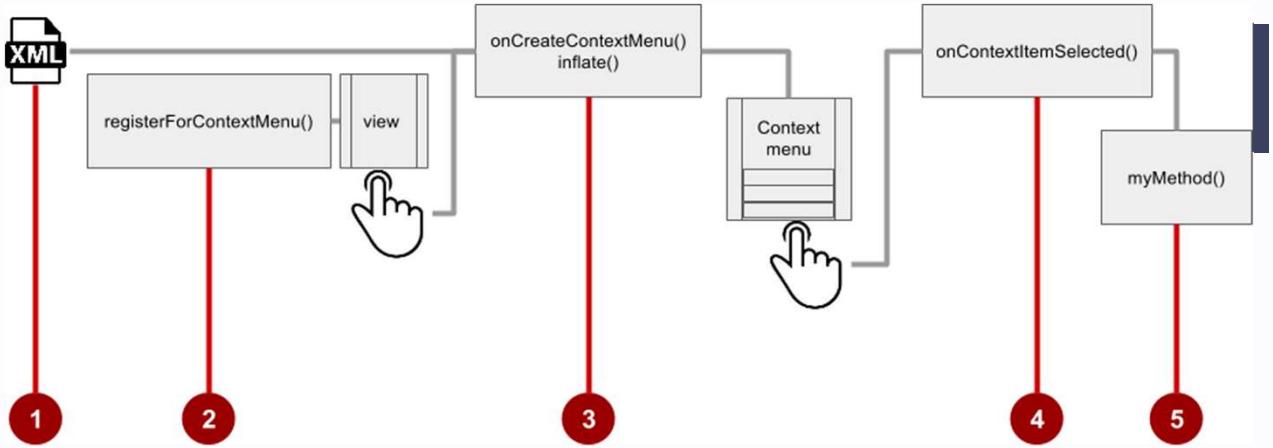
- 允许用户在选择的视图上进行操作
- 可以布置在任何视图
- 经常被使用在RecyclerView， GridView或者其他View集合上

上下文菜单的类型

- 悬浮式-长按一个视图
  - 用户可以编辑视图
  - 用户同时只能操作一个视图
- 上下文操作式-临时的操作栏
  - 操作选项可以影响选择的视图
  - 用户可以在多个视图上操作



悬浮式上下文菜单



- 1、为菜单创建XML文件，设置外观和属性
- 2、使用registerForContextMenu()注册View
- 3、在Activity中实现onCreateContextMenu()
- 4、实现onContextItemSelected()来处理点击
- 5、创建每一个选项被点击后要执行的方法

创建菜单资源

- 1、创建菜单的XML文件（menu_context.xml)

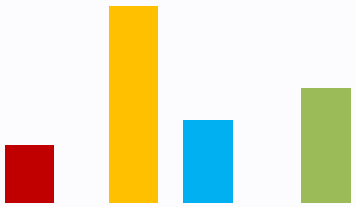
```
<item
    android:id="@+id/context_edit"
    android:title="Edit"
    android:orderInCategory="10"/>

<item
    android:id="@+id/context_share"
    android:title="Share"
    android:orderInCategory="20"/>
```

给上下文菜单注册视图

在onCreate()中

2、

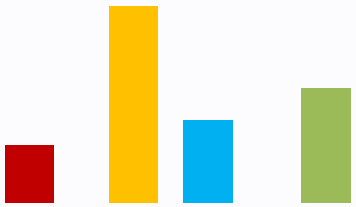
```
TextView article_text =
findViewById(R.id.article);
registerForContextMenu(article_text);
```



实现onCreateContextMenu()

- 3、指定上下文菜单

```
@Override
public void onCreateContextMenu(ContextMenu menu,
View v, ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_context, menu);
}
```

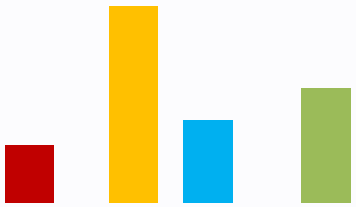


实现onContextItemSelected()

```
@Override
public boolean onContextItemSelected(MenuItem item)
{
    switch (item.getItemId()) {
        case R.id.context_edit:
            editNote();
            return true;
        case R.id.context_share:
            shareNote();
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

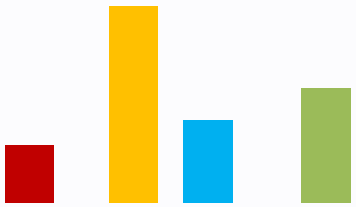
上下文操作栏（contextual action bar）

- 什么是Action Mode
- 一种可以允许在当前视图下进行修改的正常UI交互的模式
- 例如：选择文本的一部分或者长按某一个元素可以触发action mode



Action mode具有生命周期

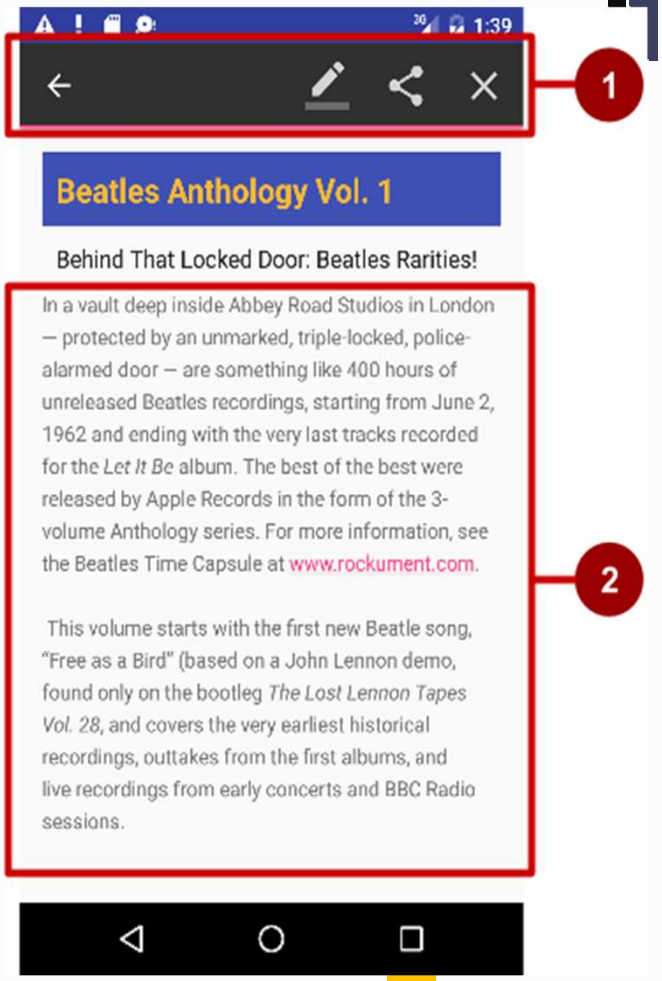
- 通过startActionMode()开始
- ActionMode.Callback接口中提供可以重写的生命周期函数
  - onCreateActionMode(ActionMode, Menu)
  - onPrepareActionMode(ActionMode, Menu)
  - onActionItemClicked(ActionMode, MenuItem)
  - onDestroyActionMode(ActionMode)



什么是contextual action bar

- 长按一个视图可以显示contextual action bar

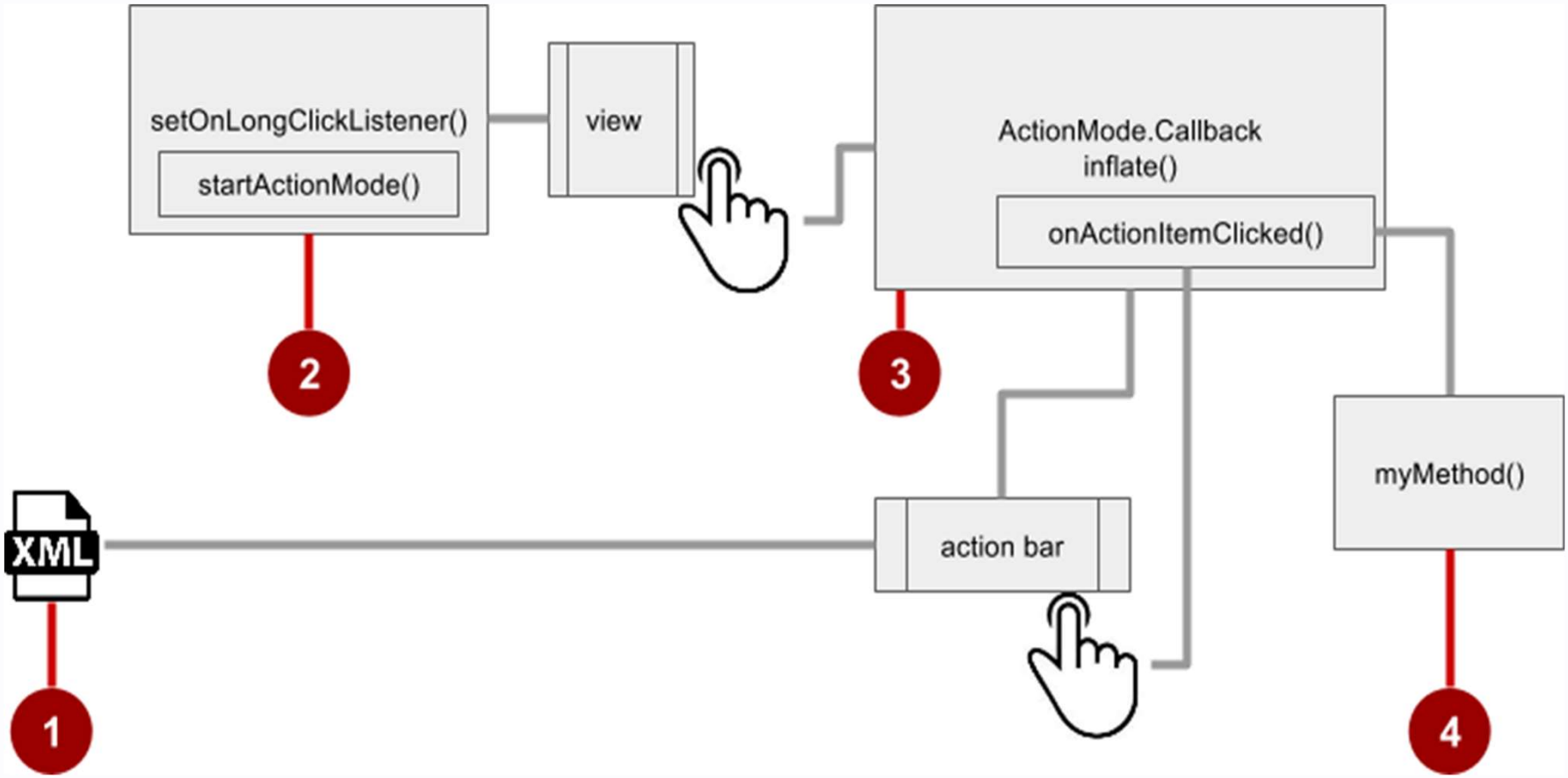
1. 带有操作的contextual action bar
   - 编辑，分享，删除
   - 完成（左箭头）
   - 在用户点击完成前可以使用
2. 长按可以触发contextual action bar的视图



contextual action bar的步骤

- 1、创建XML文件并设置图标
- 2、添加触发setOnLongClickListener()和点击处理startActionMode()
- 3、实现ActionMode.Callback接口来处理action mode的生命周期
- 4、为每个元素点击创建对应的方法

步骤


setOnLongClickListener()
startActionMode()
view
ActionMode.Callback
inflate()
onActionItemClicked()
myMethod()
XML
action bar
1
2
3
4

setOnLongClickListener

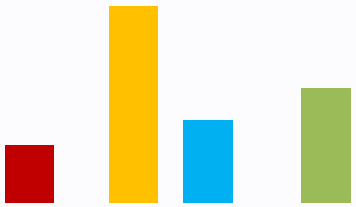
```
private ActionMode mActionMode;
```

在onCreate():

```
View view = findViewById(article);
view.setOnLongClickListener(new View.OnLongClickListener() {
   public boolean onLongClick(View view) {
      if (mActionMode != null) return false;
      mActionMode =
            MainActivity.this.startActionMode(mActionModeCallback);
      view.setSelected(true);
      return true;
   }
});
```

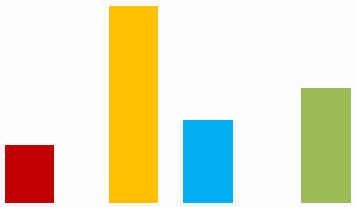
实现mActionModeCallback

```
public ActionMode.Callback
mActionModeCallback =
    new ActionMode.Callback() {
     // Implement action mode callbacks here.
};
```



实现onCreateActionMode

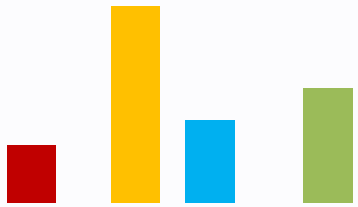
```
@Override
public boolean onCreateActionMode(ActionMode mode,
Menu menu) {
    MenuInflater inflater = mode.getMenuInflater();
    inflater.inflate(R.menu.menu_context, menu);
    return true;
}
```



实现onPrepareActionMode

- 当action mode显示时被调用
- 总是在onCreateActionMode之后被调用，如果action mode不可用则可能被多次调用

```
@Override
public boolean onPrepareActionMode(ActionMode
mode, Menu menu) {
    return false; // Return false if nothing is
done.
}
```



实现onActionItemClicked

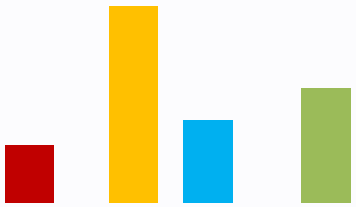
- 当用户选择一个action后被调用
- 在这个方法中处理点击事件

```
@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_share:
          // Perform action for the Share menu item.
          mode.finish(); // Action picked, so close the action bar.
          return true;
        default:
          return false;
    }
}
```

实现onDestroyActionMode

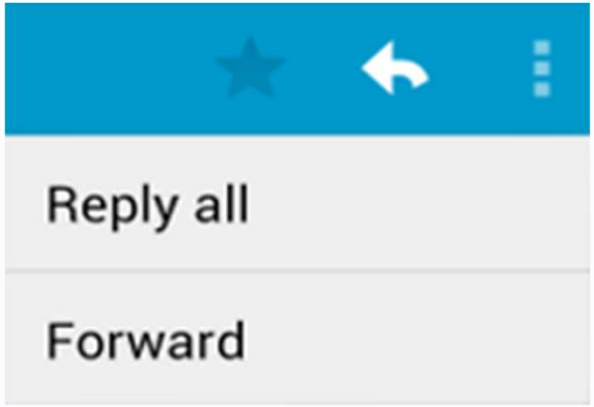
- 当用户退出action mode时调用

```
@Override
public void onDestroyActionMode(ActionMode
mode) {
    mActionMode = null;
}
```



什么是弹出菜单

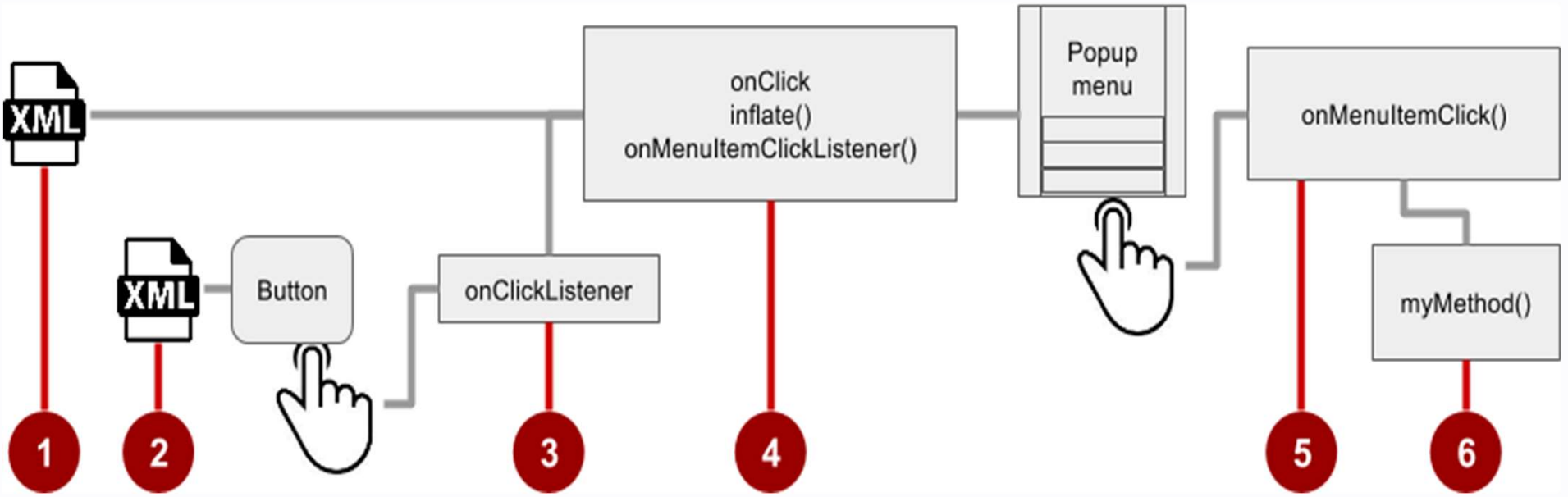
- 绑定在一个视图上的垂直列表
- 通常绑定在一个可见的图标上
- 操作不会直接影响视图的内容
  - 图标打开菜单
  - 在email应用中，回复等操作和邮件相关，但是不会影响或者操作信息



步骤

- 创建菜单XML文件，设置外观和位置等属性
- 在activity的布局文件中添加弹出菜单的图标按钮
- 给弹出菜单的图标添加点击监听
- 重写onClick()加载弹出菜单并且为其注册监听 `onMenuItemClickListener()`
- 实现`onMenuItemClick()`
- 创建每个`item`被点击后的方法

步骤


XML
XML
Button
onClickListener
onClick
inflate()
onMenuItemClickListener()
Popup
menu
onMenuItemClick()
myMethod()
1
2
3
4
5
6

添加图标按钮（ImageButton）

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button_popup"

android:src="@drawable/@drawable/ic_action_popup"/>
```

给按钮添加监听

```
private ImageButton mButton =
    (ImageButton) findViewById(R.id.button_popup);
```

在onCreate():

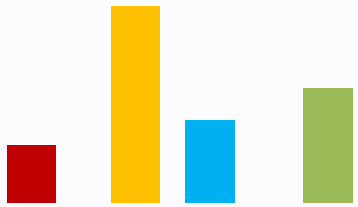
```
mButton.setOnClickListener(new View.OnClickListener()
{
    // define onClick
});
```

实现onClick()

```
@Override
public void onClick(View v) {
    PopupMenu popup = new PopupMenu(MainActivity.this,
mButton);
    popup.getMenuInflater().inflate(
        R.menu.menu_popup, popup.getMenu());
    popup.setOnMenuItemClickListener(
        new PopupMenu.OnMenuItemClickListener() {
           // implement click listener.
        });
   popup.show();
}
```

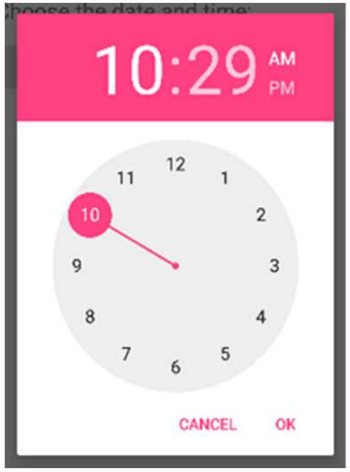
实现onMenuItemClick

```
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.option_forward:
            // Implement code for Forward button.
            return true;
        default:
            return false;
    }
}
```



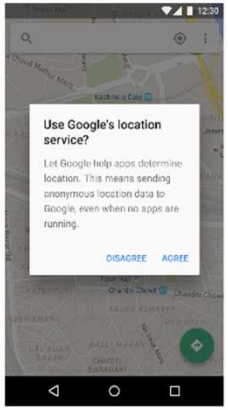
对话框

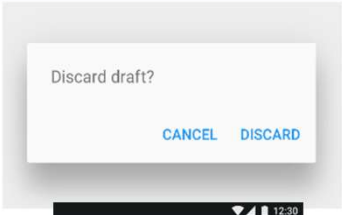
- 对话框Dialog会弹出，阻断正在进行的Activity
- 需要用户操作来关闭

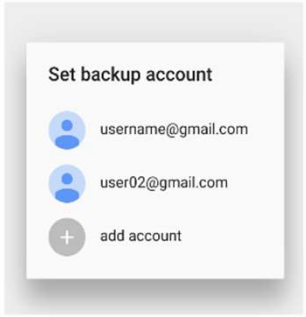


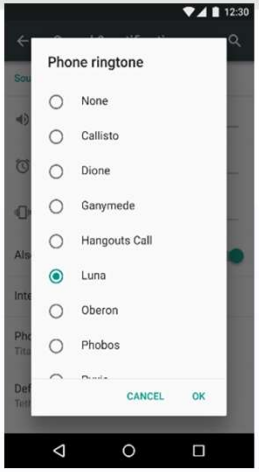
TimePickerDialog

DatePickerDialog





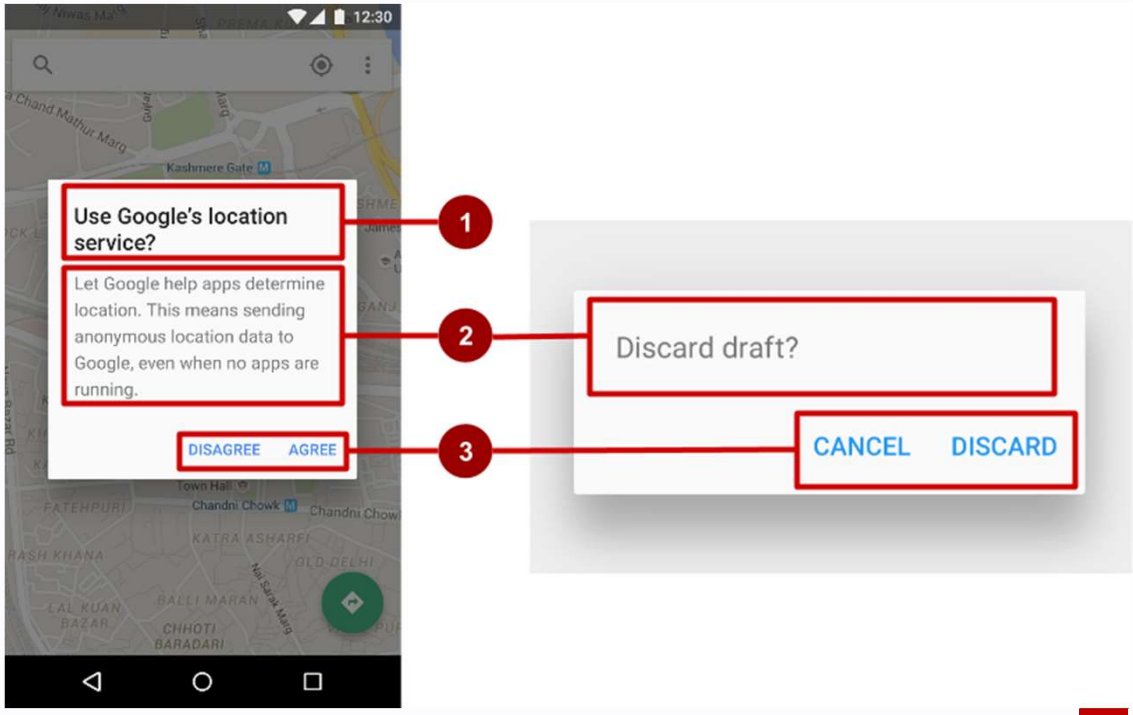




AlertDialog

AlertDialog

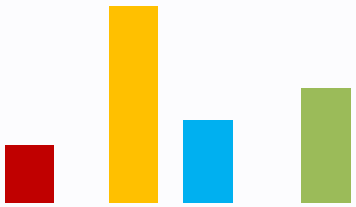
- AlertDialog可以显示：
  - 标题（Title，可选）
  - 内容区域
  - 操作按钮



创建AlertDialog

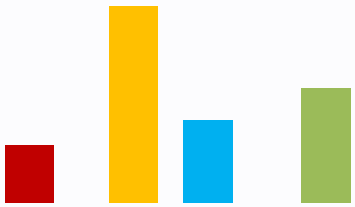
- AlertDialog.Builder创建和设置属性

```
public void onClickShowAlert(View view) {
   AlertDialog.Builder alertDialog = new
AlertDialog.Builder(MainActivity.this);
   alertDialog.setTitle("Connect to Provider");
   alertDialog.setMessage(R.string.alert_message);
   // ... Code to set buttons goes here.
```



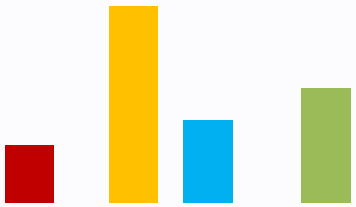
设置按钮

- alertDialog.setPositiveButton()
- alertDialog.setNeutralButton()
- alertDialog.setNegativeButton()



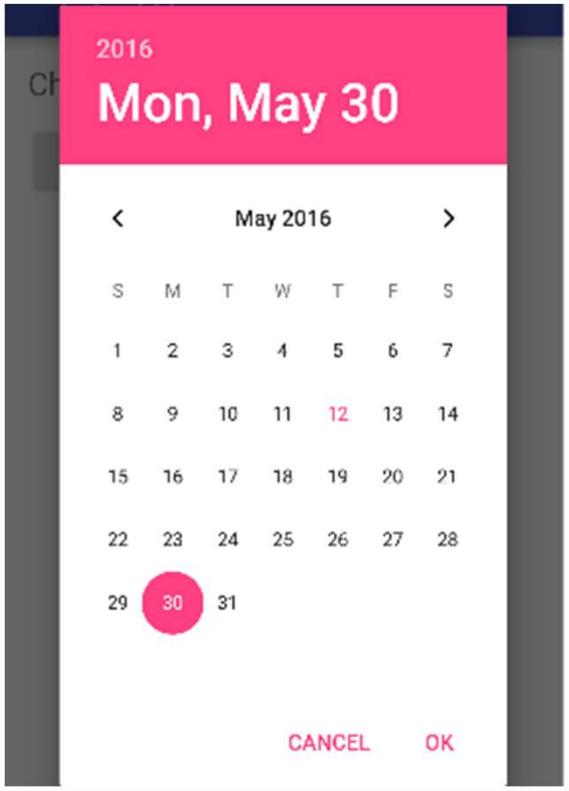
alertDialog代码示例

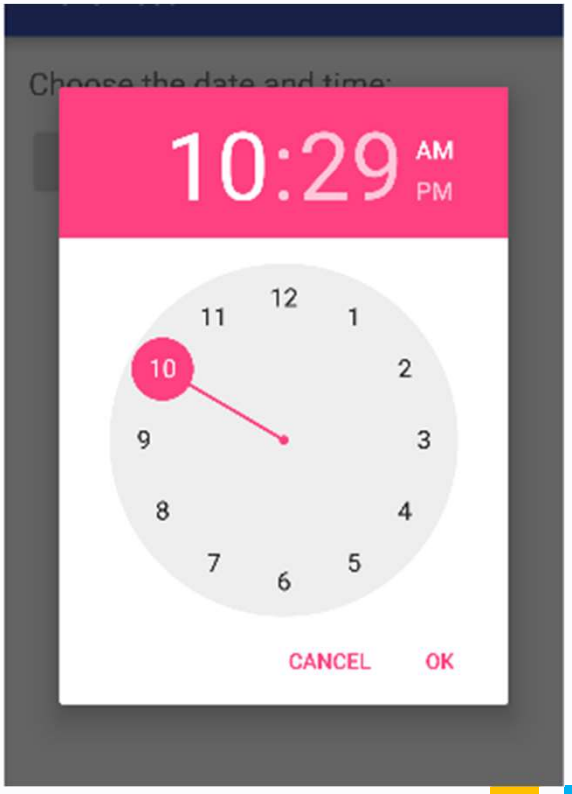
```
alertDialog.setPositiveButton(
"OK", new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface
dialog, int which) {
        // User clicked OK button.
    }
});
```



选择器（Pickers）

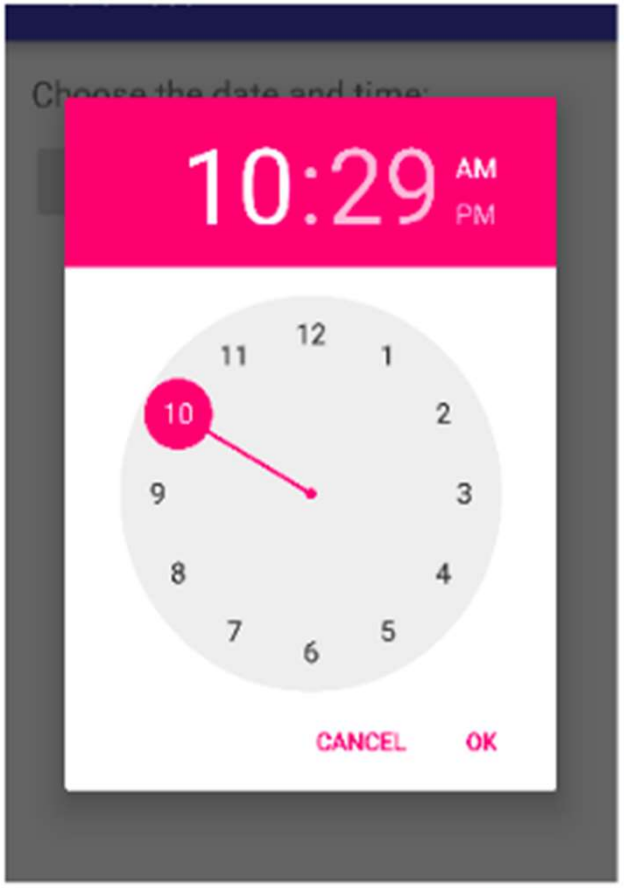
- DatePickerDialog
- TimePickerDialog





通过Fragment实现的选择器

- 用DialogFragment来显示Picker
- DialogFragment在Activity窗口上悬浮的窗口



Fragment介绍

- Fragment在Activity中相当于小型Activity
  - 有自己的生命周期
  - 接收自己输入事件
- 可以在父Activity运行时添加或者移除
- 多个Fragments可以结合在一个Activity中
- 可以在多个Activity中重复使用

使用Fragment的步骤

- 创建Fragment的布局文件
- 创建继承Fragment或Fragment子类的类，在这个类的onCreate()中加载布局文件
- 在需要加载Fragment的Activity中静态/动态加载等

举例：创建Fragment布局文件

- 正常创建Fragment文件，fragment.xml

```
<?xml version= “1.0” encoding= “utf-8” ?>
<android.support.constraint.ConstraintLayout
xmlns:android= “http://schemas.android.com/apk/res/android”
    xmlns:app= “http://schemas.android.com/apk/res-auto”
    xmlns:tools= “http://schemas.android.com/tools”
    android:id= “@+id/fragmentID”
    android:layout_width= “match_parent”
    android:layout_height= “match_parent” >

    <TextView
        android:id= “@+id/constraint_title”
        ……/>
     ……
</android.support.constraint.ConstraintLayout>
```

举例：创建继承Fragment的类

- 类名和布局文件一致FragmentID
- 重写onCreateView函数
- @Override
  public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, Bundle savedInstanceState){
  View view = inflater.inflate(R.layout. **fragmentID**, container, **false**);
  return view;
  }

举例：静态加载

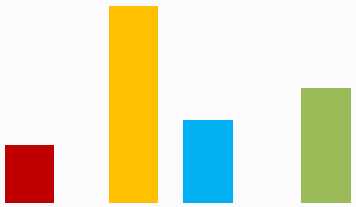
- 在需要加载Fragment的Activity的布局文件中
  - <fragment
    android:id=“@+id/fragmentID" // id必须要有
    android:name="com.xx.testactivity.fragment.FragmentID"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"/>

举例：动态加载

- 获取FragmentManager
  - FragmentManager manager = getSupportFragmentManager();
- 获取Transaction
  - FragmentTransaction transaction = manager.beginTransaction();
- 加载
  - transaction.replace(R.id.container_id, new FragmentID());
- 提交
  - transaction.commit();

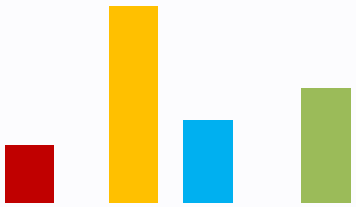
作用

- 使用Fragment在横竖屏时切换布局



4.4 用户导航

- 后退导航Back navigation
- 分层导航Hierarchical navigation
  - up navigation
  - Descendant navigation
- 用于descendant navigation的导航
  - 列表list和轮播carousel
  - 向上导航ancestral navigation
  - 横向导航lateral navigation

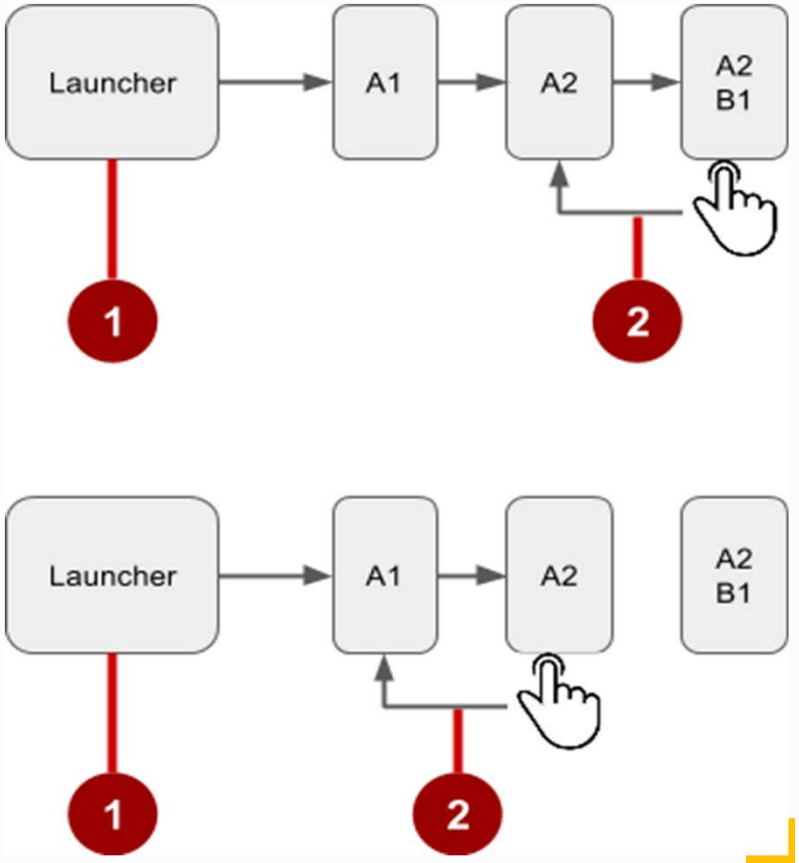


两种导航

- 向后导航（back navigation）
  - 由设备的后退按钮提供
  - 有安卓退栈控制
- 向上导航（up navigation）
  - 在应用栏提供的向上按钮
  - 通过在AndroidManifest.xml中为子活动定义父活动来控制

浏览屏幕记录

- 记录从launcher开始
- 用户点击后退◁按钮导航到前一个屏幕



改变后退按钮的行为

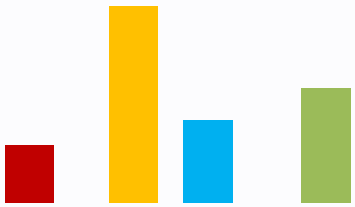
- android系统管理退栈和后退按钮
- 如有疑问，请不要更改
- 如果为了满足用户需求必须更改，仅需要重写（override）

例如：

在嵌入式浏览器中，当用户按下设备的“后退”按钮时，触发浏览器的默认后退行为

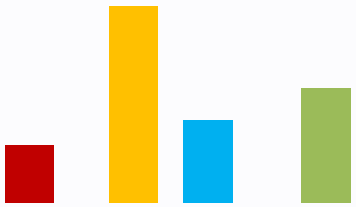
重写onBackPressed()

```
@Override
public void onBackPressed() {
    // Add the Back key handler here.
    return;
}
```



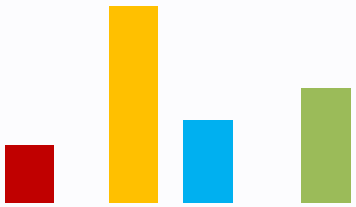
分层导航模式

- 父屏幕——可以导航向子屏幕的屏幕，如home和主Activity
- collection sibling——可以导航向子屏幕合集的屏幕，如新闻列表
- section sibling——显示内容（如故事）的屏幕

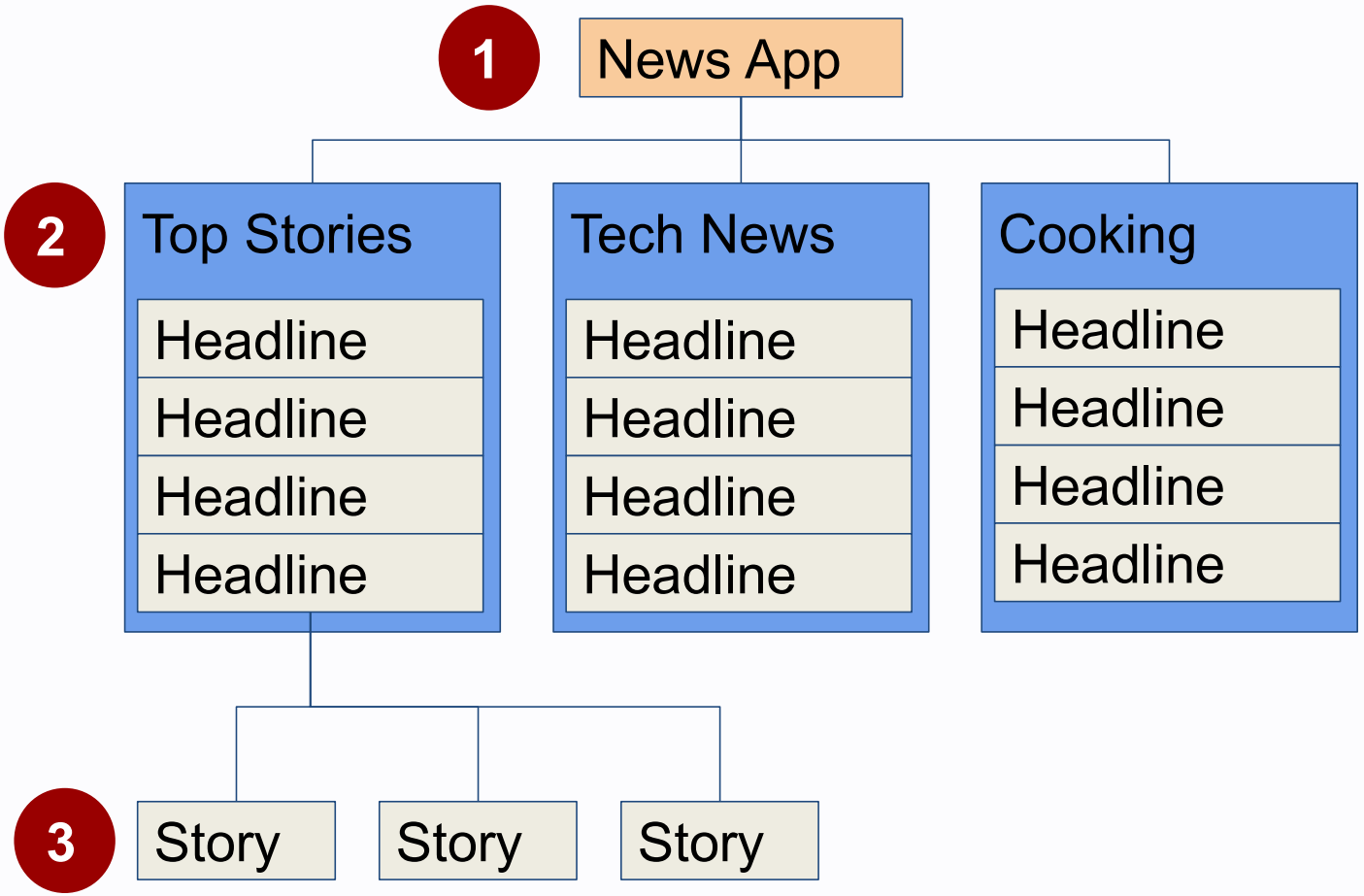


屏幕等级示例

- 1. 父（parent）屏幕
- 2. 子屏幕：collection siblings
- 3. 子屏幕：section siblings



屏幕等级示例

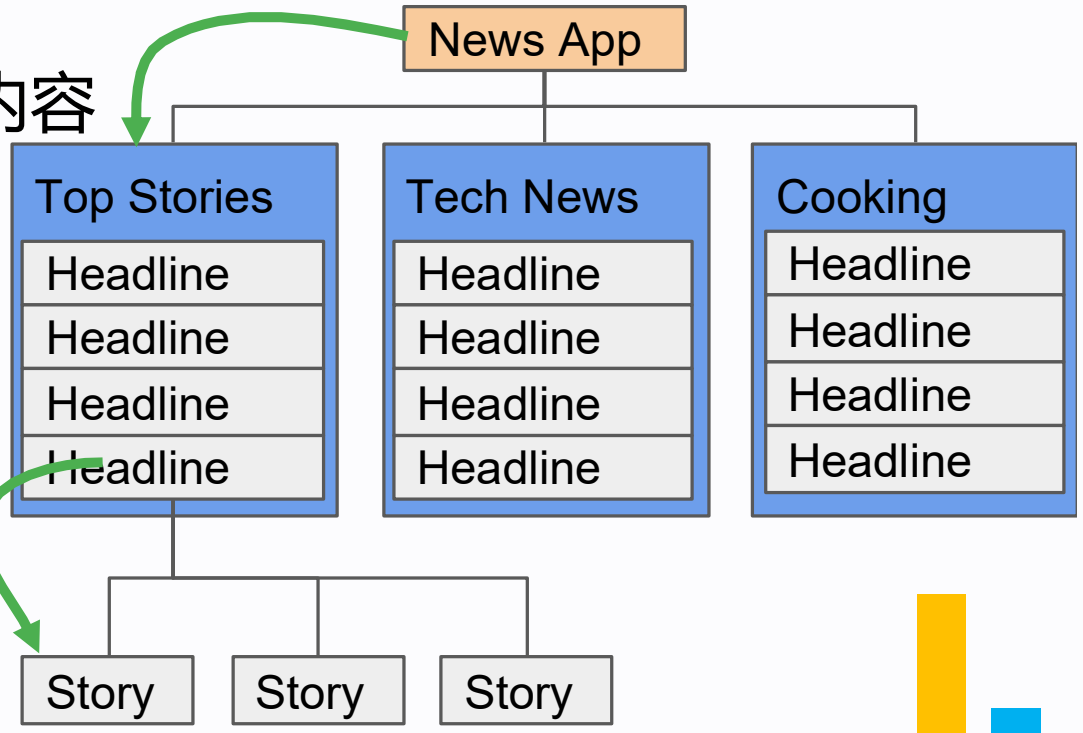

1
News App
2
Top Stories
Headline
Headline
Headline
Headline
Tech News
Headline
Headline
Headline
Headline
Cooking
Headline
Headline
Headline
Headline
3
Story
Story
Story

分层导航的类型

- Descendant navigation
  - 从父屏幕导航到子屏幕
  - 从新闻列表 > 新闻梗概 > 新闻内容
- Ancestral navigation
  - 从子屏幕/兄弟屏幕导航到父屏幕
  - 从新闻梗概导航到新闻列表
- Lateral navigation
  - 从一个兄弟屏幕到另一同级屏幕
  - 在选项卡式视图（tabbed view）间滑动

向下导航Descendant navigation

- Descendant navigation
  - 从父屏幕导航到子屏幕
  - 从主屏幕导航到列表到内容

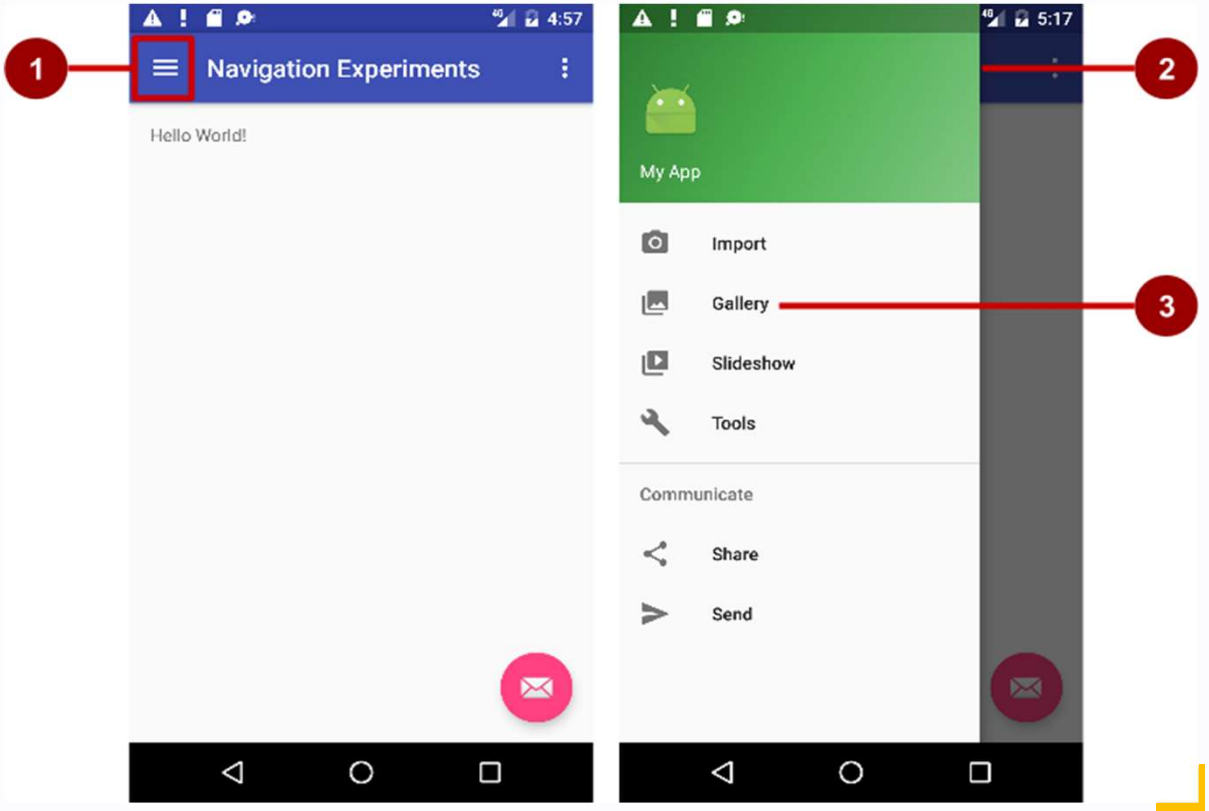


向下导航的控制

- 侧滑导航栏（navigation drawer）
- 主屏幕上按钮，图片按钮
- 其他可点击的视图（文本，图标等；水平，垂直或按照网格排列）
- 列表

侧滑导航栏navigation drawer

- 向下导航
  - 1. 应用栏图标
  - 2. 标头header
  - 3. 菜单选项

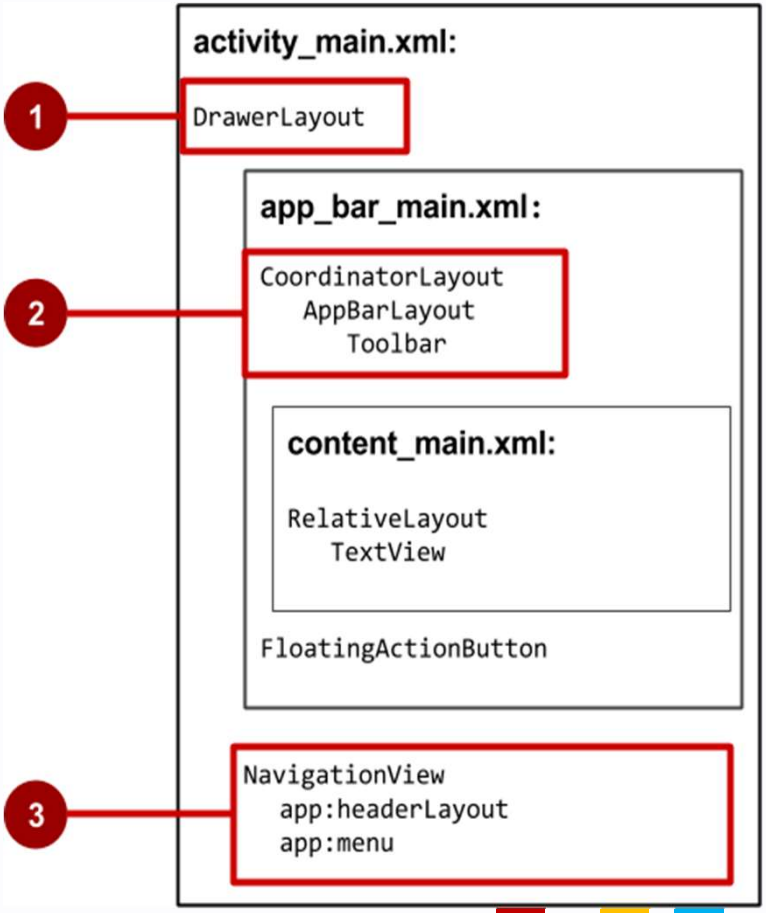


侧滑导航栏布局

- 创建布局
  - 作为活动根布局的侧滑导航栏
  - 导航视图
  - 应用栏，包括导航图标
  - 用来显示导航栏的内容布局
  - 导航栏标头的布局

侧滑导航栏的Activity布局

- 1. 根布局：DrawerLayout
- 2. 包含着带ToolBar的应用栏的CoordinatorLayout
- 3. 展示应用内容的布局
- 4. 包括header和选项的导航视图
  - NavigationView



实现侧滑导航栏的步骤

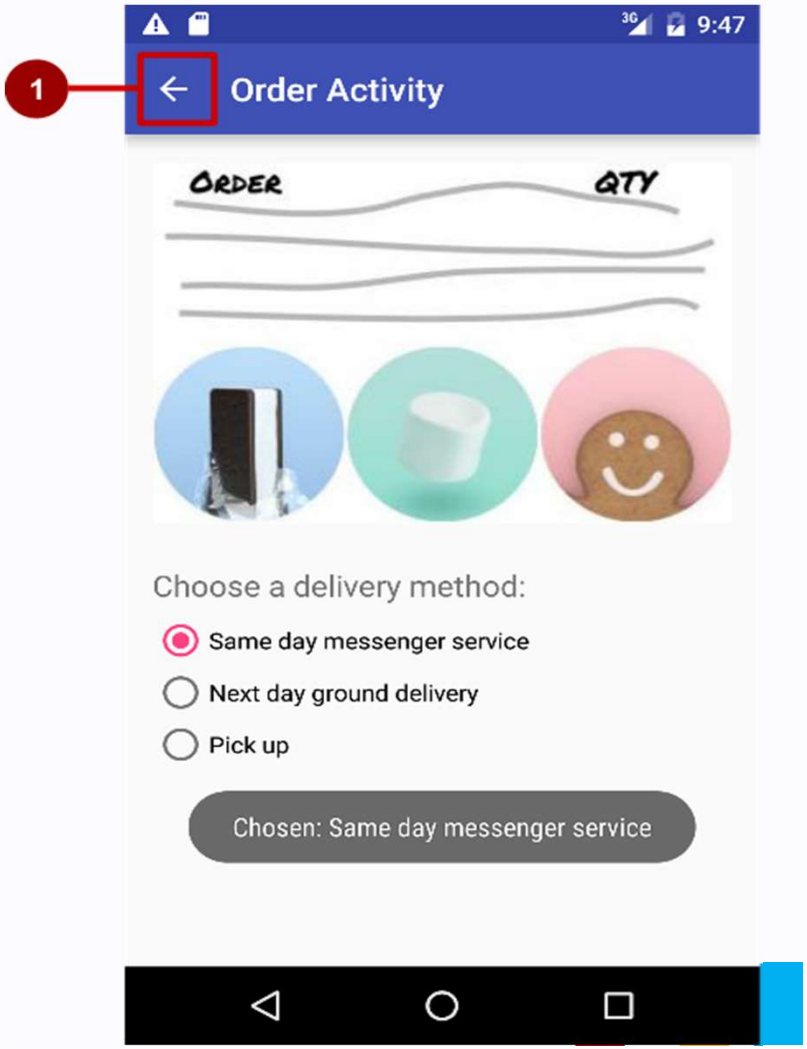
- 1. 使用项目标题和图标填充侧滑导航栏
- 2. 在Activity代码中设置侧滑导航栏和事件监听
- 3. 处理导航栏中的选项点击事件

其他向下导航模式

- 垂直列表，例如RecyclerView
- 垂直表格，例如GridView
- 带轮播的横向导航
- 多级菜单，如选项菜单
- ……

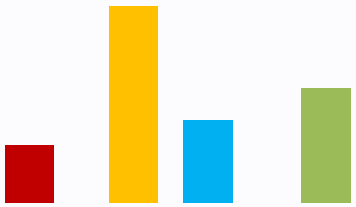
向上导航（向上按钮up button）

- 使用户可以返回上一级的屏幕



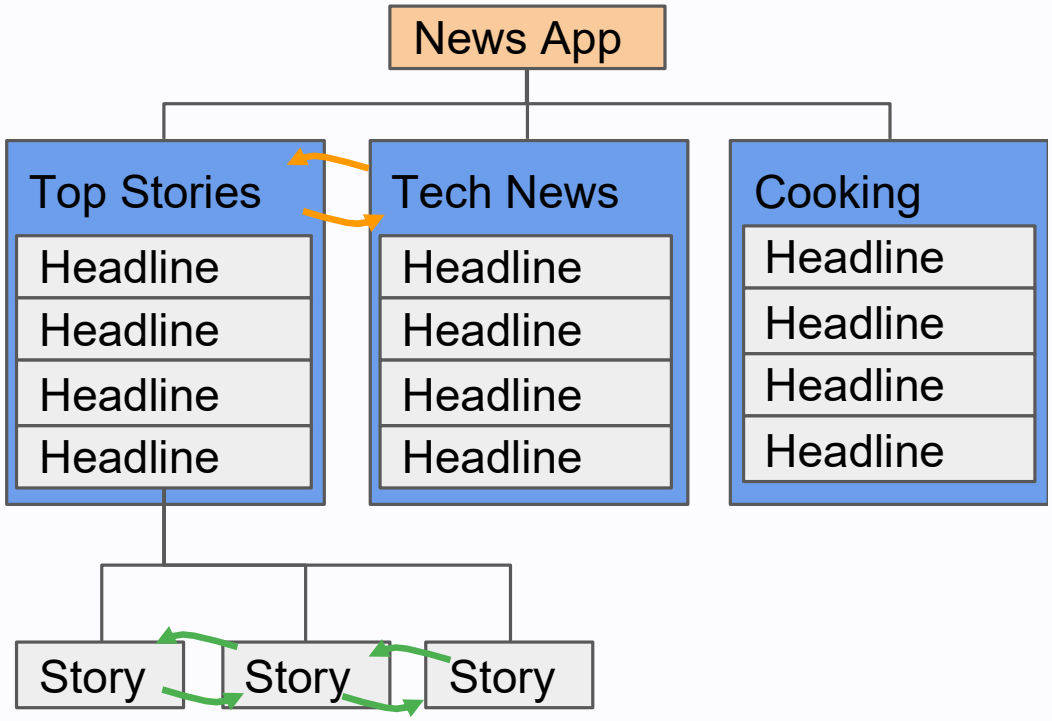
声明子活动的父项-AndroidManifest

```
<activity android:name=".OrderActivity"
  android:label="@string/title_activity_order"
  android:parentActivityName="com.example.android.

optionsmenuorderactivity.MainActivity">
  <meta-data
     android:name="android.support.PARENT_ACTIVITY"
     android:value=".MainActivity"/>
</activity>
```



标签（Tabs）和滑动（Swipes）

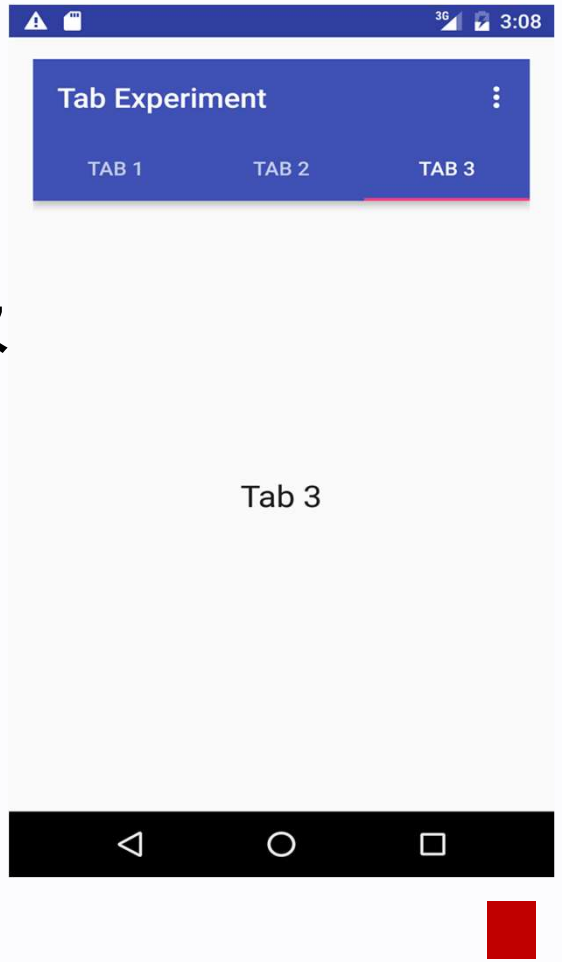
- 横向导航
  - 同级导航之间
  - 从一个列表到另一个列表
  - 在一个列表中从一个内容到另一个内容



使用Tabs和Swipes的优势

- 单击选择的标签-用户

  无需进一步导航即可访问内容
- 在相关屏幕之间导航而不访问父级



标签的实现方式

- 水平布局
- 沿着屏幕顶部运行
- 跨相关屏幕一致
- 切换不作为历史记录

实现标签的步骤

- 创建布局TabLayout
- 为每个标签创建Fragment和对应布局
- 实现PagerAdapter，继承FragmentPagerAdapter 或者FragmentStatePagerAdapter
- 创建选项卡布局（Tablayout）的实例
- 使用PagerAdapter管理
- 设置监听来确定被点击的Tab

在工具栏下方添加标签布局

```
<android.support.design.widget.TabLayout
    android:id="@+id/tab_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/toolbar"
    android:background="?attr/colorPrimary"
    android:minHeight="?attr/actionBarSize"

android:theme="@style/ThemeOverlay.AppCompat.Dark.Ac
tionBar"/>
```

在TabLayout下添加View Pager

```
<android.support.v4.view.ViewPager
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:layout_below="@id/tab_layout" />
```

在onCreate()中创建Tab layout

```
TabLayout tabLayout =
findViewById(R.id.tab_layout);
tabLayout.addTab(tabLayout.newTab().setText("Ta
b 1"));
tabLayout.addTab(tabLayout.newTab().setText("Ta
b 2"));
tabLayout.addTab(tabLayout.newTab().setText("Ta
b 3"));
tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);
```

在onCreate()中添加view pager

```
final ViewPager viewPager = findViewById(R.id.pager);
final PagerAdapter adapter = new PagerAdapter (
    getSupportFragmentManager(), tabLayout.getTabCount());
viewPager.setAdapter(adapter);
```

在onCreate()中添加监听

```
viewPager.addOnPageChangeListener(
            new
TabLayout.TabLayoutOnPageChangeListener(tabLayout));
tabLayout.addOnTabSelectedListener(
            new TabLayout.OnTabSelectedListener() {
   @Override
   public void onTabSelected(TabLayout.Tab tab) {
       viewPager.setCurrentItem(tab.getPosition());}
   @Override
   public void onTabUnselected(TabLayout.Tab tab) {}
   @Override
   public void onTabReselected(TabLayout.Tab tab) {} });
```

了解更多

- Navigation Design guide
  d.android.com/design/patterns/navigation.html
- Designing effective navigation
  d.android.com/training/design-navigation/index.html
- Creating a Navigation Drawer
  d.android.com/training/implementing-navigation/nav-drawer.html
- Creating swipe views with tabs
  d.android.com/training/implementing-navigation/lateral.html

4.5 循环视图RecyclerView

- 列表数据的分页
- 对于数据量特别大的列表，不要一次加载到视图中，速度慢且没有必要。

如何动态加载列表数据

- 回忆一下web数据中的分页
- 只获取部分数据，并显示这一部分数据，当要显示更多的数据的时候，再去请求更多的数据，并将数据进行显示。

下拉刷新的处理

- 触发下拉事件
- 执行相应操作（如网络获取数据）
- 显示相关信息

下拉自动加载列表

- 判断下拉位置
- 如果没到列表最下端，则正常显示列表中保存内容
- 如果已经拉到列表最小段，执行相应操作（如网络获取更多的数据），显示相应的信息

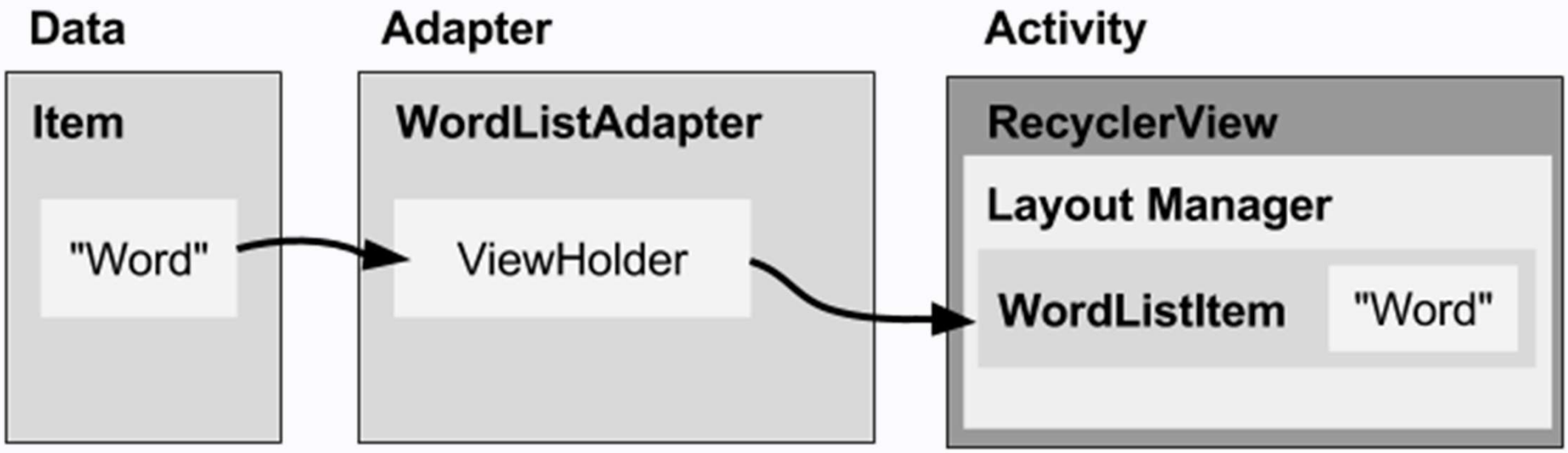
什么是循环视图RecyclerView

- RecyclerView是用于大型数据集的可滚动容器
- 高效
  - 重复使用数量有限的视图元素
  - 可以快速更新变化数据

概览

- **数据**
- **RecyclerView**滚动列表
- **布局** 每个项目item的数据布局——XML
- **布局管理** 管理视图中的UI组件—Recyclerview.LayoutManager
- **适配器Adapter** 连接视图和数据——RecyclerView.Adapter
- 视图容器ViewHolder 如何显示一个item的信息——RecyclerView.ViewHolder

组件概览

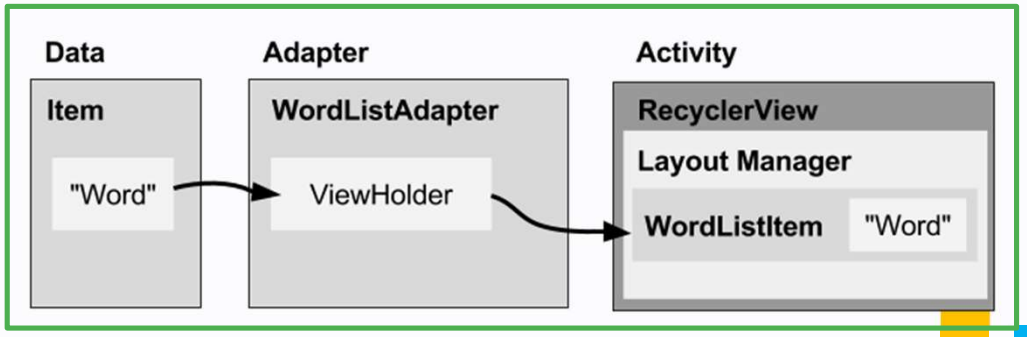

Data
Item
"Word"
Adapter
WordListAdapter
ViewHolder
Activity
RecyclerView
Layout Manager
WordListItem
"Word"

什么是布局管理器layout manager

- 每个视图有一个布局管理器
- 在RecyclerView中放置子视图
- 内置的布局管理器
  - LinearLayoutManager
  - GridLayoutManager
  - StaggeredGridLayoutManager
- 扩展RecyclerView.LayoutManager

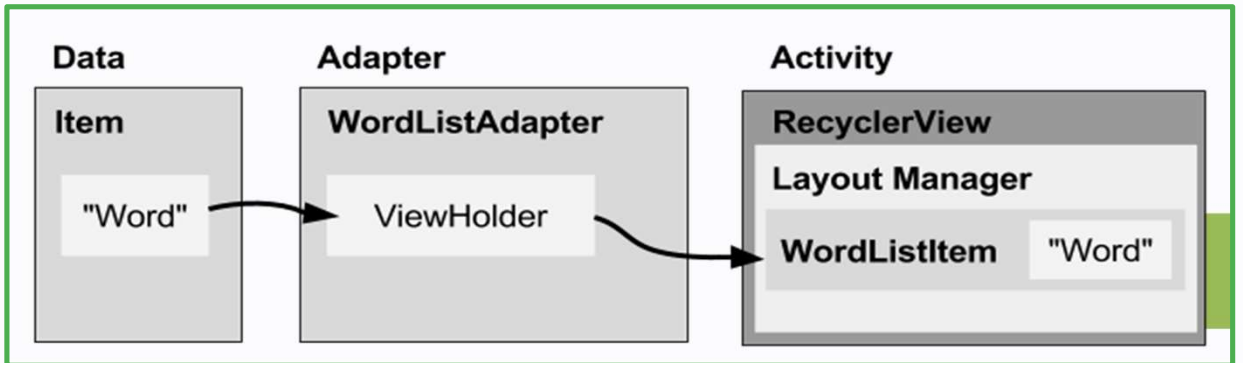
什么是适配器Adapter

- 协调不兼容的接口
  - 例如：从数据库Cursor中取数据，将其变成字符串
- 数据与视图之间的中介
- 当数据变化时创建，更新，添加，删除视图
- RecyclerView.Adapter



什么是视图容器Viewholder

- 被Adapter用来为list item准备视图
- xml文件中的布局
- 可以有可点击的元素
- 被layout manager放置
- `RecyclerView.ViewHolder`

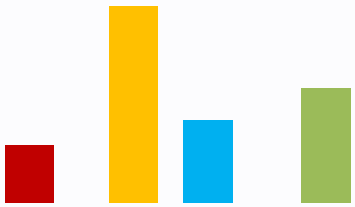


步骤总结

- （如果需要）添加RecyclerView依赖到build.gradle
- 在布局中添加RecyclerView
- 创建item的布局文件
- Extend RecyclerView.Adapter
- Extend RecyclerView.ViewHolder
- 在Activity的onCreate()中，创建带adapter和layout manager的RecyclerView

添加依赖到app/build.gradle

```
dependencies {
   ...
   compile 'com.android.support:recyclerview-
v7:26.1.0'
   ...
   }
```

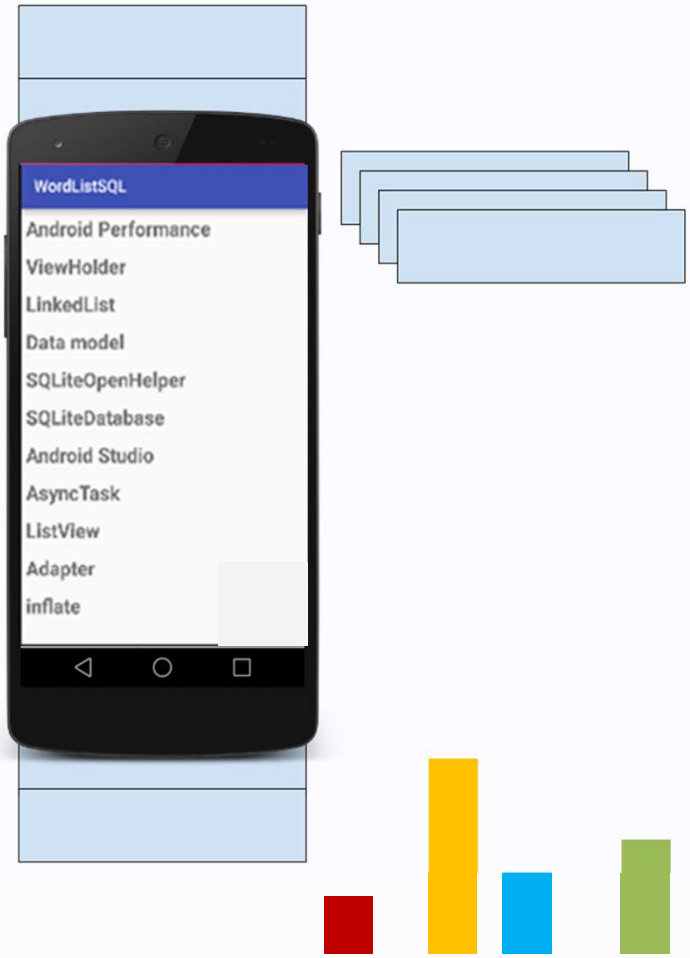


在布局文件中添加RecyclerView

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</android.support.v7.widget.RecyclerView>
```

创建item的布局文件

```
<LinearLayout …>
   <TextView
      android:id="@+id/word"
      style="@style/word_title" />
</LinearLayout>
```

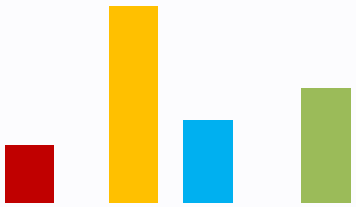


实现适配器Adapter

```
public class WordListAdapter extends
RecyclerView.Adapter<WordListAdapter.WordViewHolder> {
    public WordListAdapter(Context context,
                           LinkedList<String> wordList)
{
        mInflater = LayoutInflater.from(context);
        this.mWordList = wordList;
    }
}
```

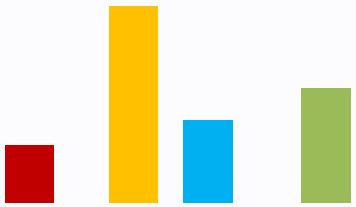
适配器需要实现三个函数

- onCreateViewHolder()
- inBindViewHolder()
- getItemCount()



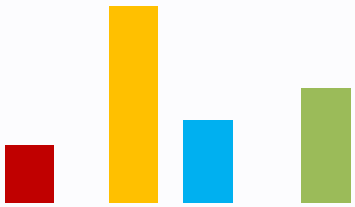
onCreateViewHolder()

```
@Override
public WordViewHolder onCreateViewHolder(
        ViewGroup parent, int viewType) {
        // Create view from layout
        View mItemView = mInflater.inflate(
             R.layout.wordlist_item, parent, false);
        return new WordViewHolder(mItemView, this);
}
```



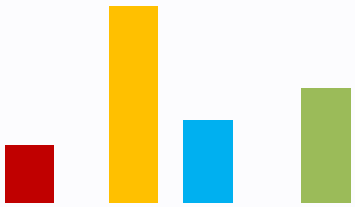
onBindViewHolder()

```
@Override
public void onBindViewHolder(
        WordViewHolder holder, int position) {
        // Retrieve the data for that position
        String mCurrent = mWordList.get(position);
        // Add the data to the view
        holder.wordItemView.setText(mCurrent);
}
```



getItemCount()

```
@Override
public int getItemCount() {
   // Return the number of data items to display
   return mWordList.size();
}
```

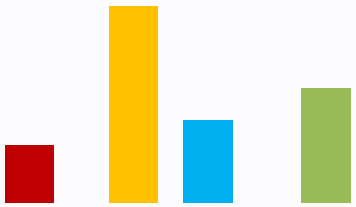


在适配器类中创建view holder

- `class WordViewHolder extends RecyclerView.ViewHolder { //.. }`

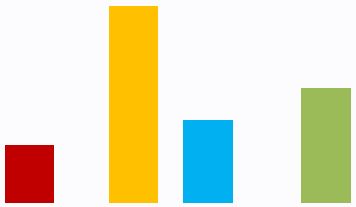
- 如果想要处理点击

```
class WordViewHolder extends RecyclerView.ViewHolder
                    implements View.OnClickListener
{ //.. }
```



View holder结构

```
public WordViewHolder(View itemView, WordListAdapter
adapter) {
  super(itemView);
  // Get the layout
  wordItemView = itemView.findViewById(R.id.word);
  // Associate with this adapter
  this.mAdapter = adapter;
  // Add click listener, if desired
  itemView.setOnClickListener(this);
}
// Implement onClick() if desired
```



在Activity的onCreate()中创建RecyclerView

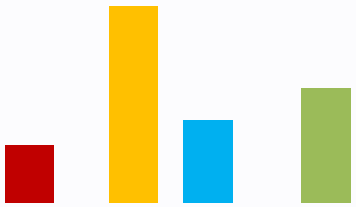
```
mRecyclerView = findViewById(R.id.recyclerview);
mAdapter = new WordListAdapter(this, mWordList);
mRecyclerView.setAdapter(mAdapter);
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

了解更多

- RecyclerView
- RecyclerView class
- RecyclerView.Adapter class
- RecyclerView.ViewHolder class
- RecyclerView.LayoutManager class

课堂练习

- 完善MyFragmentPagerAdapter
- 在Fragment1中配置recyclerview




Thank
You
ACK：Android
官方材料