浙江大学
ZHEJIANG UNIVERSITY

# Activity创建与使用

陶煜波

# 应用组件与清单文件 回顾



- Android Studio功能：创建、运行、调试

- 四类应用组件：活动 (Activity)，服务，内容提供程序，广播接收器
  - Activity - UI，服务 - 后台应用，内容提供程序 - 数据管理，广播接收器 - 消息
  - Activity、服务和广播接收器通过Intent的异步消息进行启动

  - Activity = Layout (xml，布局，静态) + Activity子类 (Java/Kotlin，功能，动态)
  - 活动App = 一个或多个Activity + 资源文件 + 清单文件
  - Intent：同一App中不同Activity，不同App的Activity、服务和广播接收器之间

- 清单文件 (manifest)
  - 组件申明，Activity的Intent过滤器
  - 用户权限，硬件与软件功能
  - 最低API级别，外部API库

# 视图、布局和资源 回顾

- 创建视图的三种方式
  - 方法一：Android Studio Layout Editor (类似xcode的可视化编辑)
  - 方法二：通过编写XML代码
  - 方法三：通过Java/Kotlin代码创建

同步更新

运行时Activity创建时

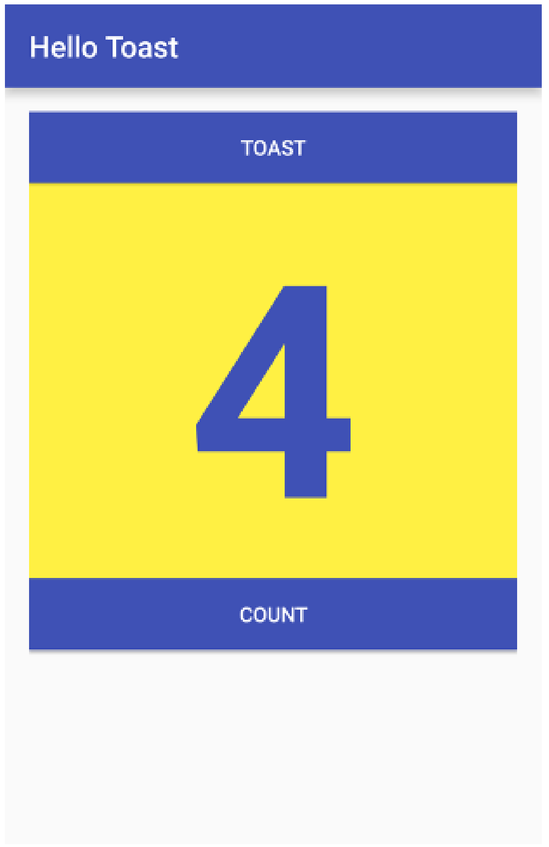
- 视图组与视图的层次性
  - LinearLayout、ConstraintLayout、FrameLayout
  - ScrollView、RecyclerView
  - Intent：各类视图的常用属性：textSize、padding、margin、gravity、layout_gravity等

- 资源
  - res资源(读写)：XML“@资源类型/资源名称”，代码R.资源类型.资源名称
  - assets资源(只读)

# 课程目录

# Activity是什么?

- Activity是一种应用组件
- 表示一个窗口
- 通常填充整个屏幕，但也可以嵌入另一个Activity
- 是一个Java/Kotlin类，通常一个Activity就是一个Java/Kotlin文件

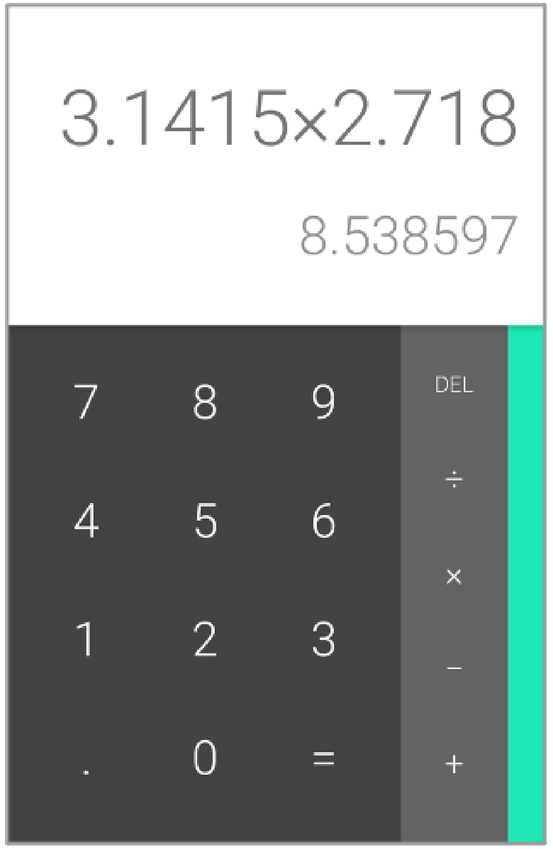


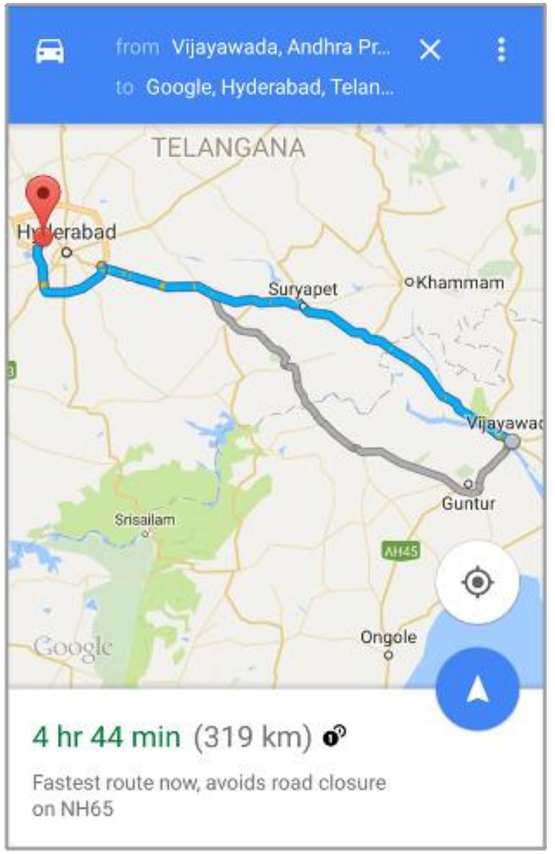


Activity实例

# Activity是什么?

- 表示一种活动，比如购物，发微博，发送email，或者获取当前方向
- 处理交互，比如按钮点击，文字输入等
- 可以通过Intent启动同一App或者不同App中的其他Activity
- 存在生命周期—创建、启动、运行、暂停、恢复、停止和销毁



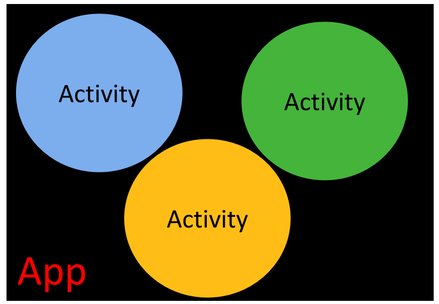




Activity实例

# Activity与App， Layout

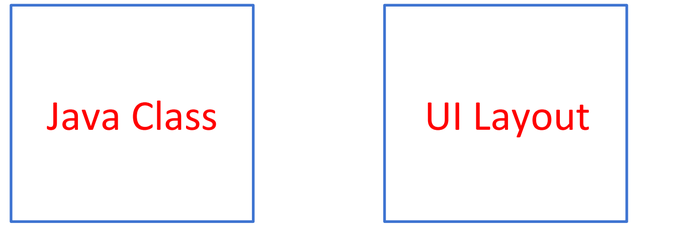
## Activity和App

- App中的Activity之间是一种松散的连接关系
- 用户看到的第一个Activity称为“Main Activity”
- Activity可以在清单manifest文件中制定父子关系，以方便App导航




## Activity和Layout

- 一个Activity经常有一个UI Layout
- Layout 被定义在一个或者多个XML文件中
- Activity在创建时，会根据Layout的XML文件生成Java Class文件



# 实现一个新的 Activity

1. 在Layout下定义Layout XML文件

2. 在Java下定义Activity类
   -继承AppCompatActivity

3. 将Activity和Layout连接起来
   -在onCreate()方法中通过setContentView设置内容视图

4. 在安卓清单文件中声明Activity

# 1. 在Layout下定义Layout XML文件

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Let's Shop for Food!" />
</LinearLayout>
```

# 2. 在 Java class 中定义 Activity

ZHEJIANG UNIVERSITY

MainActivity.java

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

# 3. 将 Activity 和 Layout 连接起来

通过R.资源类型.资源名称进行访问的

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

# 4.在manifest文件中声明 Activity

```
<activity android:name=".MainActivity">

</activity>
```

# Intent的作用

- 启动一个Activity
  - 按钮点击启动一个输入文本的Activity
    - 同一App不同Activity之间的导航
  - 点击“分享”打开一个允许用户上传照片App的Activity
    - 不同App之间的Activity启动
- 启动一项服务
  - 在后台初始化下载一个文件
- 发送广播
  - 系统通知系统中所有App设备正在充电

# 显式Intent和隐式Intent

浙江大学 ZHEJIANG UNIVERSITY

## 显式Intent

- 启动一个指定的Activity
  - Main Activity启动一个查看购物车的Activity

## 隐式Intent

- 让Android系统找到一个可以处理该请求的Activity
  - 点击分享打开一个选择界面，其中包含可选的一系列App

# 使用Intent 启动一个Activity

浙江大学 ZHEJIANG UNIVERSITY

## 使用显式Intent启动一个Activity

为了启动一个指定的Activity，使用一个显式的Intent

1. 创建一个Intent

```
Intent intent = new Intent(this, ActivityName.class);
```

2. 使用Intent来启动Activity

```
startActivity(intent);
```

## 使用隐式Intent启动一个Activity

如果需要让安卓系统找到一个可以处理请求的Activity，使用一个隐式的Intent

1. 创建一个Intent

```
Intent intent = new Intent(action, uri);
```

2. 使用Intent来启动Activity

```
startActivity(intent);
```

# 隐式Intent例子

## 显示一个网页

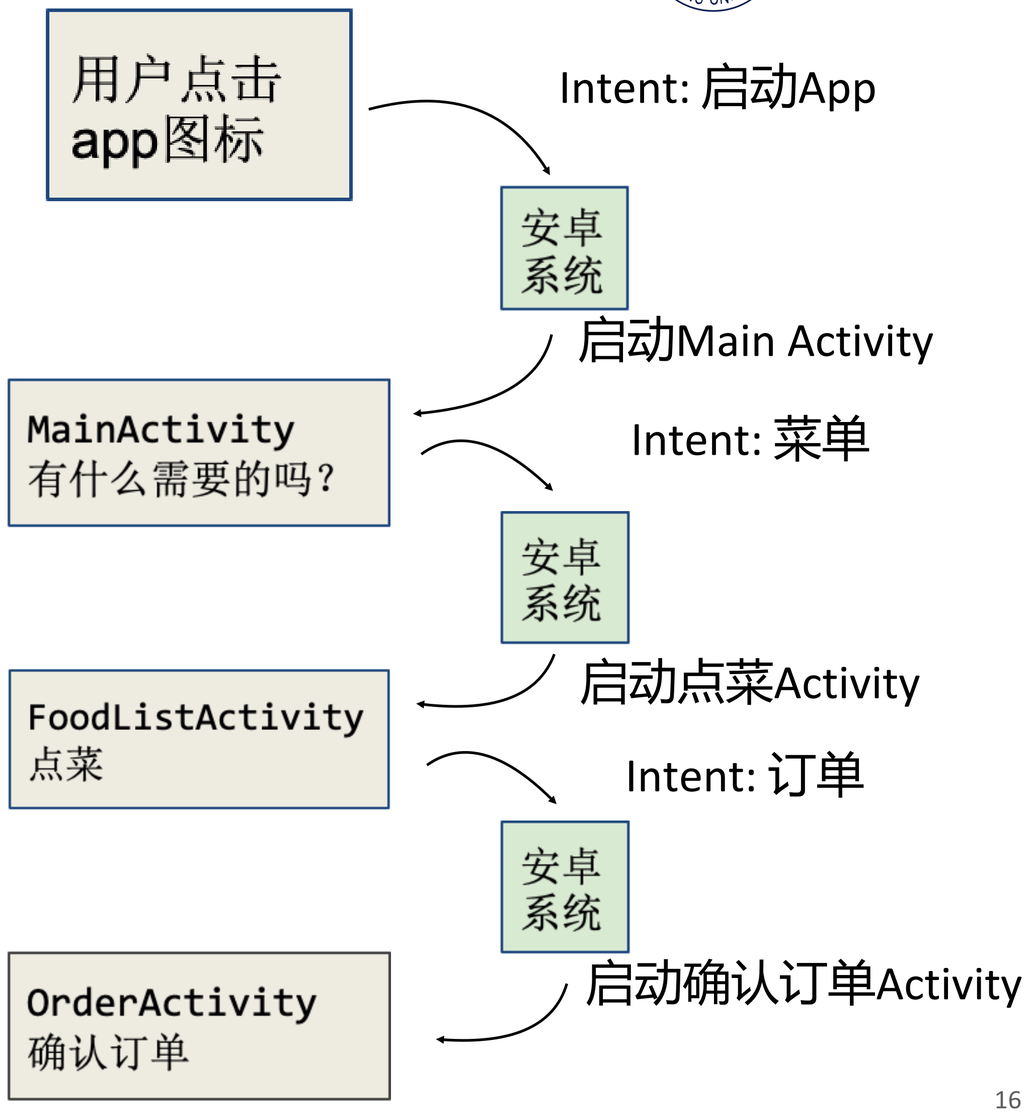
```
Uri uri = Uri.parse("https://www.zju.edu.cn/");
Intent it = new Intent(Intent.ACTION_VIEW, uri);
startActivity(it);
```

## 打电话

```
Uri uri = Uri.parse("tel:8005551234");
Intent it = new Intent(Intent.ACTION_DIAL, uri);
startActivity(it);
```

# Activity怎么运行?

- 所有的Activity实例是由安卓运行时(Android Runtime)进行管理
- 由Intent启动



# Intent中两种类型的发送数据

- Data — 可以用URI表示，数据的地址。只能有一个值
- Extras — 一个Bundle，包含了一个或多个键值对(key-values pairs)

# 数据的发送与接收

## 发送数据的 Activity

1. 创建Intent对象
2. 把data或extras放入Intent中
3. 调用startActivity()启动一个新的Activity

## 接收数据的 Activity

1. 拿到Intent对象
2. 从Intent对象中提取data或者extras

# 在Intent的data中放入URI

```
// A web page URL
intent.setData(Uri.parse("http://www.google.com"));

// A sample file URI
intent.setData(Uri.fromFile(new File("/sdcard/sample.jpg")));
```

# 在 Intent 的extras中放入数据

- putExtra(String name, int value)
  ⇒intent.putExtra("level", 406);

- putExtra(String name, String[] value)
  ⇒String[] foodList = {"Rice", "Beans", "Fruit"};
  intent.putExtra("food", foodList);

- putExtras(bundle);
  ⇒如果数据很多，创建一个bundle

```
public static final String EXTRA_MESSAGE_KEY =
"com.example.android.twoactivities.extra.MESSAGE";

Intent intent = new Intent(this, SecondActivity.class);

String message = "Hello Activity!";
intent.putExtra(EXTRA_MESSAGE_KEY, message);
startActivity(intent);
```

用extras 给Activity传递数据

# 从 Intent 当中获取数据

- getData();

```
Uri locationUri = intent.getData();
```

- int getIntExtra(String name, int defaultValue)

```
int level = intent.getIntExtra("level", 0);
```

- Bundle bundle = intent.getExtras();

⇒ 从bundle当中获取所有数据

# 把数据返回给原先的 Activity

1. 使用ActivityResultLauncher.launch()来启动接收数据Activity

2. 从第二个Activity 当中返回数据
   - 创建一个新的Intent
   - 使用putExtra()方法将回应的数据放在 Intent 中
   - 将结果设置为Activity.RESULT_OK
   - 如果用户取消了，设置为Activity.RESULT_CANCELED
   - 调用finish()来关闭 Activity
3. 在第一个Activity中通过 registerForActivityResult() 注册结果回调

# registerForActivityResult()

registerForActivityResult(contract, callback)

- 在第一个Activity中注册结果回调，并生成一个ActivityResultLauncher
- 通过 Intent extras 来传递数据
- 之后使用 ActivityResultLauncher.launch(intent) 启动Activity
- 结束之后回到先前的Activity，执行结果回调函数来处理返回的数据

# 1. registerForActivityResult()

```
private final ActivityResultLauncher<Intent> textActivityLauncher =
registerForActivityResult(
new ActivityResultContracts.StartActivityForResult(),
this::handleTextActivityResult);

Intent intent = new Intent(this, TextActivity.class);

textActivityLauncher.launch(intent);
```

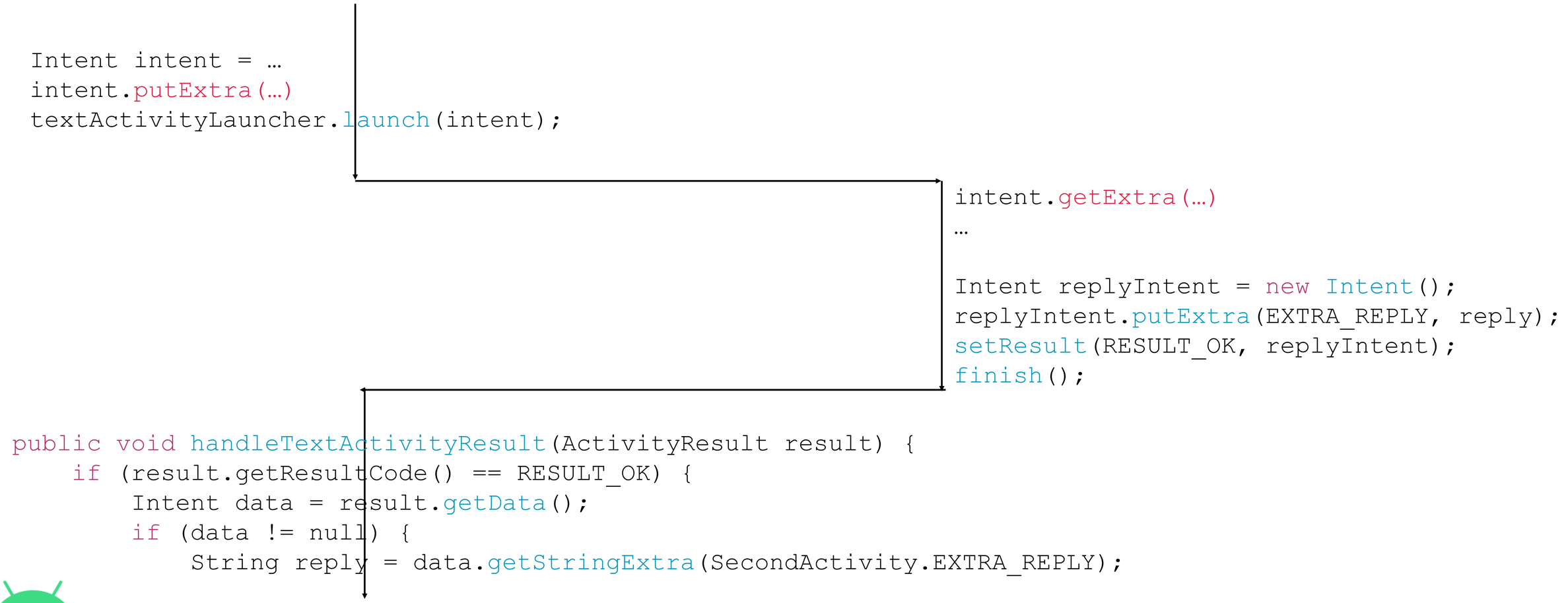
# 2. 从第二个Activity当中返回数据

```
// 创建一个 intent
Intent replyIntent = new Intent();
// 把 data 放入 extra
replyIntent.putExtra(EXTRA_REPLY, reply);
// 把 activity 的 result 设置为 RESULT_OK
setResult(RESULT_OK, replyIntent);
// 结束当前 activity
finish();
```

# 3. 实现结果回调函数

```
public void handleTextActivityResult(ActivityResult result) {
  if (result.getResultCode() == RESULT_OK) {
    Intent data = result.getData();
    if (data != null) {
        String reply = data.getStringExtra(SecondActivity.EXTRA_REPLY);
        // … do something with the data
    }
  }
}
```

# Intent数据传输

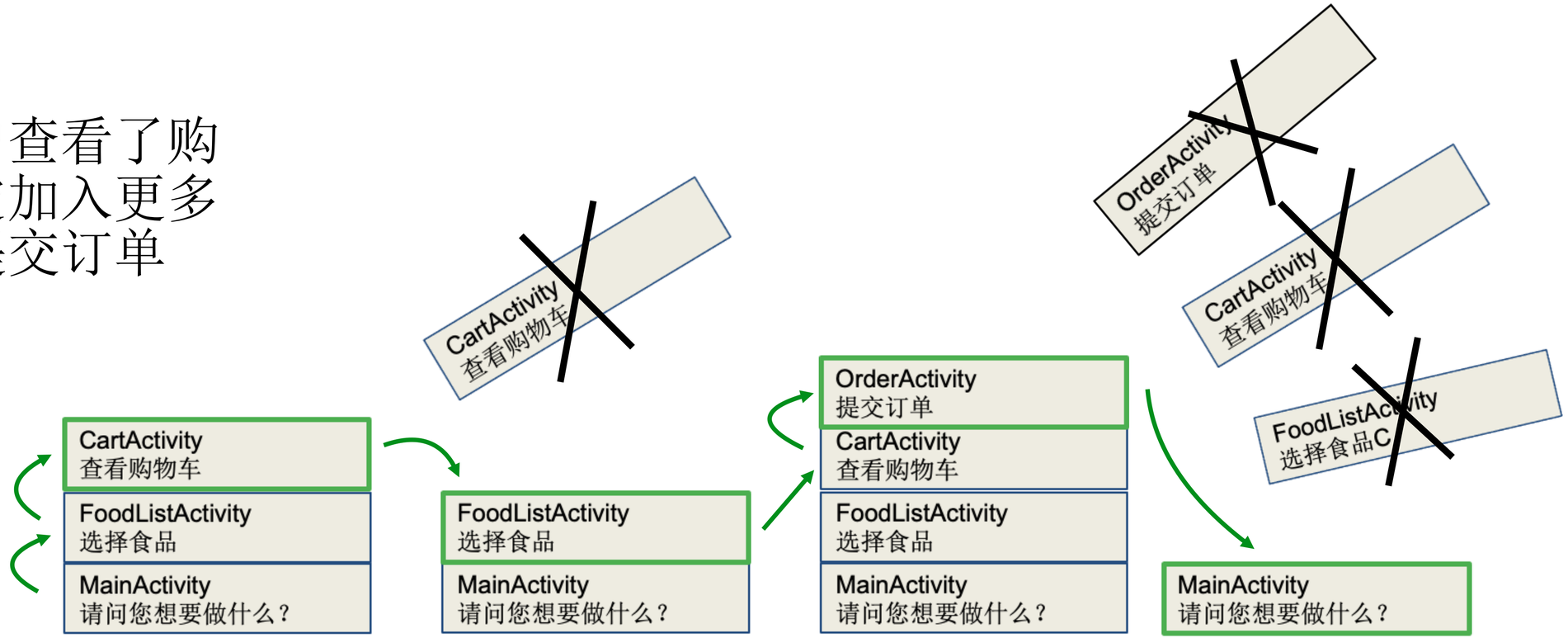

FirstActivity
SecondActivity
Intent intent = …
intent.putExtra(…)
textActivityLauncher.launch(intent);
intent.getExtra(…)
…
Intent replyIntent = new Intent();
replyIntent.putExtra(EXTRA_REPLY, reply);
setResult(RESULT_OK, replyIntent);
finish();
public void handleTextActivityResult(ActivityResult result) {
if (result.getResultCode() == RESULT_OK) {
Intent data = result.getData();
if (data != null) {
String reply = data.getStringExtra(SecondActivity.EXTRA_REPLY);

# Activity栈

- 当Activity启动的时候，前一个Activity停止了，并被放入Activity栈当中
- 栈的特点：后进先出。当现在的Activity停止的时候，或者用户点击了回退按钮，这个Activity从栈中弹出，系统将恢复前一个Activity

# Activity栈

- 场景描述：用户查看了购物车之后，决定加入更多的物品，然后提交订单



# 两种类型的导航

## 向后导航 

- 设备的返回按钮
- 由Android系统的Back栈控制

## 向上导航 

- 在App的动作栏
- 由清单文件中定义的Activity的父子关系控制

# 向后导航

- Back栈存储了最近浏览过的视图
- Back栈包含当前任务的所有Activity实例，存储的顺序和用户启动这些实例的顺序相反(栈的先进后出特性)
- 每一个任务都有它的Back栈
- 切换任务的同时，会激活切换到的任务的Back栈

# 向上导航

- 返回当前Activity的父Activity
- 在 manifest 中定义Activity的父子关系
- 设置 parentActivityName

```
<activity
    android:name=".ShowDinnerActivity"
    android:parentActivityName=".MainActivity" >
</activity>
```

# 小结

- 一个Activity对应一个Layout（UI界面）和一个Java类（事件响应）

- 创建Activity
  - 创建Layout，定义Java类，onCreate中关联Layout，清单文件申明

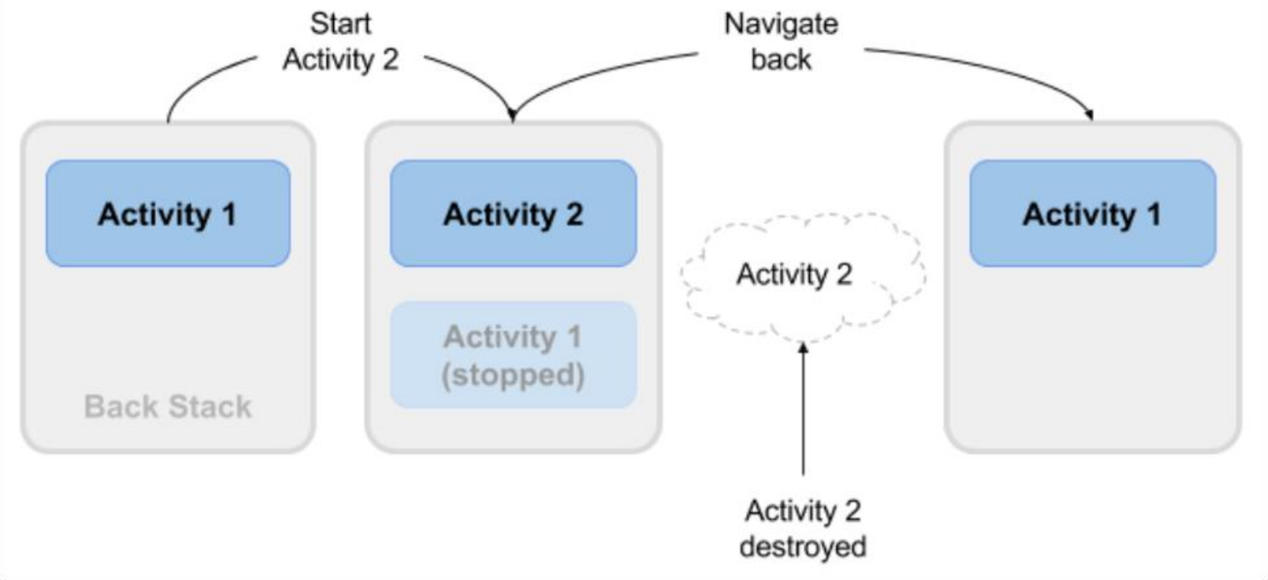
- Inten作用 - 启动Activity、服务、广播
  - 显示Intent 与 隐式Intent
  - 应用组件之间的数据传递setData与setExtra vs. getData与getExtra
  - 应用组件之间的数据返回registerForActivityResult、ActivityResultLauncher startActivityForResult与结果回调函数

- Activity栈
  - 向后导航 - 用户浏览顺序
  - 向上导航 - App清单文件父子关系

# 课程目录

# 什么是活动生命周期

- 活动生命周期是一个活动(Activity)从创建到销毁期间经历的所有状态集
- 更正式的说，活动生命周期是包含Activity所有状态的有向图，以及从一个状态转换到下一个状态时的回调(有向边的动作)

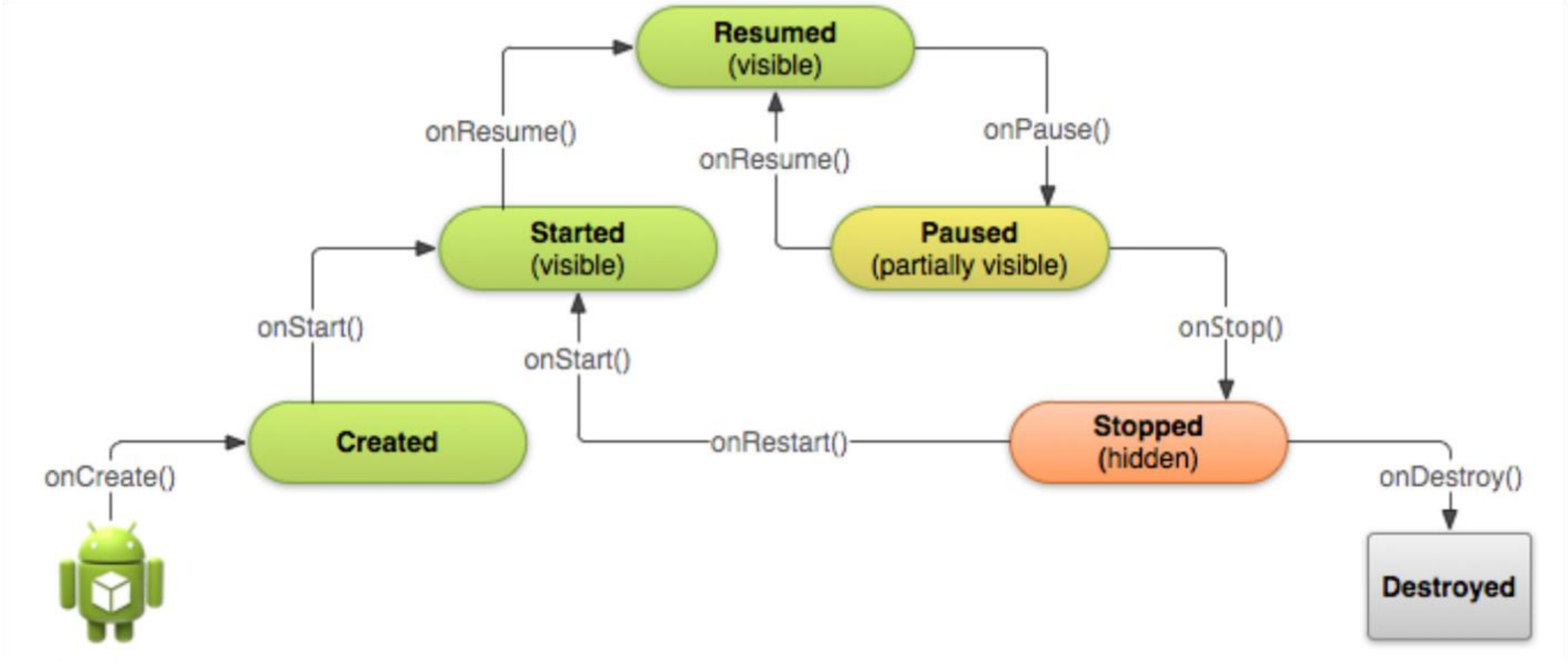


# 活动生命周期包含的状态集

- 状态集
  - Created 创建
  - Started 启动
  - Resume 恢复
  - Paused 暂停
  - Stopped 停止
  - Destroyed 销毁

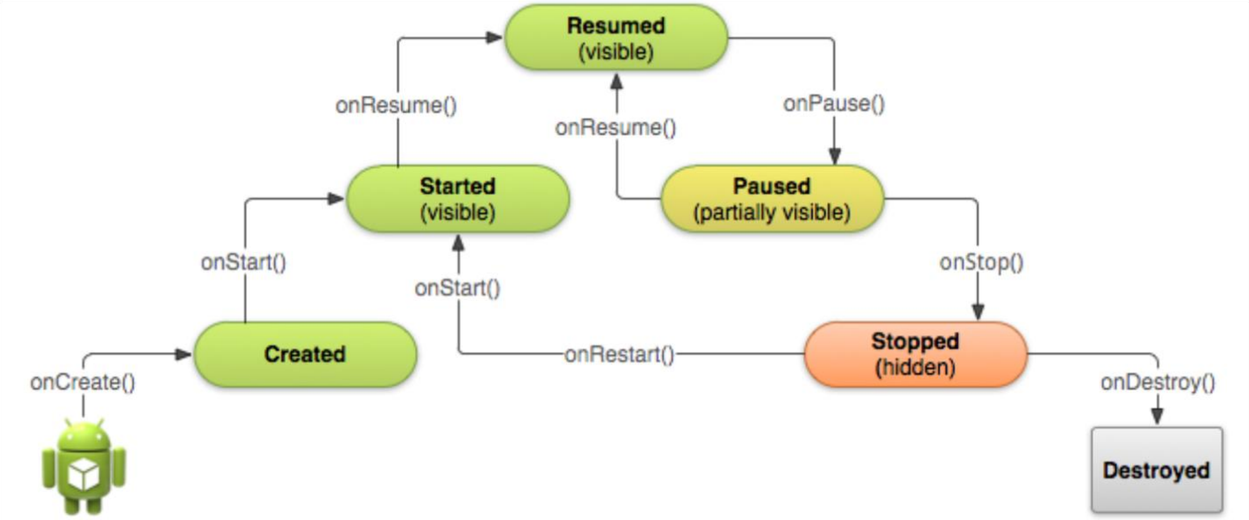
- Activity状态的改变一般来说是因为用户操作，譬如旋转设备，或是因为系统原因(系统调度等)

# 活动状态和回调图


Resumed
(visible)
onResume()
onResume()
onPause()
Started
(visible)
Paused
(partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed

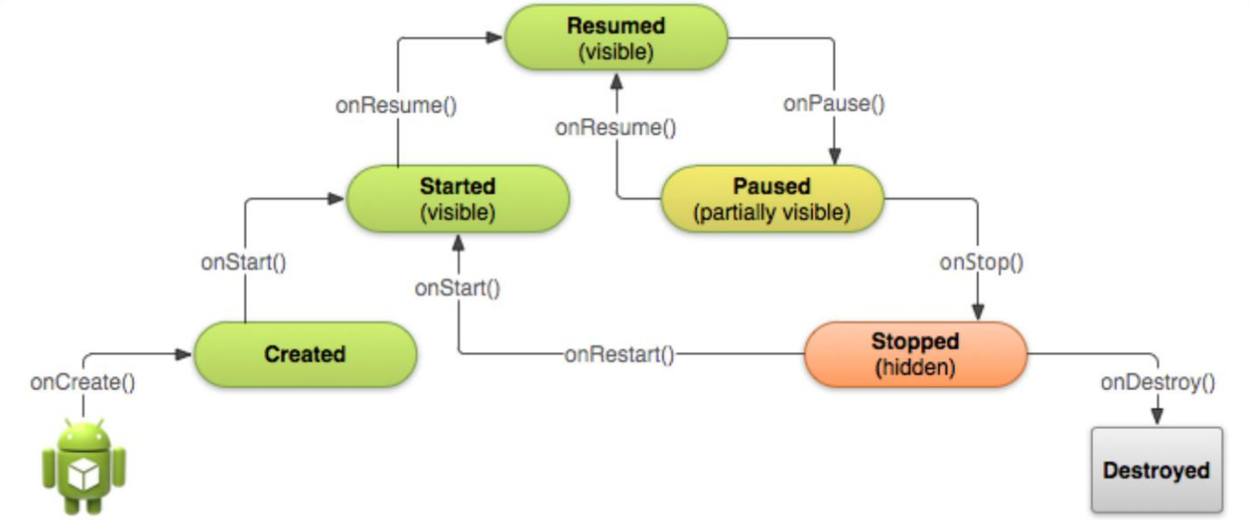
# 回调的一般情景

- 一个Activity启动后出现在手机屏幕上，之后再由使用者按下返回键结束，一段时间后因各种原因被系统销毁。在这个情景中回调执行顺序是：



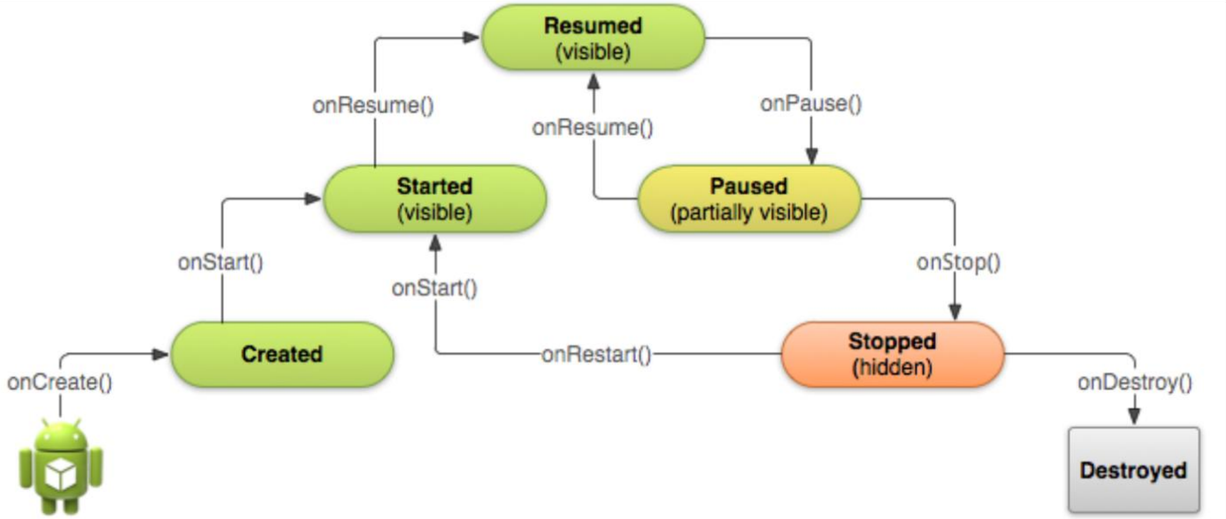
# 回调的一般情景

- 当Activity准备要产生时，先调用onCreate方法
- Activity产生后(还未出现在手机屏幕上)，调用onStart方法
- 当Activity出现在手机屏幕后，调用onResume方法
- 当使用者按下返回键结束Activity时，先调用onPause方法
- 当Activity从屏幕上消失时，调用onStop方法
- Activity被系统销毁之前，调用onDestroy方法



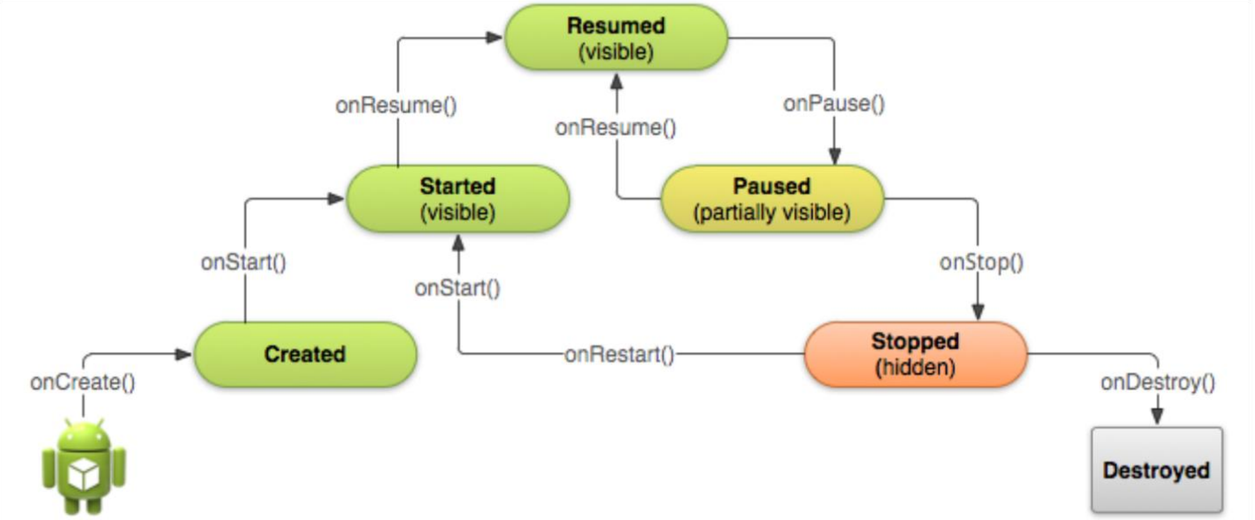
# 暂停情景

- 当Activity已经显示在手机屏幕上了，但这个Activity如果有对话框出现在Activity的前面，此时Activity是无法交互使用的，称之为在暂停状态下。在这个情景中回调执行顺序是：
  - 当出现对话框，Activity部分可见，但无法交互使用时，调用onPause方法
  - 对话框消失，调用onResume方法后，Activity才完全可见



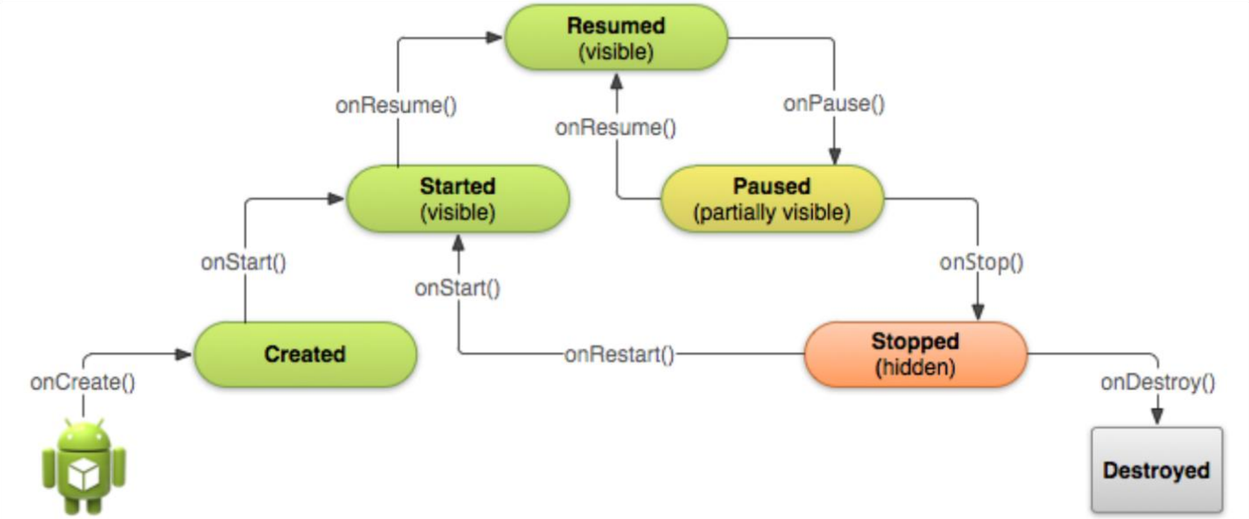
# 切换Activity情景

- 当Activity在手机屏幕時，用户开启最近使用的App清单，并点击了另一个App时，在前景的Activity会停止并进入背景，直到使用者再在App清单中点击这个Activity，这个Activity才会被重新执行。在这个情景中回调执行顺序是：



# 切换Activity情景

- 用户点击另一个App执行，让原本在前景的Activity进入背景前，会先调用onPause方法，再调用onStop方法，此时Activity完全进入背景，不在手机画面上
- 当Activity由用户从最近使用App清单中点击后，先调用onRestart方法
- 之后再调用onStart方法
- Activity显示在手机画面后，再调用onResume方法



# 覆盖（Override）回调函数



- 每个新的Activity都需要开发者覆盖onCreate方法
- 剩下的回调方法当开发者需要时覆盖

# onCreate() –> Created

- 首次创建时调用，例如当用户点击启动器图标时
- 完成所有静态的创建工作：创建视图，将数据绑定到列表，…
- 在Activity的生命周期中只调用一次，以后恢复Activity时使用之前已冻结的状态，而不再调用onCreate()
- 始终跟随onStart()调用

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // The activity is being created.
}
```

onCreate() 示例

# onStart() –> Started

- 当Activity即将出现在屏幕时调用
- 在生命周期中可以多次被调用

```
@Override
protected void onStart() {
    super.onStart();
    // The activity is about to become visible.
}
```

onStart() 示例

# onRestart() –> Started

- Activity处于停止状态，但即将再次启动。在它启动之前onRestart()被调用
- onRestart()设计上是一个非常短暂的状态
- 始终紧跟onStart()

```
@Override
protected void onRestart() {
    super.onRestart();
    // The activity is between stopped and started.
}
```

onRestart() 示例

# onResume() –> Resumed/Running

- 当Activity已经出现在屏幕并即将开始与用户交互时调用
- 此时意味着Activity已移至Activity堆栈的顶部
- 开始接受用户输入
- 代表Activity正处于运行状态
- 始终跟随onPause()

```
@Override
protected void onResume() {
    super.onResume();
    // The activity has become visible
    // it is now "resumed"
}
```

onResume() 示例

# onPause() –> Paused

- 当系统打开另外一个Activity时调用
- 此时用户正在离开Activity，但Activity仍部分可见
- 这一状态通常用来将未保存的修改持久化以及停止动画等消耗资源的内容
- 覆盖这一回调的实现运行时间必须很短，因为在此方法返回之前，下一个Activity不会被打开

```
@Override
protected void onPause() {
    super.onPause();
    // Another activity is taking focus
    // this activity is about to be "paused"
}
```

onPause() 示例

# onStop() –> Stopped

- 当Activity不再对用户可见时调用
- 此时有可能是系统打开一个新的Activity，或是一个旧的Activity被带到前景，或是这个Activity正在被销毁
- onPause()短时间内未能处理完的逻辑可以放在这里实现

```
@Override
protected void onStop() {
    super.onStop();
    // The activity is no longer visible
    // it is now "stopped"
}
```

onStop() 示例

# onDestroy() –> Destroyed

- 这是Activity被销毁之前的最后一次调用
- 通常发生在用户通过导航返回上一个Activity，或是配置改变(设备旋转、用户开启了多窗口模式等)
- 有可能是该Activity主动结束自身或是系统决定销毁该Activity来节省内存
- 可以通过调用isFinishing()方法来检查是否正在销毁
- 系统可能会在不调用它的情况下销毁Activity，因此建议使用onPause()或onStop()来保存退出前的数据或状态

```
@Override
protected void onDestroy() {
    super.onDestroy();
    // The activity is about to be destroyed.
}
```

onDestroy() 示例

# 课程目录

# 什么是活动实例状态?

- 如果系统由于系统原因(例如配置更改或内存不够)而销毁Activity，系统会保留这个实例的状态
- 当用户尝试导航返回这个Activity时，系统将使用已保存的数据创建该Activity的新实例，这些数据描述了活动在销毁时的状态
- 系统保存的这部分用于恢复先前状态的数据称为实例状态

# 配置更改

- 当用户执行以下操作时，配置更改会使Activity中的当前布局或其他资源无效：
  - 旋转设备
  - 选择不同的系统语言，因此区域设置会发生变化
  - 进入多窗口模式 (从Android 7开始有这个功能)
- 当发生配置更改时，Android系统会
  1. 关闭Activity，通过调用：
     -onPause()
     -onStop()
     -onDestroy()
  2. 重新打开Activity，通过调用：
     - onCreate()
     - onStart()
     - onResume()

# 保存和恢复Activity状态：保存的内容

- 系统仅保存
  - 具有唯一ID（android：id）的视图状态，例如EditText中的文本内容
  - 启动这个Activity的Intent
- 开发者负责保存其他Activity和用户数据

# 保存和恢复Activity状态：保存实例状态的方法

- 在Activity中实现onSaveInstanceState()
  - 当Activity可能被销毁时，这个方法会由Android运行时调用
  - 在这个方法中，仅为当前会话期间下的该Activity实例保存数据，不要做多余的

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    // Add information for saving HelloToast counter
    // to the to the outState bundle
    outState.putString("count",
                       String.valueOf(mShowCount.getText()));
}
```

# 保存和恢复Activity状态：恢复实例状态

- 有两种方法取出已保存的实例状态

1. (推荐做法) 在onCreate(Bundle mySavedState)中取出，这样做的好处是能尽快恢复之前的样子

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mShowCount = findViewById(R.id.show_count);

    if (savedInstanceState != null) {
        String count = savedInstanceState.getString("count");
        if (mShowCount != null)
            mShowCount.setText(count);
    }
}
```

# 保存和恢复Activity状态：恢复实例状态

- 有两种方法取出已保存的实例状态

2. 通过实现onRestoreInstanceState(Bundle mySavedState)来完成(注意，该方法会在在onStart()之后调用)

```
@Override
public void onRestoreInstanceState(Bundle mySavedState) {
  super.onRestoreInstanceState(mySavedState);

  if (mySavedState != null) {
      String count = mySavedState.getString("count");
      if (count != null)
          mShowCount.setText(count);
  }
}
```

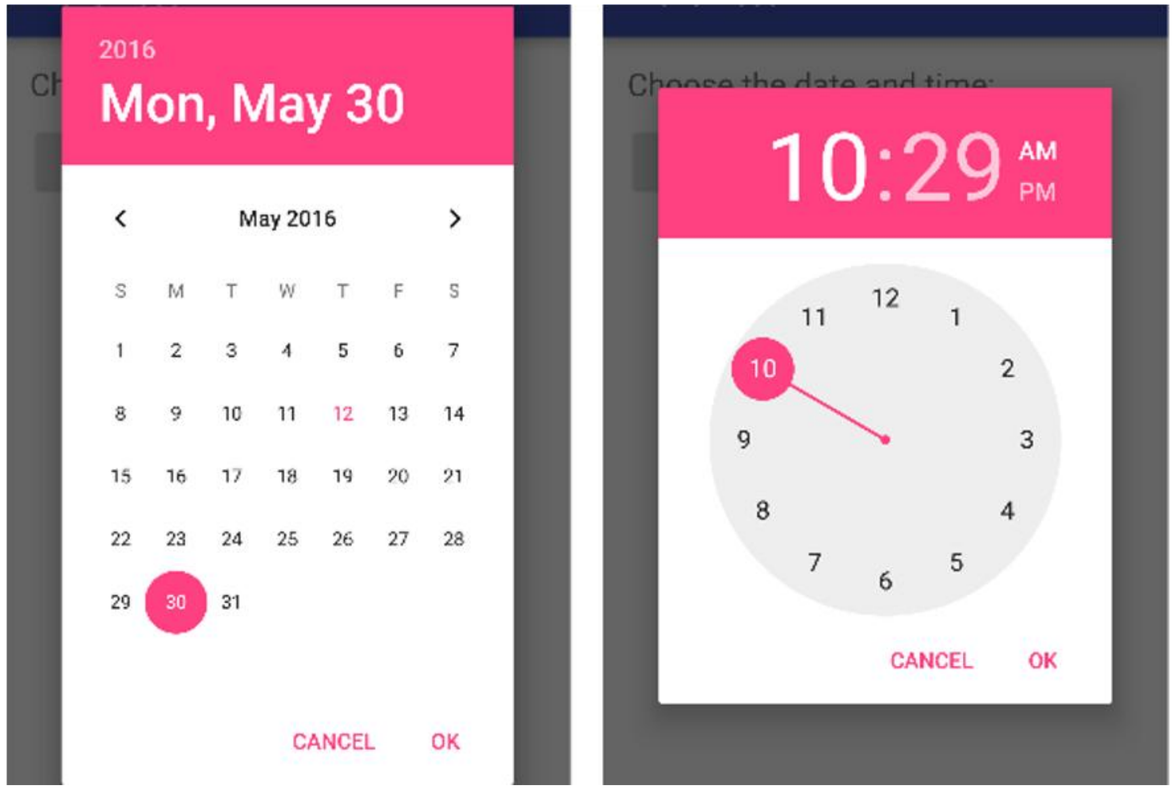
# 课程目录

# 片段（fragments）

- 片段就像Activity中的mini-Activity
- 管理自己的生命周期
- 接收自己的输入事件
- 可以在父Activity运行时添加或删除多个片段
- 可以组合在一个Activity中
- 可以在多个Activity中重用

# 选择器

- 日期选择器（左）
- 时间选择器（右）
- 选择器使用了片段(fragments)
  - 使用了DialogFragment来显示选择器
  - DialogFragment是一个置于顶层的窗口



# 创建一个日期选择器对话框

1. 添加一个空白片段，扩展DialogFragment并实现DatePickerDialog.OnDateSetListener
2. 在onCreateDialog()中初始化日期并返回对话框
3. 在onDateSet()中处理日期
4. 在Activity中显示选择器和添加方法以使用日期

# 1. 添加片段

```
public class DatePickerFragment extends DialogFragment
                                implements DatePickerDialog.OnDateSetListener {

}
```

# 2. 实现onCreateDialog()

```
public class DatePickerFragment extends DialogFragment
                                implements DatePickerDialog.OnDateSetListener {
    @NonNull
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker.
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);
        // Create a new instance of DatePickerDialog and return it.
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }
}
```

# 3. onDateSet()中处理日期

```
@Override
public void onDateSet(DatePicker datePicker, int year, int month, int day) {
    MainActivity activity = (MainActivity) getActivity();
    // call method in MainActivity, to pass selected date
    activity.processDatePickerResult(year, month, day);
}
```

# 4. 在MainActivity中显示选择器并使用选中的日期


```
public void showDatePicker(View view) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}

public void processDatePickerResult(int year, int month, int day) {
    // process date
}
```

# 创建一个时间选择器对话框

- 与日期对话框类似，只需要
  - 把实现DatePickerDialog.OnDateSetListener 变成实现TimePickerDialog.OnTimeSetListener
  - 把在onDateSet中处理日期变成在onTimeSet中处理时间

# 回顾：两种类型的导航

## 向后导航 

- 设备的返回按钮
- 由 Android 系统的 back 栈 控制

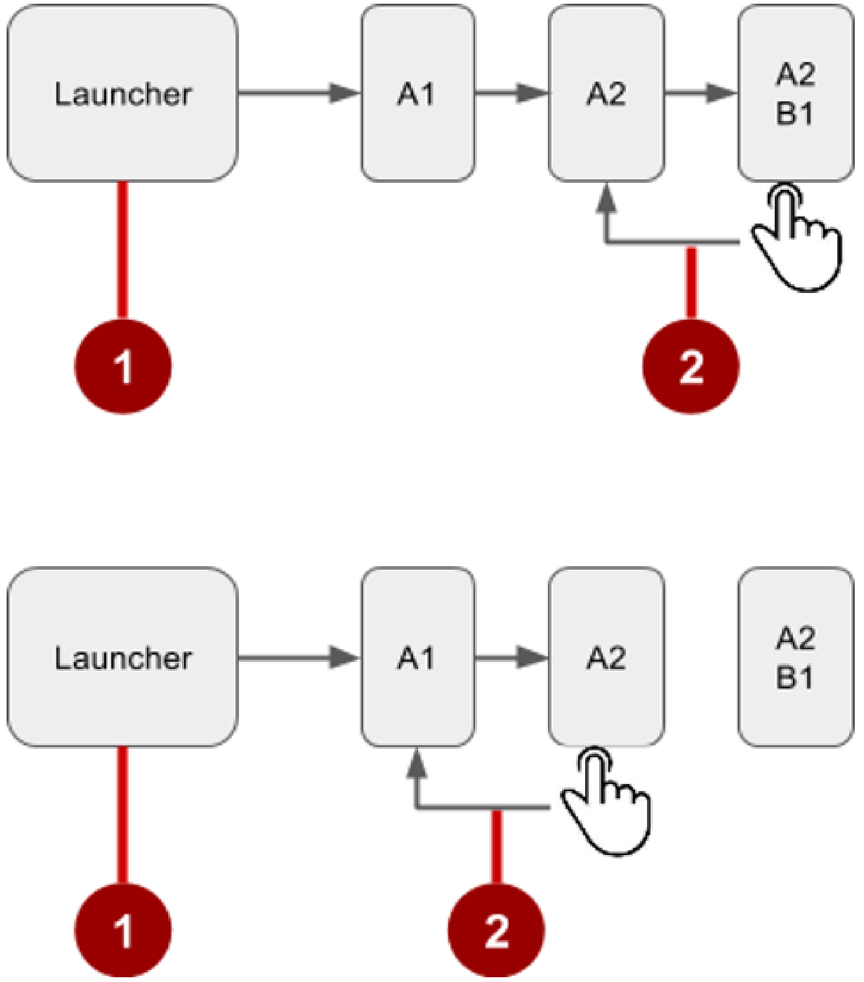
## 向上导航 

- 在 app 的动作栏
- 由 manifest 文件中规定的 Activity 的父子关系控制

# 历史记录的导航

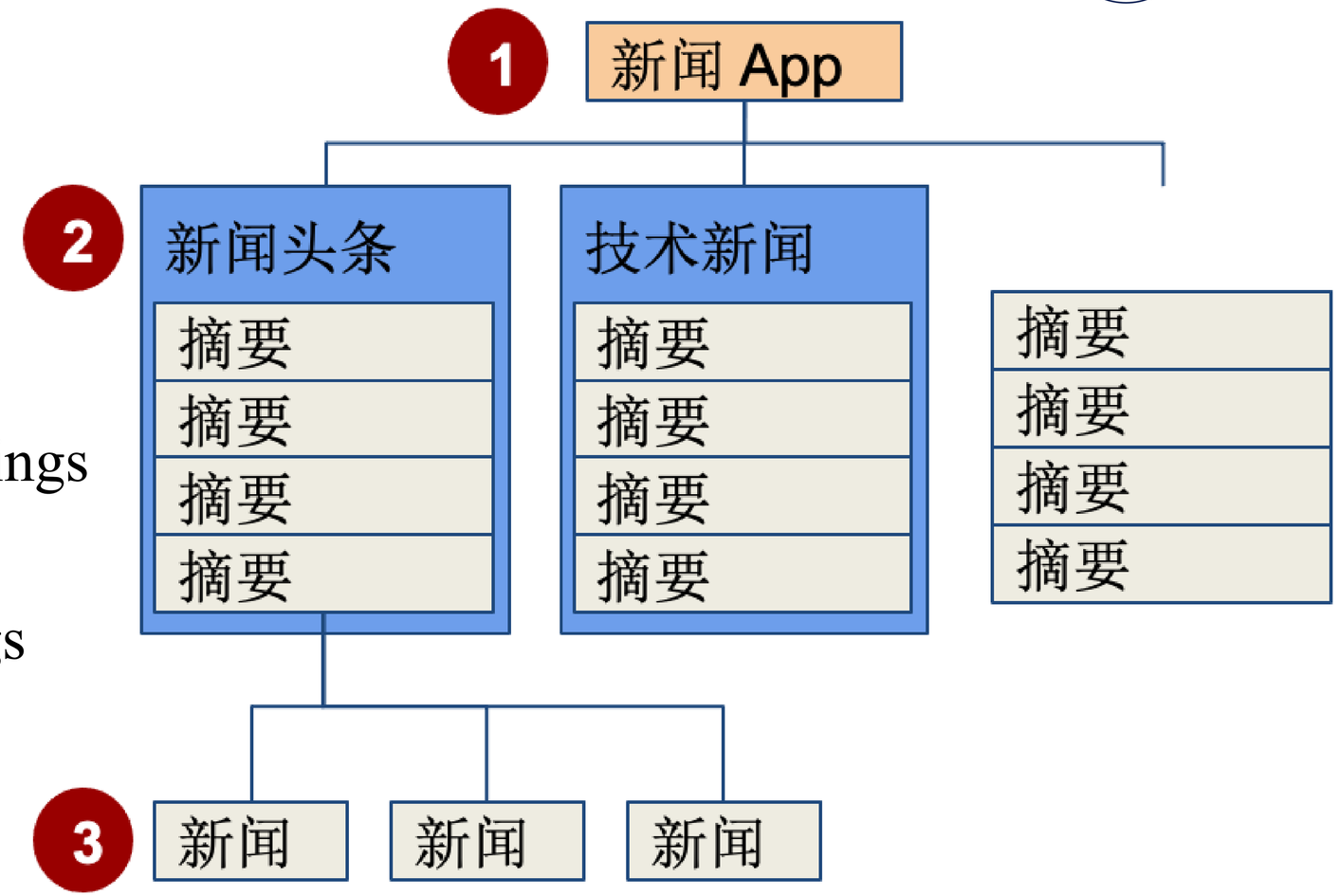
1. 从Launcher开始的导航

2. 用户点击向后导航 ◁ 来导航到先前的画面，返回的顺序与启动顺序相反



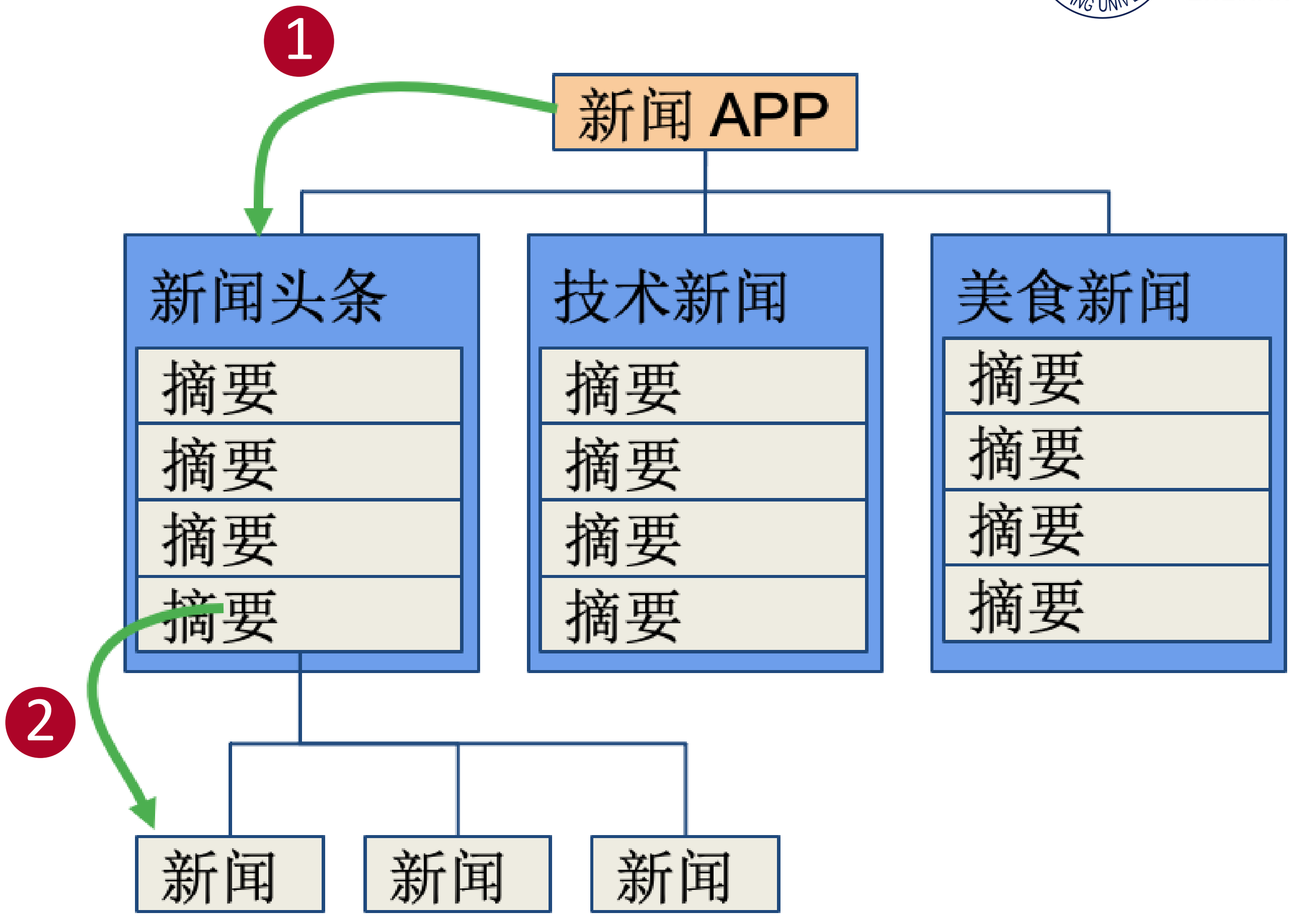
# 层次导航

1. 父窗口
2. 子窗口： collection siblings
3. 子窗口： section siblings

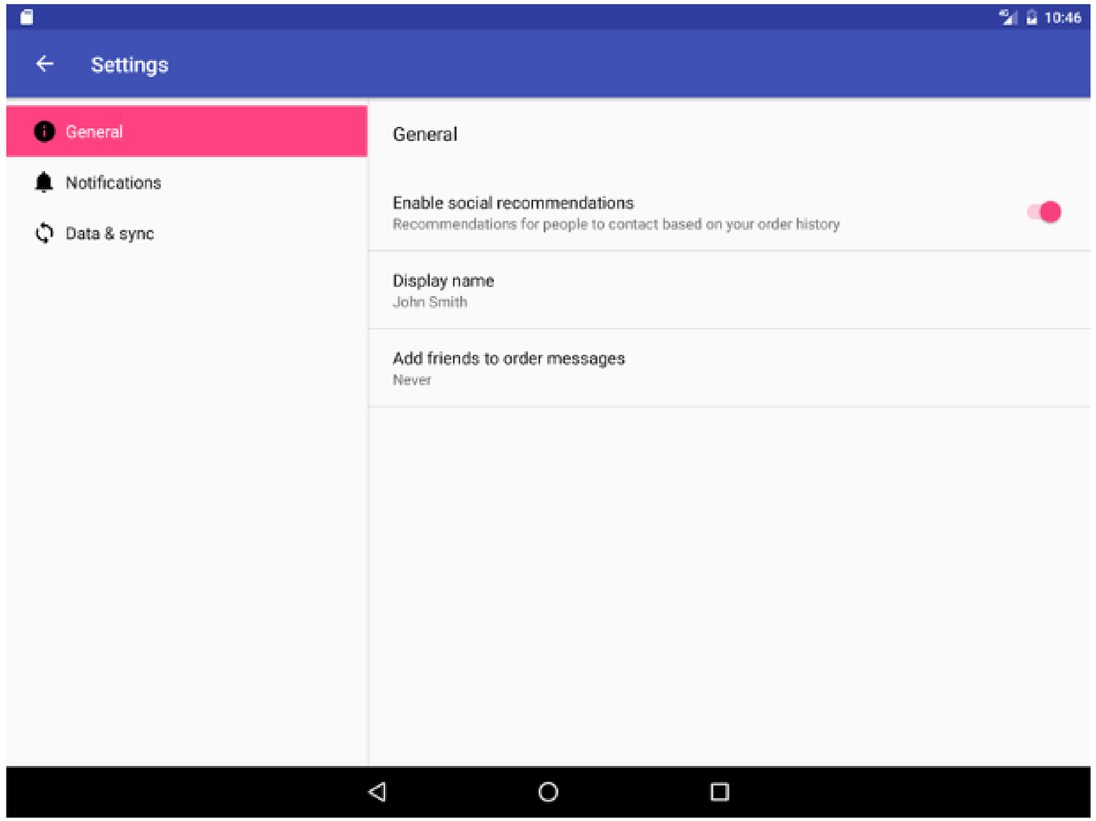


# 后代导航

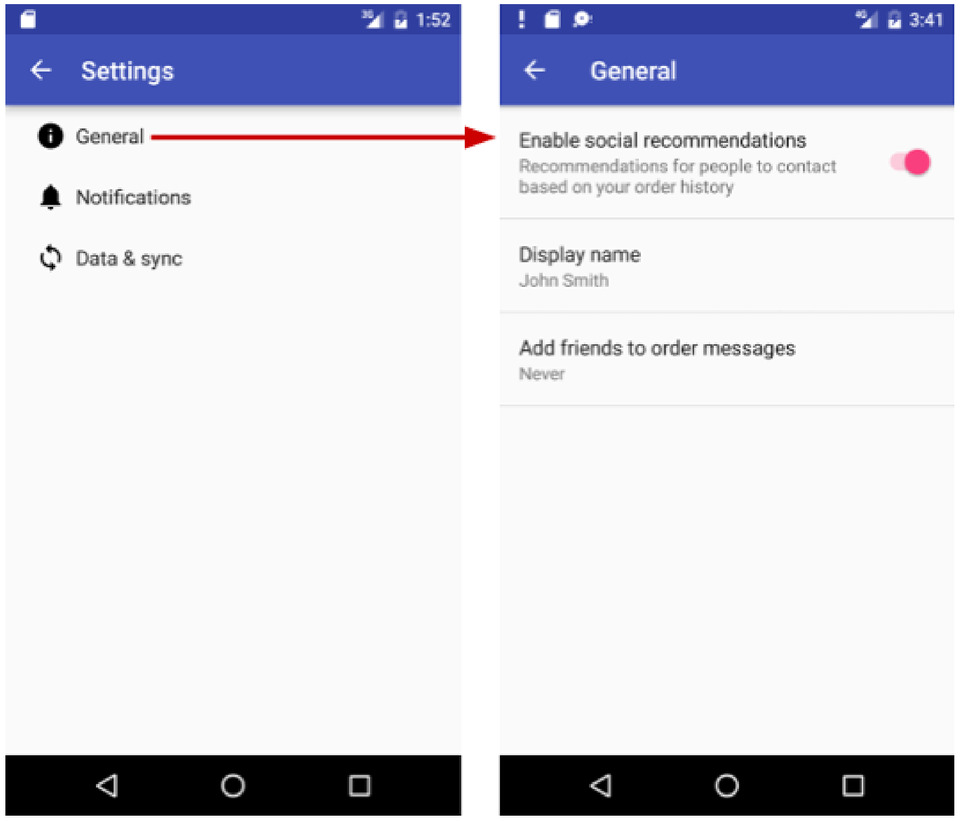
1. 从父窗口到一个子窗口
2. App 主界面 -> 一系列的新闻摘要 -> 一条新闻



# Master/Detail 工作流（后代导航）




平板上：并排出现

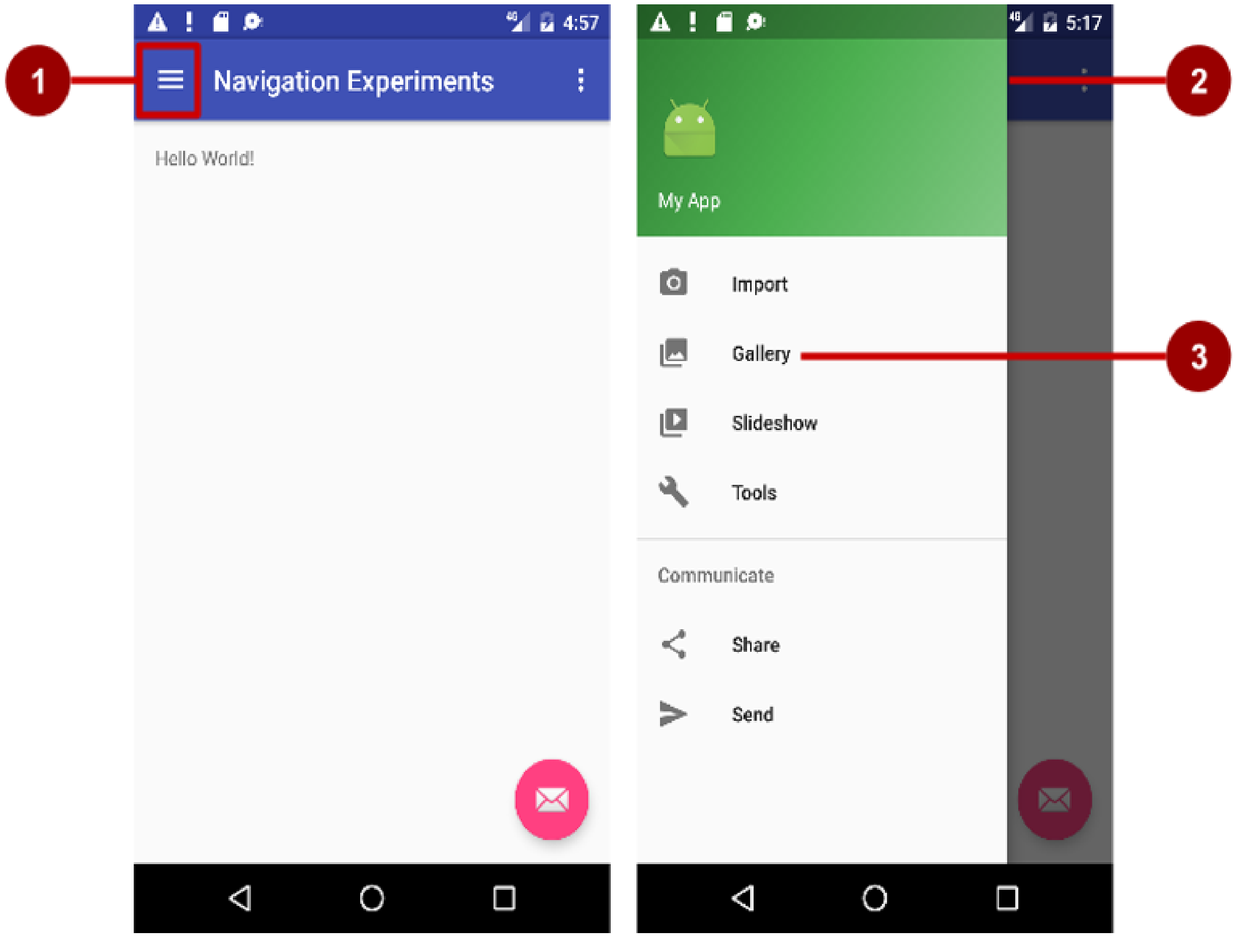



手机上：多个画面

# 抽屉导航栏

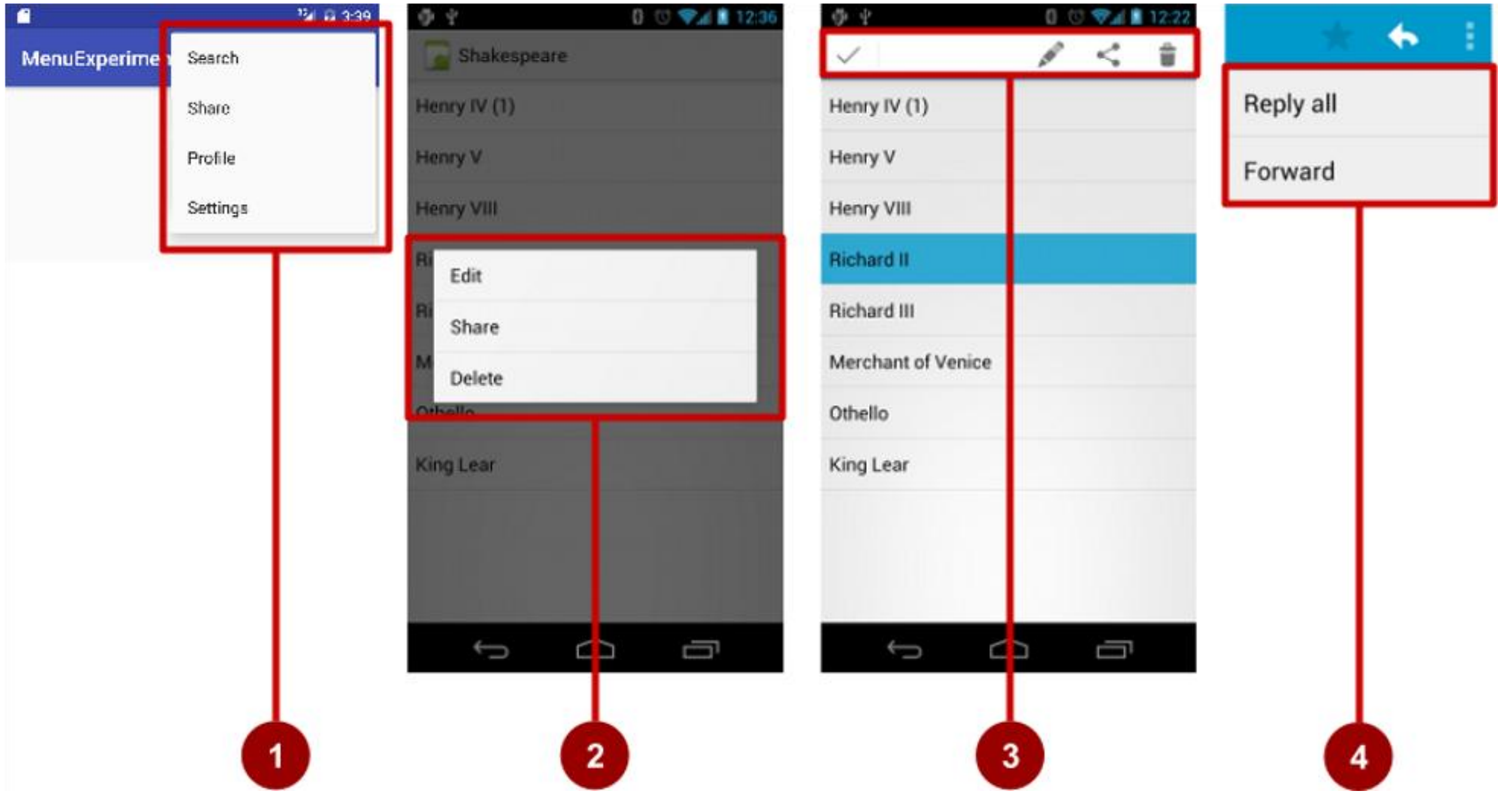


1. 在App导航栏中的图标
2. 头部
3. 菜单栏



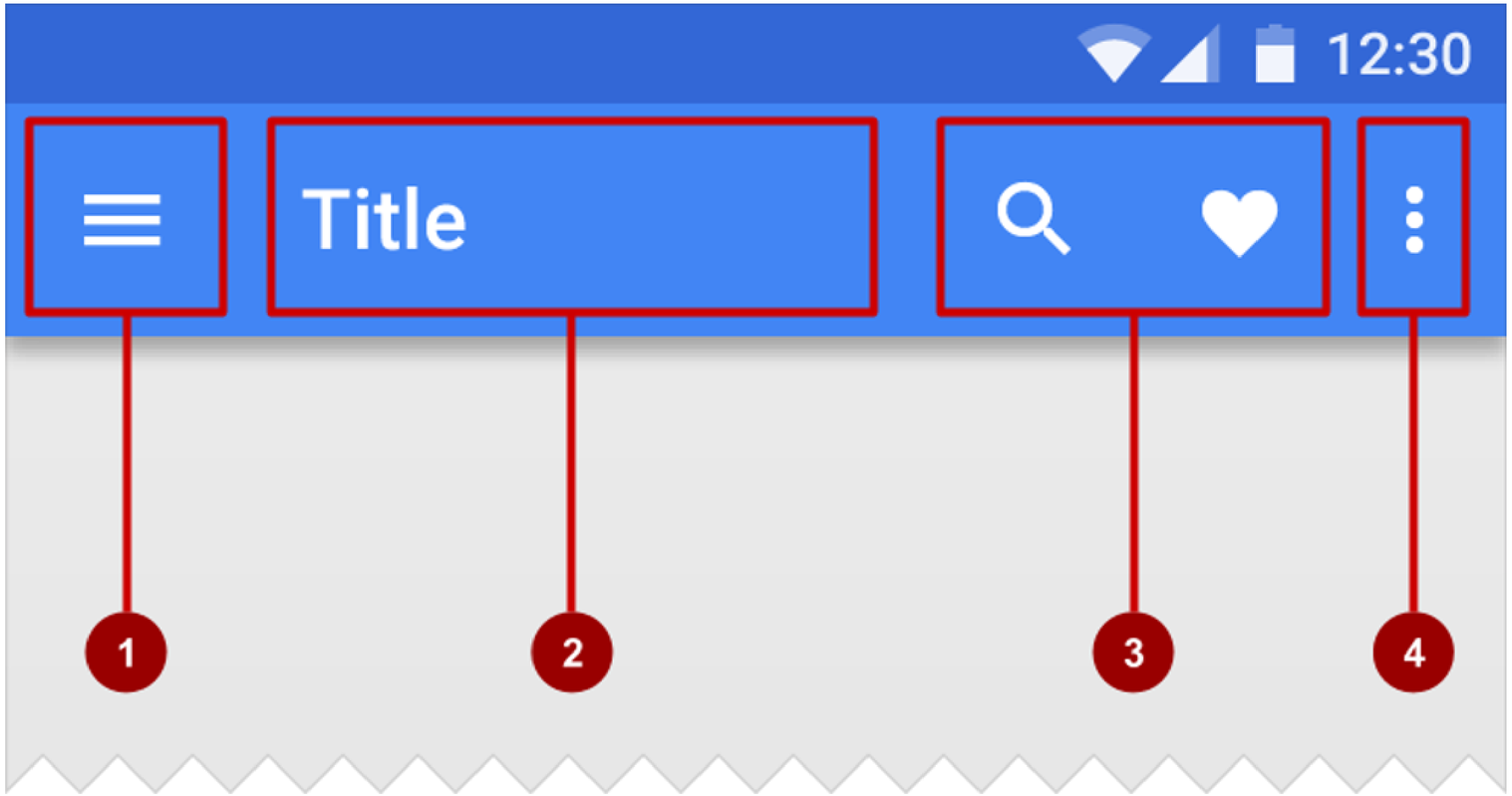
# 菜单

1. 带选项菜单的应用栏
2. 上下文菜单
3. 上下文操作栏
4. 弹出式菜单



# 应用栏

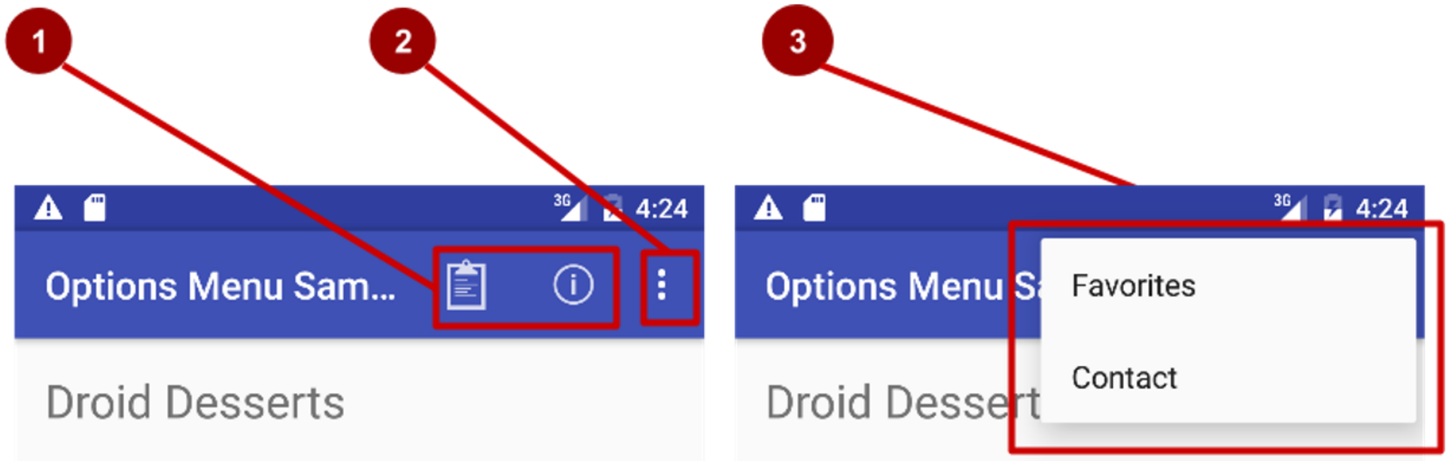
1. 用来打开导航抽屉的导航图标
2. 当前Activity的标题
3. 选项菜单项的图标
4. 其余选项菜单的操作溢出按钮



# 选项菜单

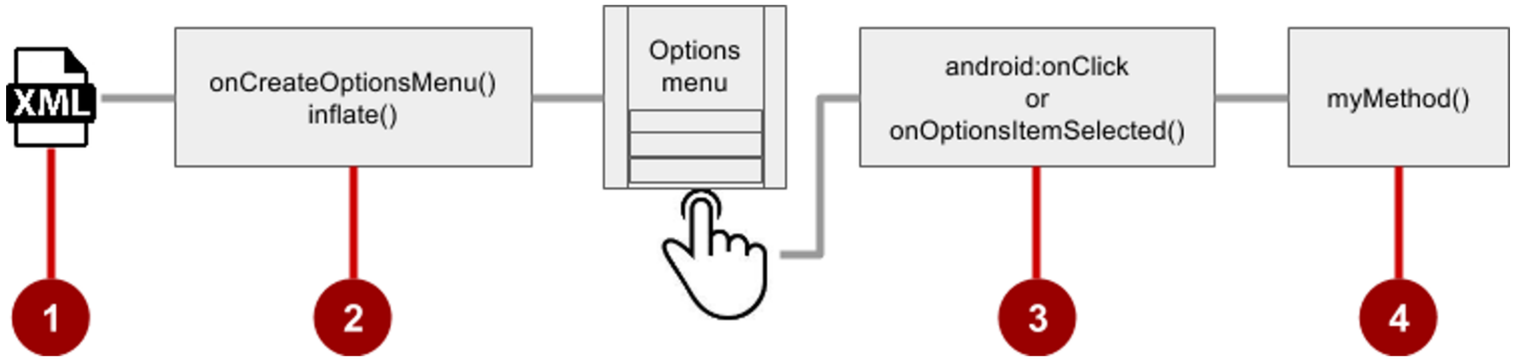
- 通常用于导航到其他Activity和编辑应用程序的设置项

1. 应用程序栏中的重要项目的动作图标
2. 点击三个点，即“动作溢出按钮”来查看更多的选项
3. 这些选项会出现在应用栏的右上角



# 实现选项菜单的步骤

1. 添加XML菜单资源（menu_main.xml）
2. 实现onCreateOptionsMenu()来填充菜单内容
3. 实现onClick()或onOptionsItemSelected()
4. 自定义方法封装处理点击等事件后的逻辑



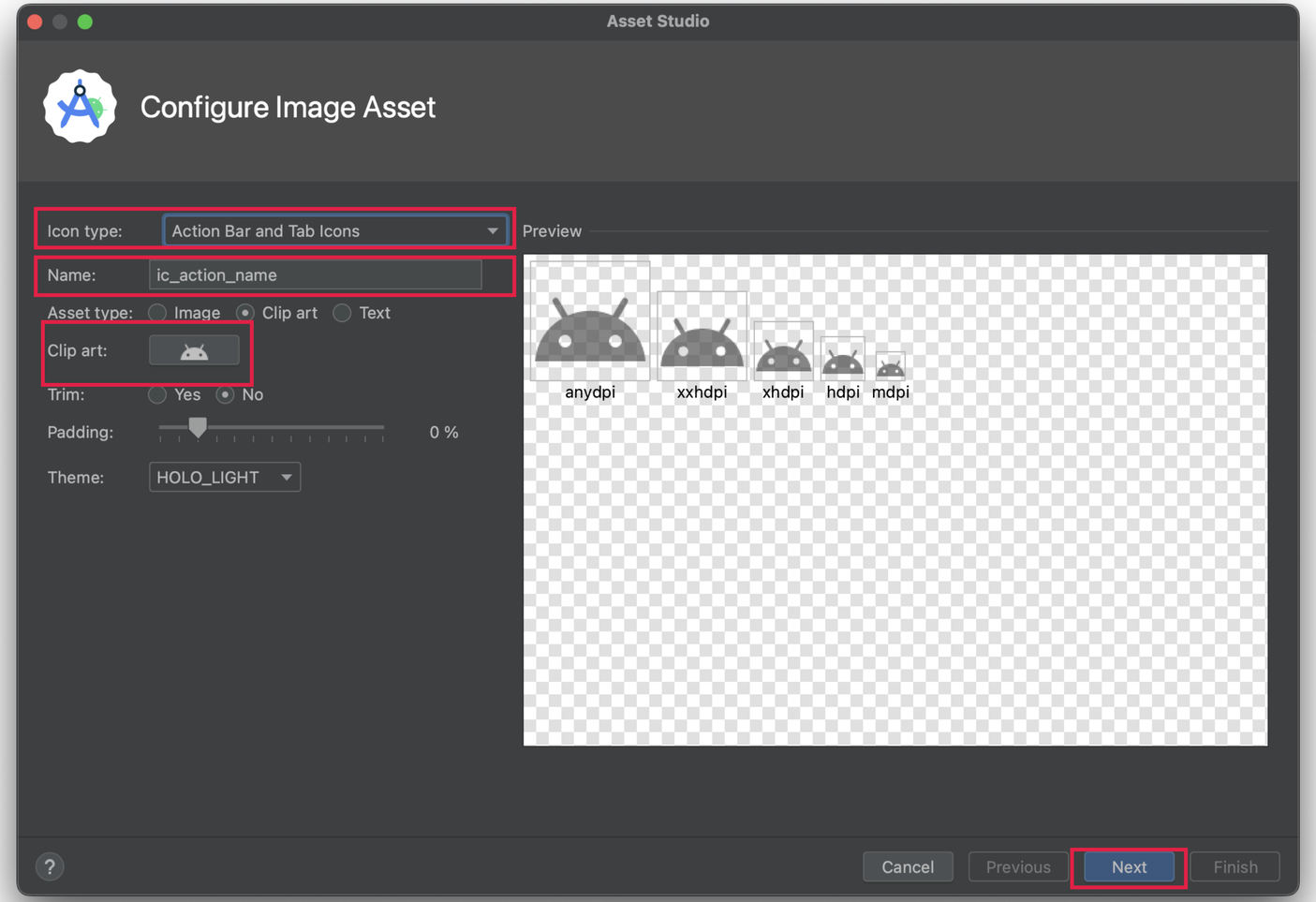
# 1. 创建菜单资源

1. 创建菜单资源目录
2. 创建XML菜单资源（menu_main.xml）
3. 添加条目（如右图的设置和收藏夹）

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

# 为菜单条目添加图标

1. 右键单击drawable
2. 选择New -> Image Asset
3. 选择Action Bar and Tab Items
4. 编辑图标名称
5. 单击Clip Art图像，然后点击图标
6. 单击Next，然后点击Finish



# 添加菜单项属性

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

# 2. 填充选项菜单内容

- 覆盖Activity中的onCreateOptionsMenu()

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

# 3. 覆盖onOptionsItemSelected()

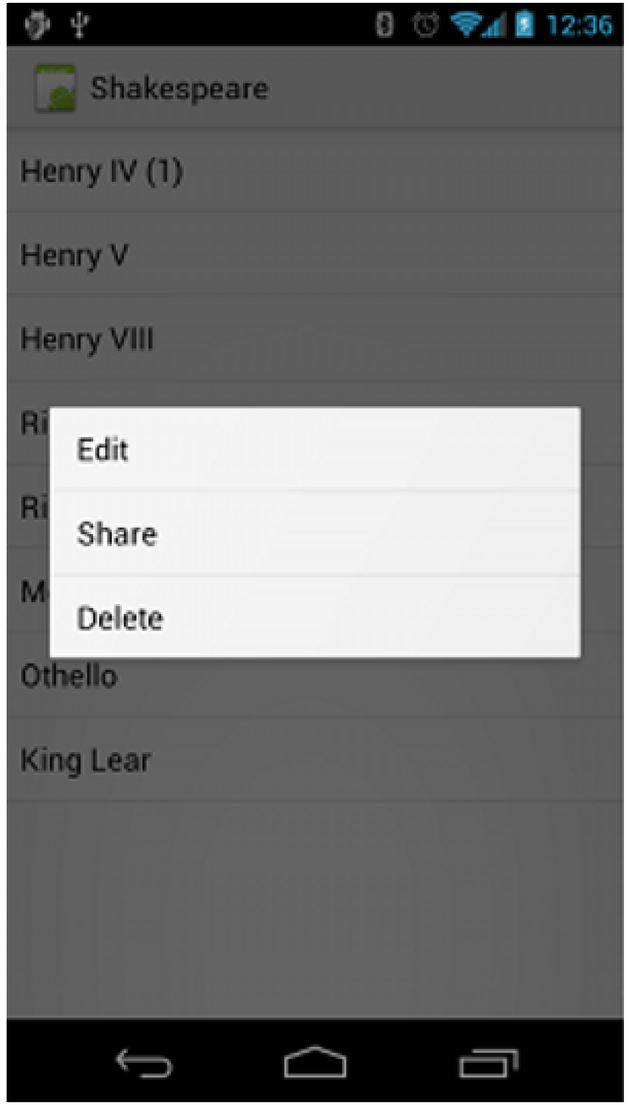
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.action_settings:
         showSettings();
         return true;
      case R.id.action_favorites:
         showFavorites();
         return true;
      default:
         return super.onOptionsItemSelected(item);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

# 上下文菜单

- 允许用户对选定的视图执行操作
- 可以部署在任何视图上
- 常用于RecyclerView，GridView等视图集合中的条目编辑

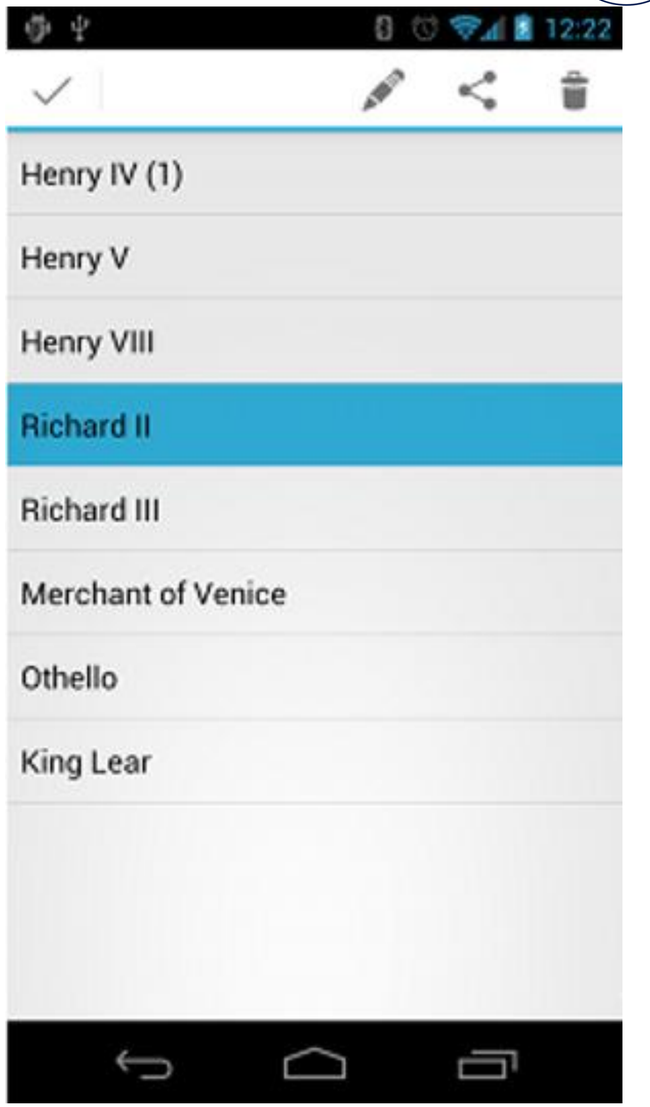
# 上下文菜单的类型

- 漂浮式上下文菜单
  - 通常通过长按视图出现
  - 用户通过操作可以修改视图
  - 用户一次对一个视图执行操作



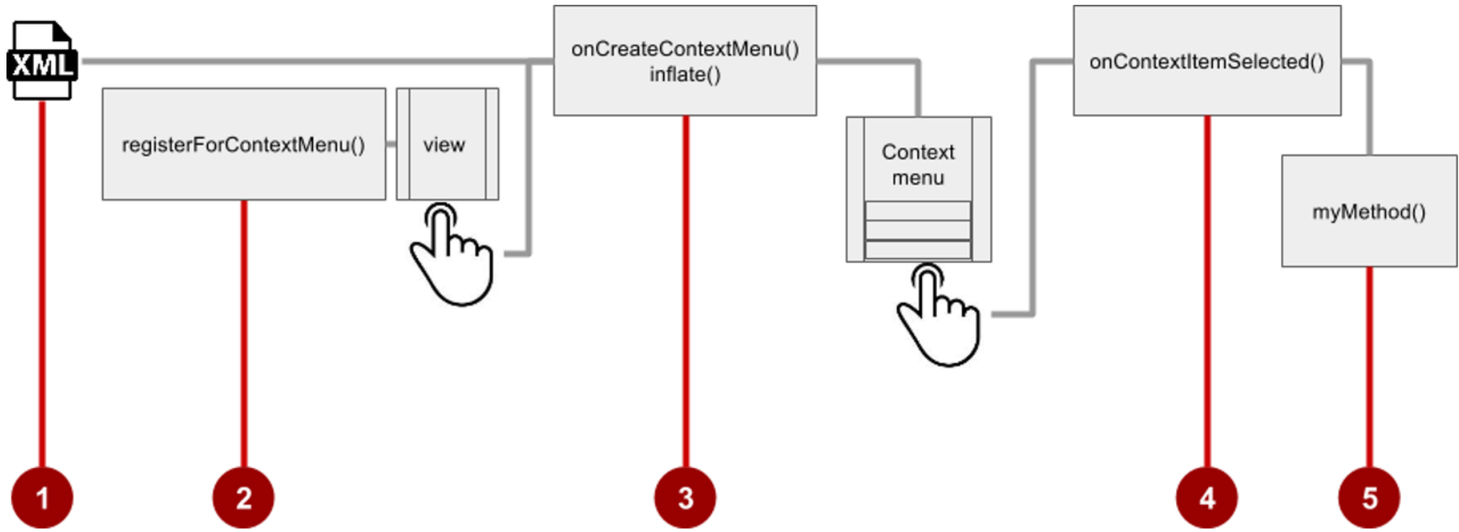
# 上下文菜单的类型

- 上下文操作模式
  - 临时操作栏占满应用栏或位于下方
  - 操作项影响所选的View元素
  - 用户可以一次对多个View元素执行操作



# 创建步骤

1. 创建XML菜单资源文件并分配外观和位置属性
2. 使用registerForContextMenu()注册视图
3. 在Activity中实现onCreateContextMenu()以填充菜单
4. 实现onContextItemSelected()以处理菜单项点击
5. 自定义方法



# 1. 创建菜单资源

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

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

# 2. 注册视图

- 在Activity的onCreate()中，注册视图的菜单监听器View.OnCreateContextMenuListener

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

# 3. 实现onCreateContextMenu()

- 指明具体的上下文菜单

```
@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);
}
```

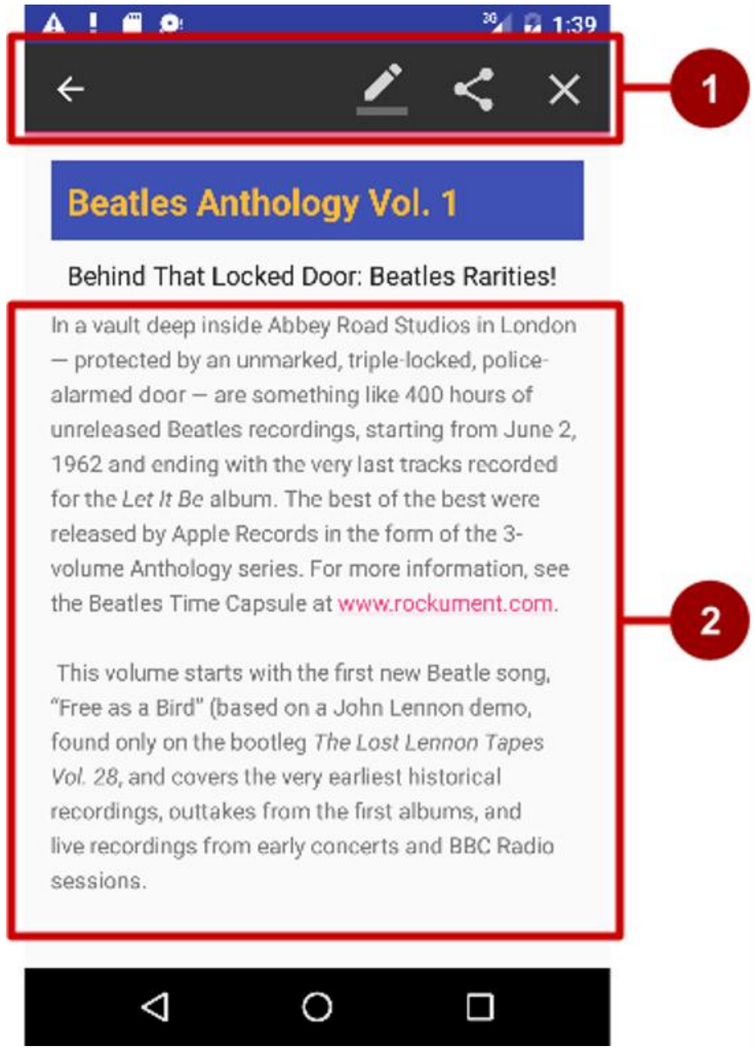
# 操作模式

- 一种UI模式，可以临时替换部分常规的UI交互
- 例如：通过选择文本或长按可以触发操作模式
- 开发者通过startActionMode()来启动操作模式
- 操作模式提供了一系列生命周期的回调接口
  - onCreateActionMode(ActionMode, Menu)
  - onPrepareActionMode(ActionMode, Menu)
  - onActionItemClicked(ActionMode, MenuItem)
  - onDestroyActionMode(ActionMode)

# 上下文操作栏

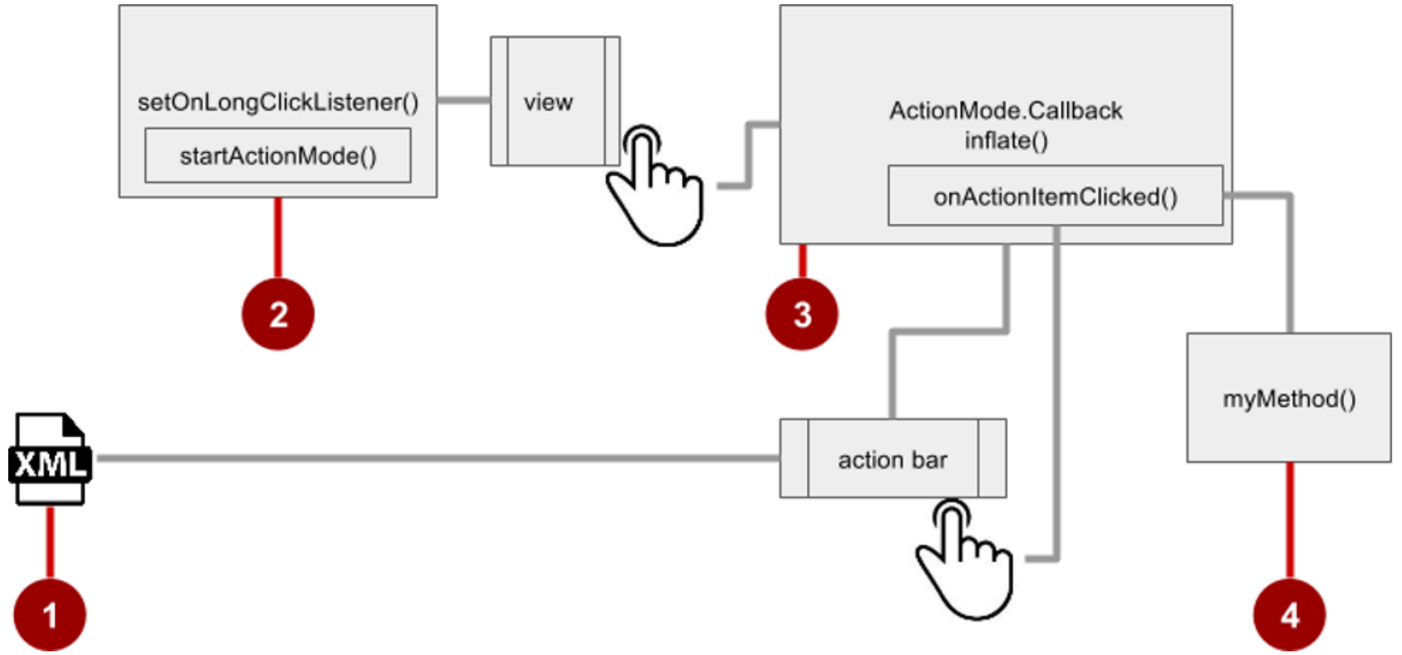
1. 带有操作项的的上下文操作栏
   - 右边是编辑、分享、删除
   - 左边是已完成
   - 当用户点击已完成后，操作栏退出

2. 长按触发上下文操作栏的视图



# 创建上下文操作栏的步骤

1. 创建XML菜单资源文件并分配图标
2. 在视图中设置长按的监听器setOnLongClickListener()，在回调中触发上下文操作栏并调用startActionMode()
3. 实现ActionMode.Callback接口来处理操作模式生命周期，以及实现处理菜单项点击的回调onActionItemClicked()
4. 自定义方法



# setOnLongClickListener()

```
private ActionMode mActionMode;
In 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;
     }
  });
```

# 定义mActionModeCallback

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

# 实现mActionModeCallback

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

# 实现onPrepareActionMode()

- 每次操作模式显示时调用
- 始终在onCreateActionMode()之后调用
- 如果操作模式无效，可能会被多次调用

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

# 实现onActionItemClicked()

- 用户选择某项操作条目时调用
- 此回调方法中处理点击后的逻辑

```
@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;
  }
}
```

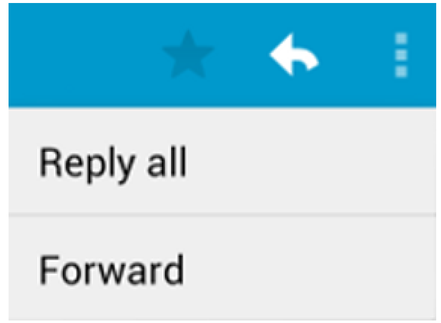
# 实现onDestroyActionMode()

- 当用户退出操作模式时调用

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

# 弹出式菜单

- 和视图锚定的垂直列表
- 通常锚定到一个可见的图标
- 操作不直接影响视图内容
  - 之前提到的菜单溢出图标打开的是弹出式菜单
  - 下图给出的是电子邮件应用中的弹出式菜单例子，其中两个选项“全部回复”和“转发”与电子邮件相关，但不会直接影响邮件内容

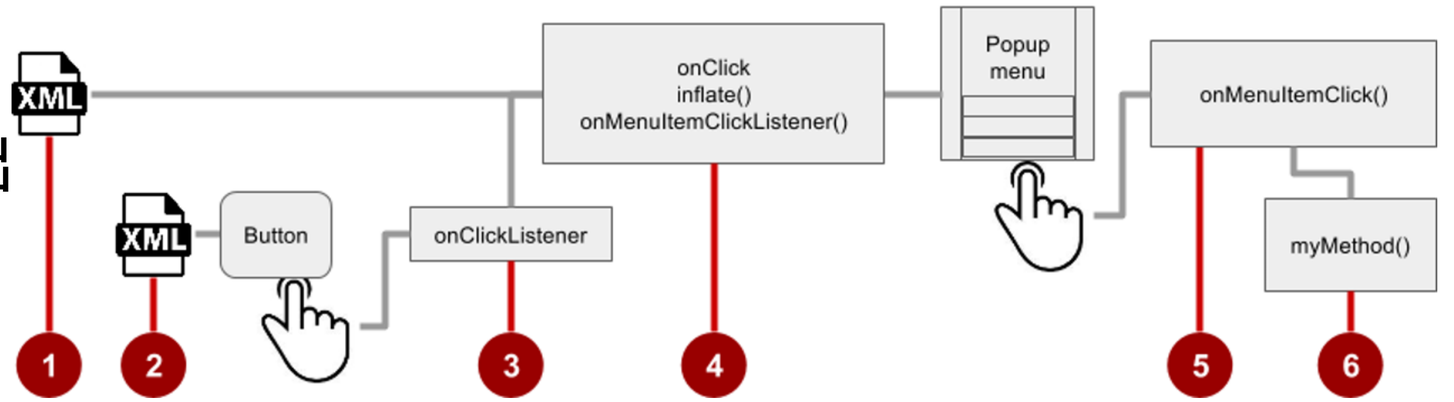


# 创建弹出式菜单的步骤

1. 创建XML菜单资源文件并分配外观和位置属性

2. 在XML活动布局文件中，为弹出式菜单图标添加ImageButton

3. 为ImageButton设置onClickListener

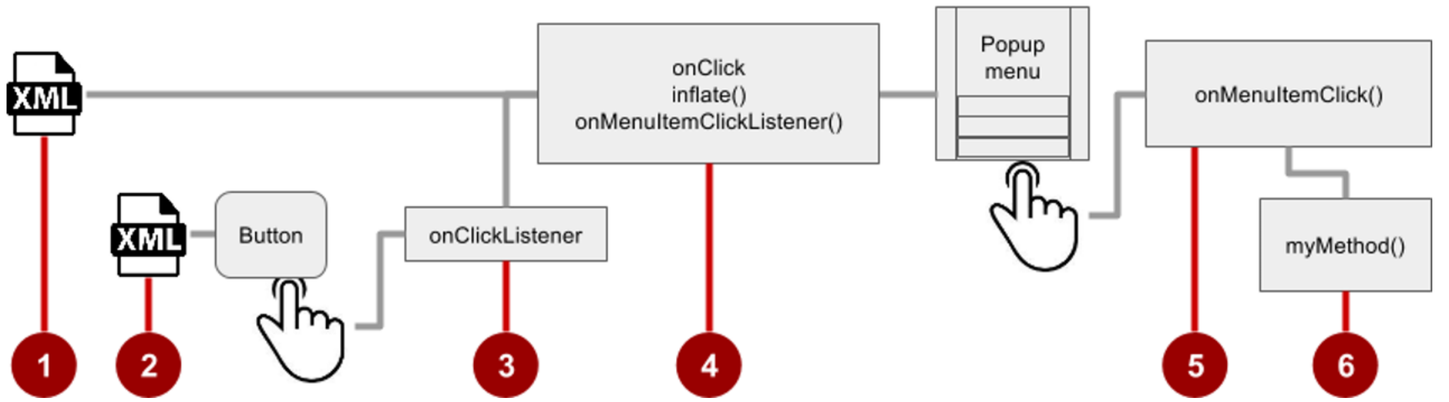


# 创建弹出式菜单的步骤

4. 覆盖onClick()来填充弹出式菜单，并使用onMenuItemClickListener()来注册菜单项的点击监听

5. 实现onMenuItemClick()

6. 自定义方法



# 2. 添加ImageButton

```
<ImageButton
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:id="@+id/button_popup"
 android:src="@drawable/@drawable/ic_action_popup"/>
```

# 3. 为ImageButton设置点击监听

```
private ImageButton mButton;

In onCreate():

mButton = findViewById(R.id.button_popup);

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

# 4. 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();
}
```

# 5. 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;
  }
}
```

# 小结

- 片段 - mini-Activity
  - 时间选择器，日期选择器
- 导航
  - 向后导航，向上导航
- 菜单
  - 带选项菜单的应用栏
  - 上下文菜单
  - 上下文操作栏
  - 弹出式菜单

# 参考资料

- Android Application Fundamentals
- Starting Another Activity
- Activity (API Guide)
- Activity (API Reference)
- Intents and Intent Filters (API Guide)
- Intent (API Reference)
- Navigation

# 参考资料

- Activities (API Guide)
- Activity (API Reference)
- Managing the Activity Lifecycle
- Pausing and Resuming an Activity
- Stopping and Restarting an Activity
- Recreating an Activity
- Handling Runtime Changes
- Bundle

# 参考资料

1.Navigation Design guide

d.android.com/design/patterns/navigation.html

2.Designing effective navigation

d.android.com/training/design-navigation/index.html

3.Creating a Navigation Drawer

d.android.com/training/implementing-navigation/nav-drawer.html

4.Creating swipe views with tabs

d.android.com/training/implementing-navigation/lateral.html

# 参考资料


5.Adding the App Bar
6.Menus
7.Menu Resource
8.Fragments
9.Dialogs
10.Pickers
11.Drawable Resources


浙江大学
ZHEJIANG UNIVERSITY

# Thanks!