

移动应用软件开发

本次课内容：应用基本组件Activity

王继良
软件学院
时间：周三（2）





目录

1 Activity

2 Intent

3 隐式Intent

4 生命周期

5 Android与iOS异同

ACK: PPT的很多内容都来自Google官方材料



目录

Activity

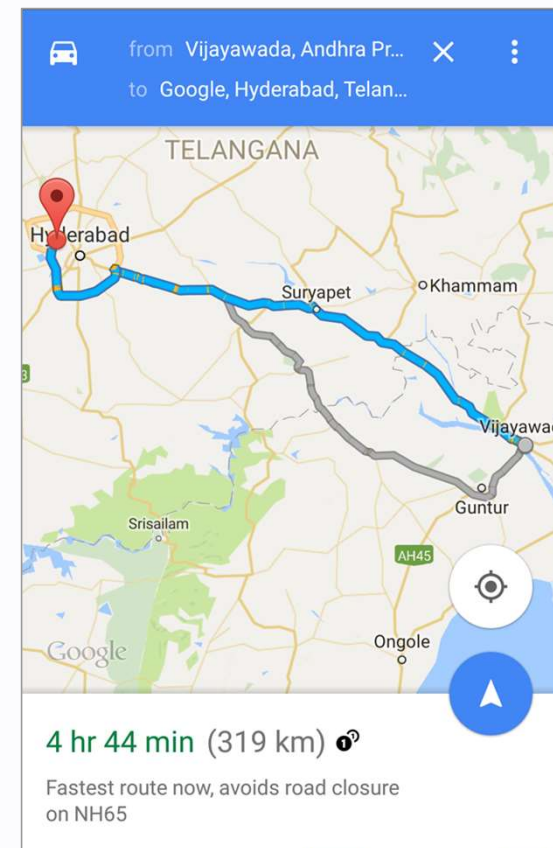
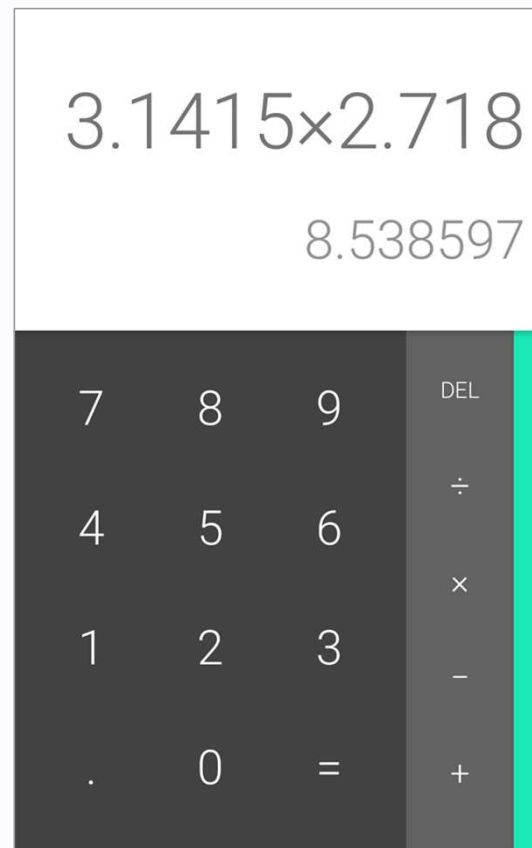
1.1 Activity 的定义

1.2 Activity 的作用

1.3 Activity 的组成部分

1.4 Activity 的创建和声明

如何理解一个移动应用？



想象一个窗户上真的需要贴上这些文字

Activity, Window, View,
LayoutInflater, Xml。
相互配合，最终给我们展示出应
用精美的可以交互的界面。

My Food List

Cheese

Pepperoni

Black Olives

Pineapple

Strawberries

Artichokes

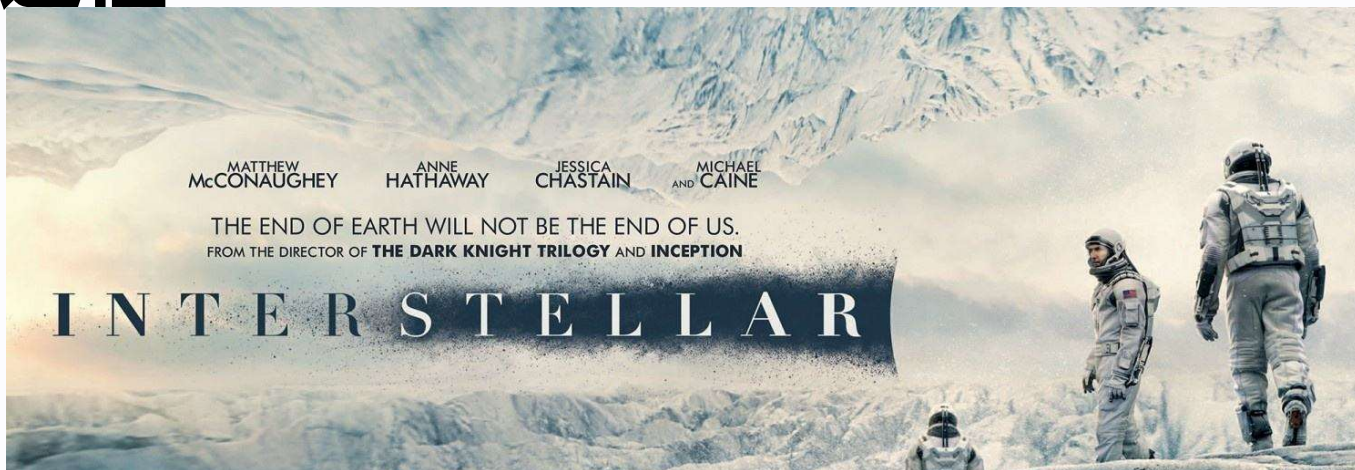
Red peppers

Mushrooms

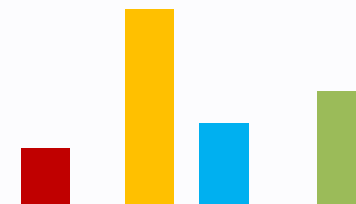


电影的拍摄过程

你看到的场景→



实际：导演，演员，
场务，布景师，灯光
师，造型师，化妆师，
后期处理，剪辑



Activity是什么?

- Activity 是一个应用组件
 - ART提供的一个基本功能
- Activity 是应用跟用户直接的交互接口
- Activity 是一个Java类, 每个 Activity 都是一个单独的文件
- Activity = 界面+交互+...
- Activity
 - 有一个用于绘制用户界面的窗口, 窗口通常会充满屏幕, 但也可小于屏幕并浮动在其他窗口之上
 - 定义窗口上的操作和事件相应
 - ...



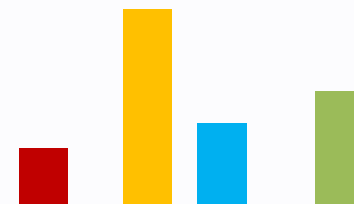
Activity能做什么？

- Activity 可以执行拨打电话、拍摄照片、发送电子邮件或查看地图等操作
- Activity 可以提供让用户进行交互的屏幕，例如点击按钮，输入文本或登陆验证等操作
- 每个 Activity 均可启动另一个 Activity，无论对方是在当前 app 还是其他 app 中
- Activity 拥有完整的生命周期：创建，启动，运行，暂停，恢复，停止和销毁

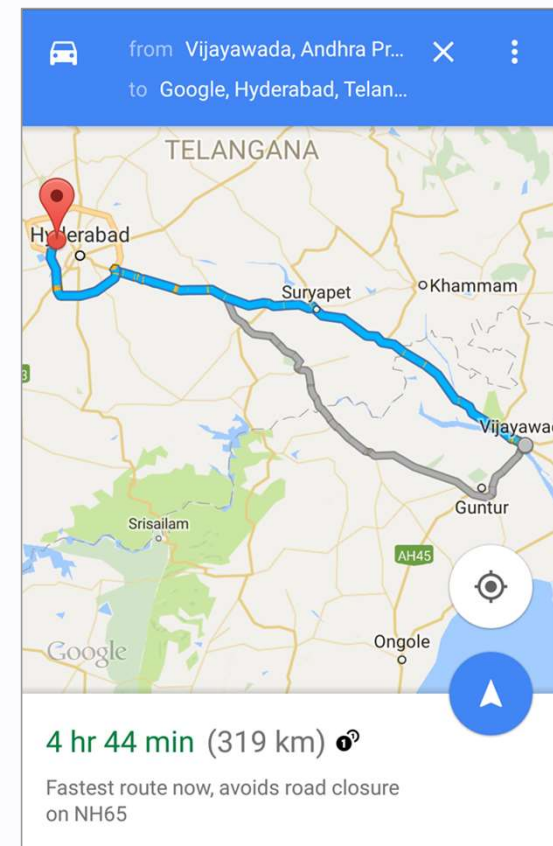
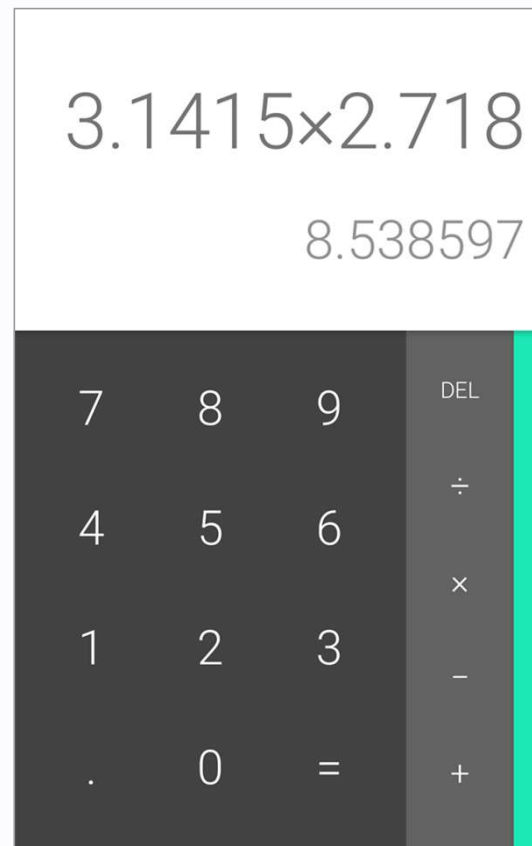


如何理解Activity

- Activity和移动应用的组织形式
- Activity和移动操作系统的基本原理



Activity举例



Activities and Apps

- 一个应用通常由多个彼此联系的 Activity 组成
APP=[Activity]+
- 可以指定某个 Activity 为 “主” Activity，即首次启动应用时呈现给用户的那个 Activity
- 可以在 Android Manifest 文件中以父子类的方式来组织 Activities 的关系，辅助导航功能



Activities and Layouts

- 每个 Activity 通常都具有一个单独的 UI 布局(Layouts)
- Layouts 通常会在一个或多个XML文件中定义
- Activity 会在创建时会将布局 “inflates”



如何实现一个新的Activity?

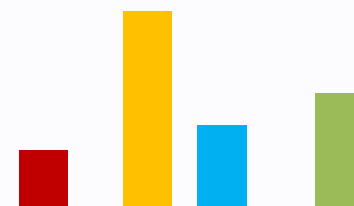
1. 在 XML 文件中定义 layout **(定义样式)**
2. 创建 Activity Java class **(定义操作)**
 - 继承 AppCompatActivity
3. 将 Activity 与 Layout 连接起来 **(关联操作和样式)**
 - 在 onCreate() 文件中设置Activity 用户界面的布局
4. 在 Manifest 文件中声明 Activity



如何实现一个新的Activity?

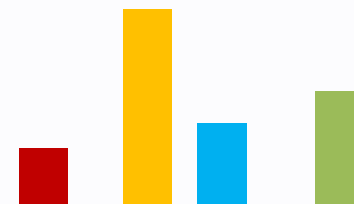
1. 在 XML 文件中定义 layout
2. 创建 Activity Java class
 - 继承 AppCompatActivity
3. 将 Activity 与 Layout 连接起来
 - 在 onCreate() 文件中设置Activity 用户界面的布局
4. 在 Manifest 文件中声明 Activity

运行2后其他几个步骤是自动完成的。



在 XML 文件中定义 layout

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



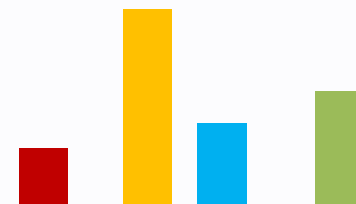
XML 文件存在的意义

- 设计视图和定义布局的最常见方法是借助保存在应用资源内的 XML 布局文件
- 通过这种方式可以将用户界面的设计与定义 Activity 行为的源代码分开维护



区别？

- 你觉得使用的时候感受区别最大在哪里？



回忆一下HTML程序写法

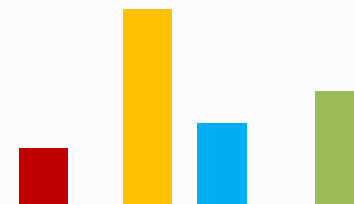
布局:

```
<button type="button" onclick="inputbtn()">
  Click
</button>
<input id="inputtext"> </input>
```

事件相应:

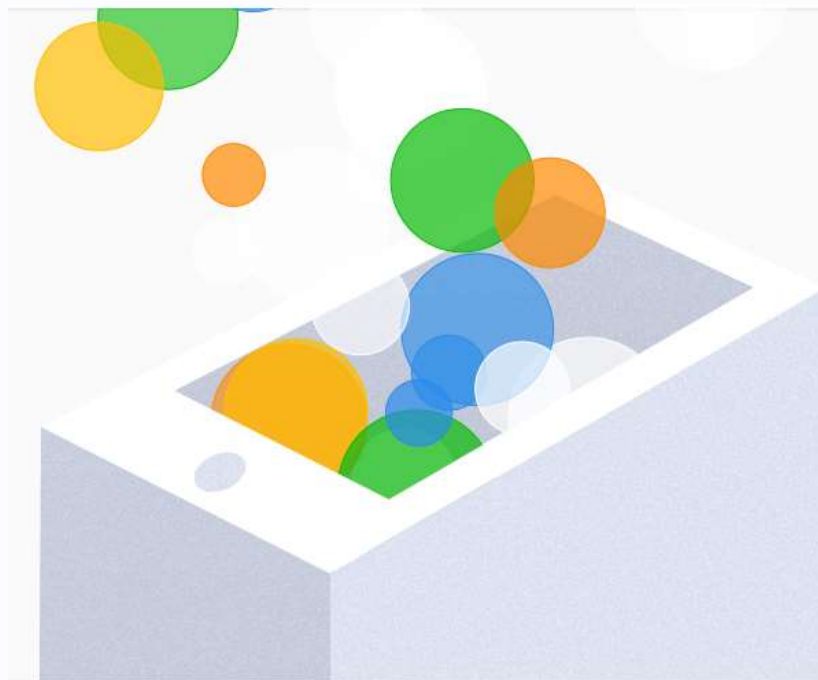
```
function inputbtn(){
  let inputTextV =
  document.getElementById('inputtext').value;
  ...
}
```

```
<style type="text/css">
  div.LARGE-TIMER{
    /*width: 100px;*/
    /*height: 100px;*/
    /*background: green;*/
    /*position: absolute;*/
    /*left: 0px;*/
    text-align: center;
    font-size: 20vw;
    font-family: Arial;
    position: center;
  }
</style>
```



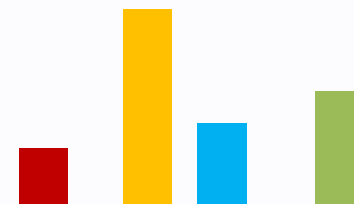
微信小程序开发

- WXSS
- js
- json

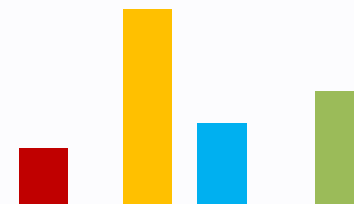


微信小程序

小程序是一种新的开放能力，开发者可以快速地开发一个小程序。小程序可以在微信内被便捷地获取和传播，同时具有出色的使用体验。

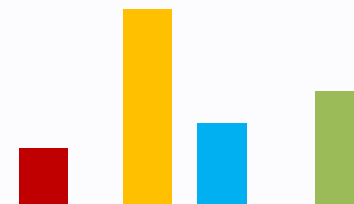
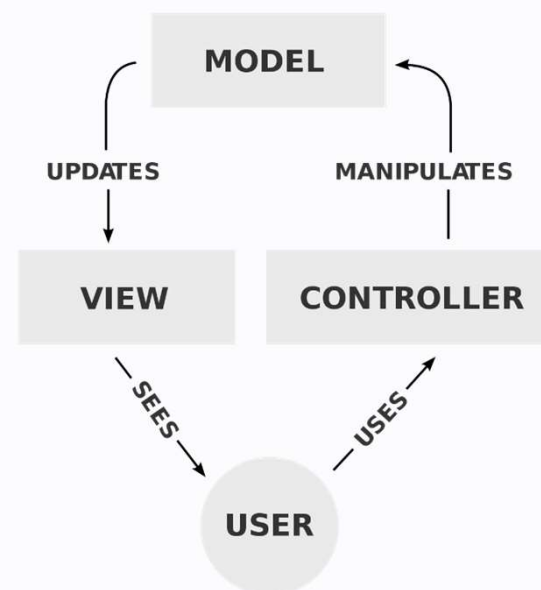


回忆一下MFC



MVC的发展

- Model
- View
- Controller
- 小程序设计过程大家应该有深刻体会



创建 Activity Java class

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

必须创建 Activity 的子类（或使用其现有子类）



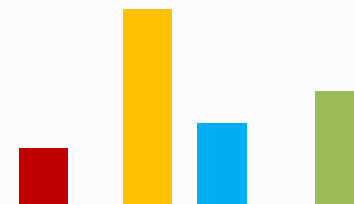
将 Activity 与 Layout 连接起来

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

Resource

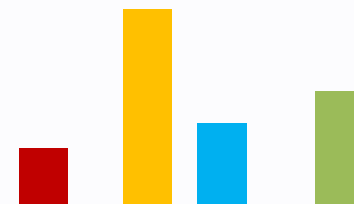
is layout

in this XML file



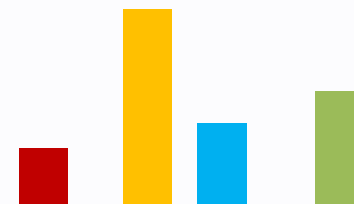
用户界面与实现

- Activity 的界面由层级式视图(衍生自 View 类的对象)提供
- 每个视图都控制 Activity 窗口内的特定矩形空间, 可对用户交互作出响应, 例如在用户触摸时启动某项操作的按钮
- 可以利用 Android 提供的许多现成视图设计和组织布局



创建 Activity 的重要方法

- onCreate()：系统会在创建 Activity 时调用此方法，因此必须实现。在实现内初始化必需组件并调用 setContentView() 以定义用户界面的布局。
- onPause()：系统将此方法作为用户离开 Activity 的第一个信号进行调用。
- 还有几种其他回调方法以便提供流畅用户体验，以及处理 Activity 的不同状态。



一个App通常包含多个Activity?

- 如何实现?
- 应该要实现什么功能?



一个App通常包含多个Activity

- 在什么条件转化到哪个Activity?
 - 点击按钮，跳转到购物车Activity
 - 点击按钮，跳转到登录Activity
- 带着什么数据去，带着什么数据回来。



在 manifest 文件中声明 Activity

MainActivity 需要包含 intent-filter 以便从程序图标界面启动

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

```
  <intent-filter>
```

```
    <action android:name="android.intent.action.MAIN" />
```

```
    <category android:name="android.intent.category.LAUNCHER" />
```

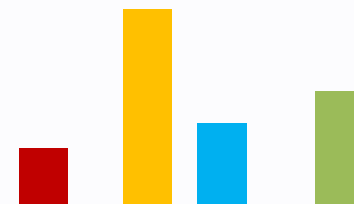
```
  </intent-filter>
```

```
</activity>
```



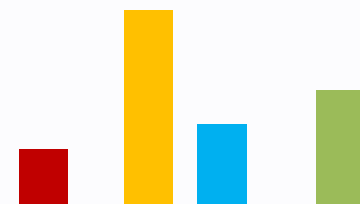
在 manifest 文件中声明其他内容

- 可以加入其他特性，以定义 Activity 标签、Activity 图标或风格主题等用于设置 Activity UI 风格的属性
- android:name 属性指定类名，是唯一必需的属性，不应更改否则会破坏诸如应用快捷方式等一些功能
- 系统自动创建的主 Activity 包含一个 Intent 过滤器，声明了该 Activity 置于 launcher 类别内





Intent





目录

Intent

2.1 Intent 的定义

2.2 Intent 的作用

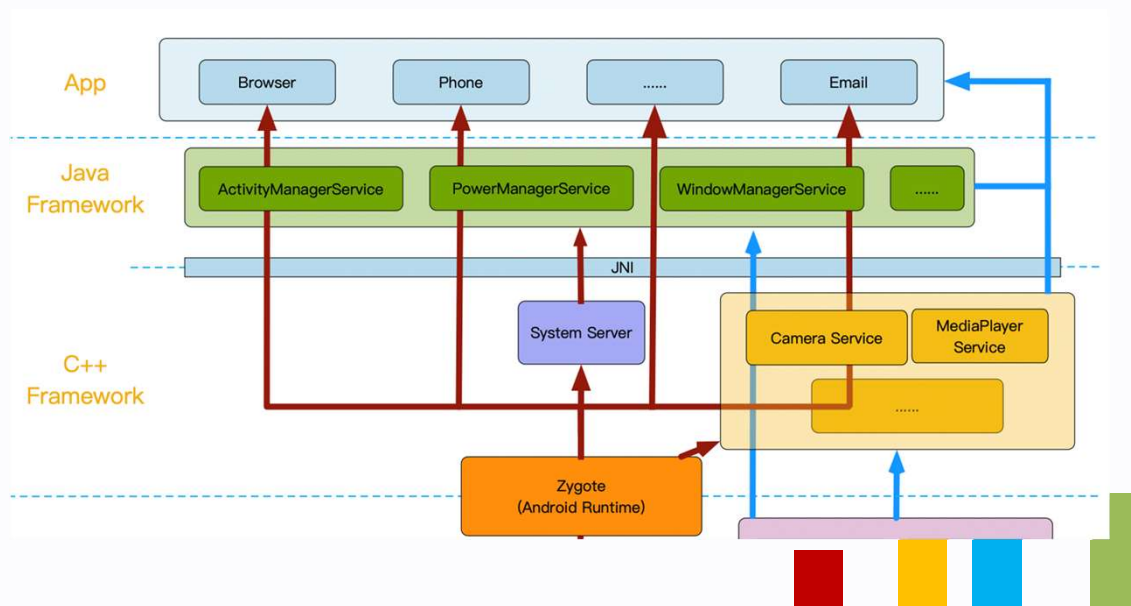
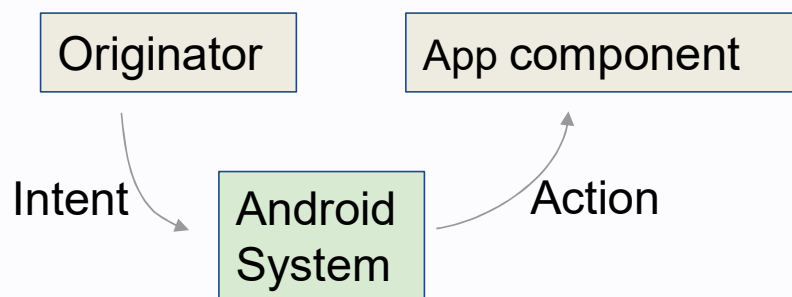
2.3 Intent 的类型

2.4 Intent 的构建

2.5 Intent 的使用

Intent 是什么?

- Intent 是一个消息传递对象
- Intent 是对要执行的操作的描述
- Intent 通过 Android 系统从其他应用组件请求操作



Android虚拟机如何启动不同Activity

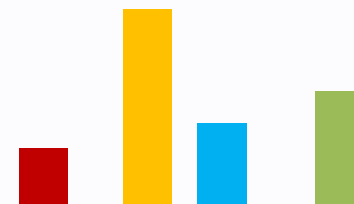
- Intent是交互的主要手段



Intent 能做什么

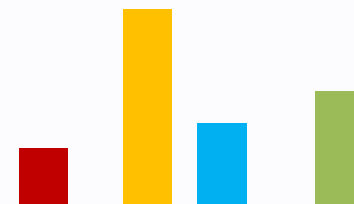
- 启动Activity
 - 单击按钮即可启动新的活动以输入文本
 - 单击共享可打开一个应用可以发布照片
- 启动服务
 - 开始在后台下载文件
- 传递广播
 - 系统通知所有应用手机正在充电
- 启动新的实例/应用
- 一句话：intent用来告诉Android下一步操作是什么？

Android四大组件：Activity、Service、Broadcast Receiver、Content Provider。



显式和隐式 Intent

- 显式Intent：（直接告诉Android）启动某个具体的活动/组件
 - 启动应用内的新 Activity 以响应用户操作
 - 启动服务以在后台下载文件
- 隐式Intent：（要求系统）找到可以处理此请求的活动/组件
 - 请求在地图上显示用户指定的位置
 - 请求分享信息到网络

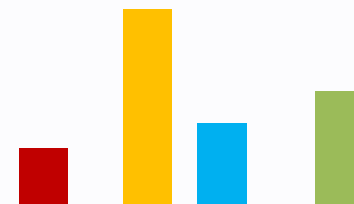


显式 Intent 示例

//在某个活动中执行, 因此“ this”由上下文可得

//文件Ur1是字符串类型的URL

```
Intent downloadIntent = new Intent(this, DownloadService.class);  
downloadIntent.setData(Uri.parse(fileUr1));  
startService(downloadIntent);
```



隐式 Intent 示例（思考每一行是什么意思？）

//用字符串创建文本信息

```
Intent sendIntent = new Intent();
```

```
sendIntent.setAction(Intent.ACTION_SEND);
```

```
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
```

```
sendIntent.setType("text/plain");
```

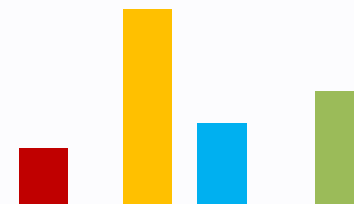
//验证有Activity可以被Intent执行

```
if (sendIntent.resolveActivity(getPackageManager()) != null) {
```

```
    startActivity(sendIntent);
```

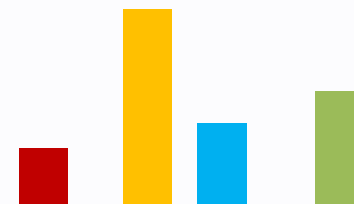
```
}
```

运行的结果是什么？



如何构建 Intent ?

- **组件名称：** 由名称定义的应用组件，没有即隐式
 - 例如：com.example.ExampleActivity
- **操作：** 指定要执行的通用操作的字符串
 - ACTION_VIEW：可向用户显示信息（使用图库应用查看的照片；或者要使用地图应用查看的地址）
 - ACTION_SEND：可发送或者共享数据（电子邮件应用或社交共享应用）
- **数据：** 引用待操作数据和/或该数据 URI
 - 提供的数据类型通常由 Intent 的操作决定



如何构建 Intent ?

- **类别：** 应处理 Intent 组件类型
 - CATEGORY_BROWSABLE: 目标 Activity 允许通过浏览器启动, 以显示链接引用的数据, 如图像或电子邮件。
 - CATEGORY_LAUNCHER: 该 Activity 是任务的初始 Activity, 在系统的应用启动器中列出。
- **Extra:** 完成请求操作所需的附加信息的键值对。
 - EXTRA_EMAIL : 指定目标收件人
- **标志:** 元数据, 标志系统如何启动和处理Activity
 - Activity 应属于哪个任务
 - Activity 是否属于最近的 Activity 列表



Intent 示例

- 展示网页

```
Uri uri = Uri.parse("http://www.google.com");  
Intent it = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(it);
```

- 拨打电话

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



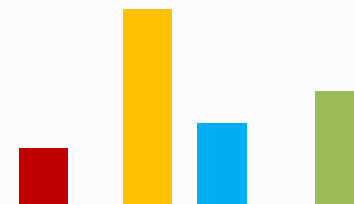
使用 Intent data 发送URL示例

```
// 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")));
```



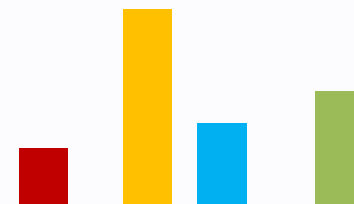
使用 Intent extras 发送信息示例

- `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);`
⇒ 如果数据量过多, 可以创建 bundle 集合所有数据并发送



使用 extras 向 Activity 发送数据

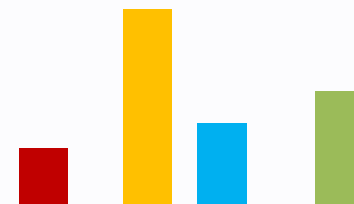
```
public static final String EXTRA_MESSAGE_KEY =  
    "com.example.android.twoactivities.extra.MESSAGE";  
  
Intent intent = new Intent(this, SecondActivity.class);  
String message = "Hello Activity!";  
intent.putExtra(EXTRA_MESSAGE_KEY, message);  
startActivity(intent);
```



使用 type 和 category 发送数据

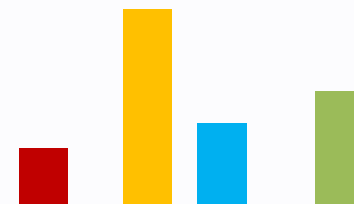
- 设置 mime type 和 category

```
intent.setType("application/pdf"); // set MIME type  
intent.addCategory(Intent.CATEGORY_OPENABLE);
```



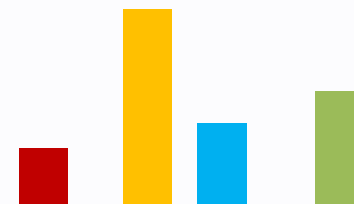
从 Intent 获取数据

- `getData();`
⇒ `Uri locationUri = intent.getData();`
- `getIntExtra (String name, int defaultValue)`
⇒ `int level = intent.getIntExtra("level", 0);`
- `Bundle bundle = intent.getExtras();`
⇒ 同理使用 `bundle` 可以一次性接受所有数据



如何从 Activity 返回数据？（阅读）

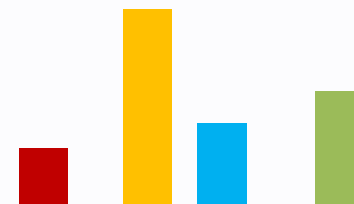
- 使用 `startActivityForResult()` 启动第二个活动
- 从第二个活动返回数据的流程：
 - 创建一个新的 Intent
 - 使用 `putExtra()` 将响应数据放入 Intent
 - 正常返回则将结果设置为 `Activity.RESULT_OK`
 - 如果用户取消则设置为 `RESULT_CANCELED`
 - 调用 `finish()` 关闭活动
- 在第一个 Activity 中实现 `onActivityResult()`



startActivityResult 函数解析

```
public static final int CHOOSE_FOOD_REQUEST = 1;  
Intent intent = new Intent(this, someActivity.class);  
startActivityResult(intent, CHOOSE_FOOD_REQUEST);
```

- 启动 Activity (intent) 并为其分配标识符 (requestCode)
- 通过Intent Extras返回数据
- 完成后弹出堆栈, 返回上一个Activity, 然后执行 onActivityResult() 回调以处理返回的数据
- 使用 requestCode 标识返回了哪个活动



返回数据并完成第二个 Activity

```
// Create an intent
Intent replyIntent = new Intent();

// Put the data to return into the extra
replyIntent.putExtra(EXTRA_REPLY, reply);

// Set the activity's result to RESULT_OK
setResult(RESULT_OK, replyIntent);

// Finish the current activity
finish();
```



实现 onActivityResult

```
public void onActivityResult(int requestCode,
                             int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == TEXT_REQUEST) { // Identify activity
        if (resultCode == RESULT_OK) { // Activity succeeded
            String reply =
                data.getStringExtra(SecondActivity.EXTRA_REPLY);
            // ... do something with the data
        }
    }
}
```





目录

隐式Intent

3.1 隐式 Intent 与用户和系统

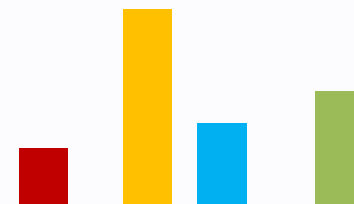
3.2 隐式 Intent 的不同响应模式和两点注意

3.3 隐式 Intent 应用选择器和 filter

3.4 隐式 Intent 和 Activity 运行的整体流程

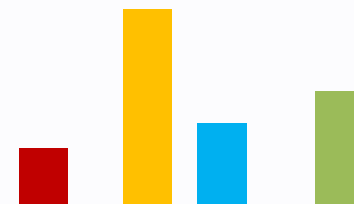
隐式 Intent 与 用户

- 通过描述打算执行的动作（例如分享文章，查看地图或拍照）在另一个应用程序中启动活动
- 指定一个动作并有选择地提供执行动作所用的数据
- 不指定目标Activity类，仅指定预期的动作



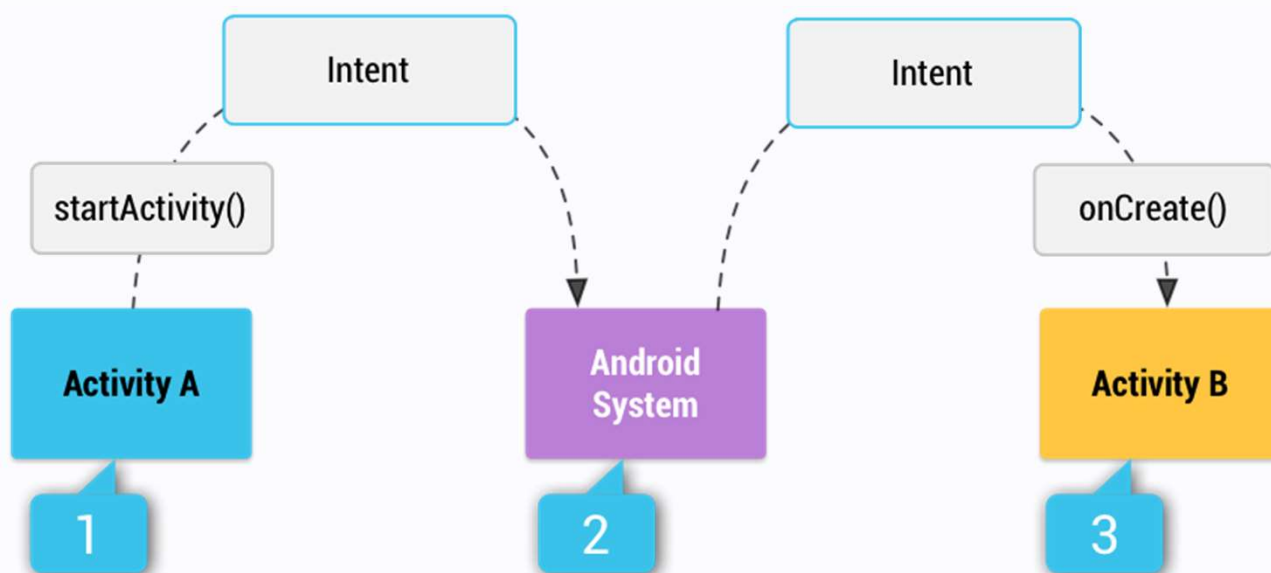
隐式 Intent 与 系统

- 使用隐式 Intent 时，Android 系统通过将 Intent 的内容与在设备上其他应用的清单文件中声明的 Intent 过滤器进行比较，从而找到要启动的相应组件。
- 如果 Intent 与 Intent 过滤器匹配，则系统将启动该组件，并向其传递 Intent 对象。
- 如果多个 Intent 过滤器兼容，则系统会显示一个对话框，支持用户选取要使用的应用。



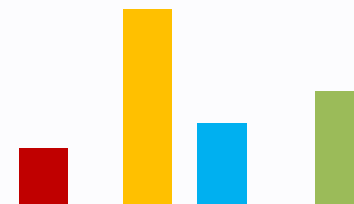
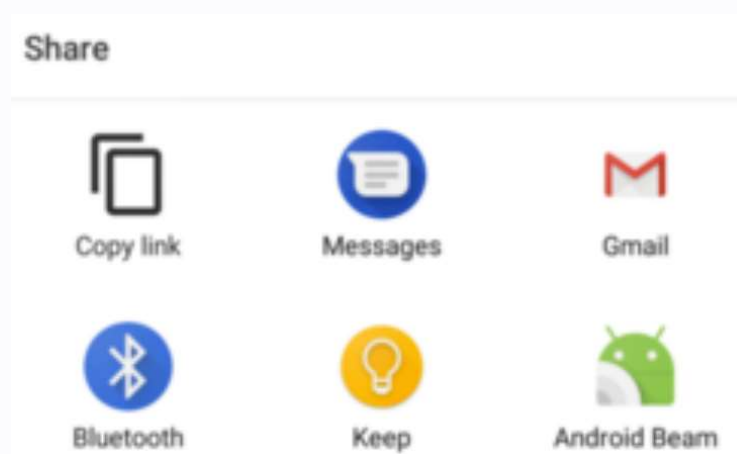
隐式 Intent 与 系统 图示

- A 创建包含操作描述的 Intent，并将其传递给 startActivity()
- Android 系统搜索所有应用中与 Intent 匹配的过滤器
- 该系统通过调用匹配 B 的 onCreate() 方法并将其传递给 Intent，以此启动匹配 Activity。



隐式 Intent 的不同响应模式

- 如果有多个应用响应隐式 Intent，则用户可以选择要使用的应用，并将其设置为该操作的默认选项。
- 如果多个应用可以响应 Intent，且用户可能希望每次使用不同的应用，则应采用显式方式显示选择器对话框。



隐式 Intent 的两点注意

- 用户没有任何应用能处理隐式 Intent 时，发送请求的应用也会崩溃。因此先调用 `resolveActivity()` 验证接收方是否为空，至少存在一个时再进行安全调用。
- 为了确保应用的安全性，启动 Service 时请始终使用显式 Intent，且不要为服务声明 Intent 过滤器。因为您无法确定哪些服务将响应 Intent，且用户无法看到哪些服务已启动。



应用选择器示例

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
// 尽量使用字符串资源调用UI组件
String title = getResources().getString(R.string.chooser_title);
// 创建 intent 以展示选择器对话框
Intent chooser = Intent.createChooser(sendIntent, title);
// 验证原始 intent 可以收到至少一个可处理的activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```



注册 filter 以接收隐式 Intent

- 在manifest文件中为 Activity 声明一个或多个Intent过滤器
- 过滤器揭示 Activity 接受隐式 Intent 的能力



Filter 示例

```
<activity android:name="ShareActivity">  
  <intent-filter>  
    <action android:name="android.intent.action.SEND"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:mimeType="text/plain"/>  
  </intent-filter>  
</activity>
```

数据测试

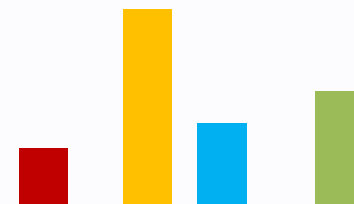
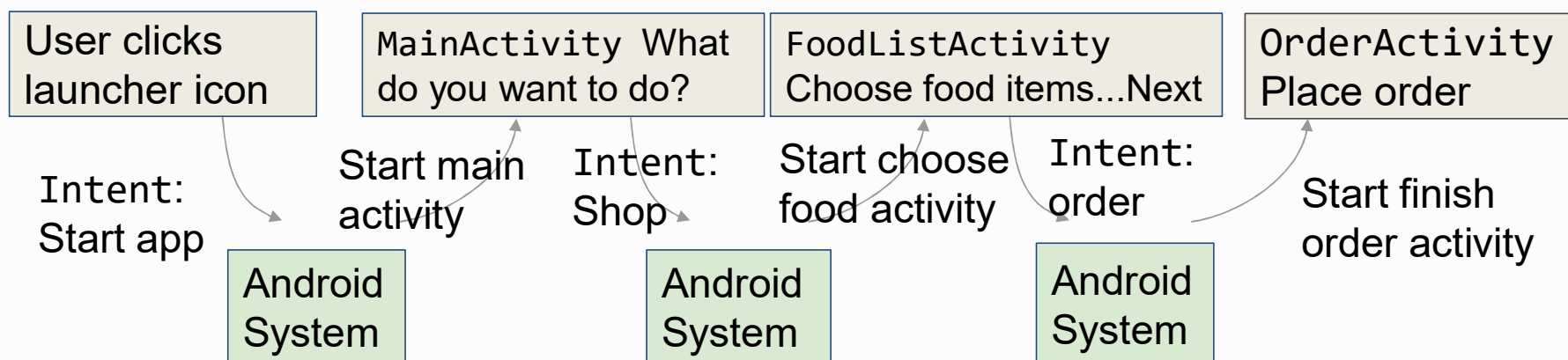
操作测试

类别测试



Intent 和 Activity 运行的整体流程

- 所有Activity实例均由Android runtime管理
- 用 Intent 向Android runtime 发出一条消息来运行Activity





目录

生命周期

4.1 Activity 生命周期

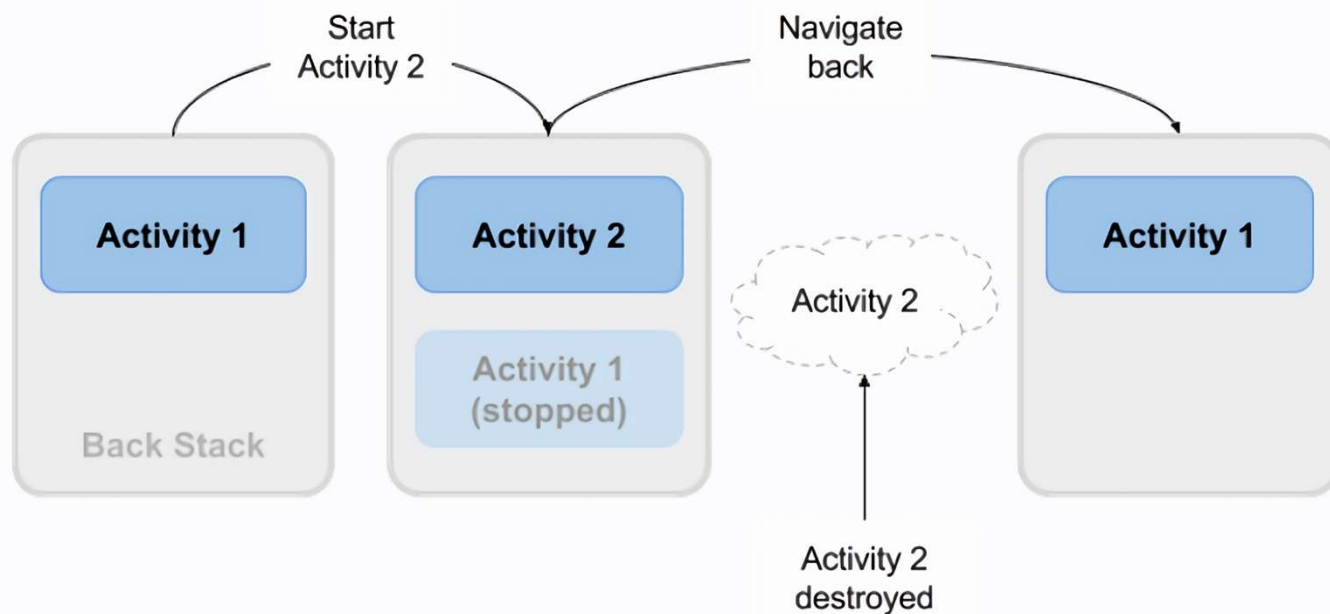
4.2 Activity 回调函数


4.3 Activity 实例状态

4.4 Activity 状态保存和还原

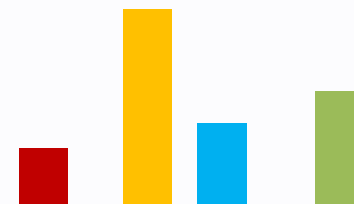
Activity 生命周期的定义

- Activity 从创建到销毁的整个生命周期中的状态集
- Activity 可以处于的所有状态的有向图，以及与从每个状态转换到下一个状态相关的回调





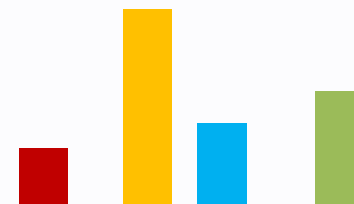
假设你写了一个Activity



Activity 生命周期的作用

良好的生命周期回调实现有助于避免：

- 当用户在使用应用时接听来电，或切换至另一应用时崩溃
- 当用户未使用它时，消耗宝贵的系统资源
- 当用户离开应用并在稍后返回时，丢失用户的进度
- 当屏幕在横向和纵向之间旋转时，崩溃或丢失用户的进度



回调函数

onCreate() //正在创建，静态初始化

onStart() //即将变得可见并具有屏幕焦点

onRestart() //如果Activity停止可通过此函数调用onStart()

onResume() //已可见，开始和用户交互

onPause() //另一个Activity正在前台

onStop() //不再可见，但依旧存在，所有状态已保存

onDestroy() //在安卓系统销毁前的最后一次调用

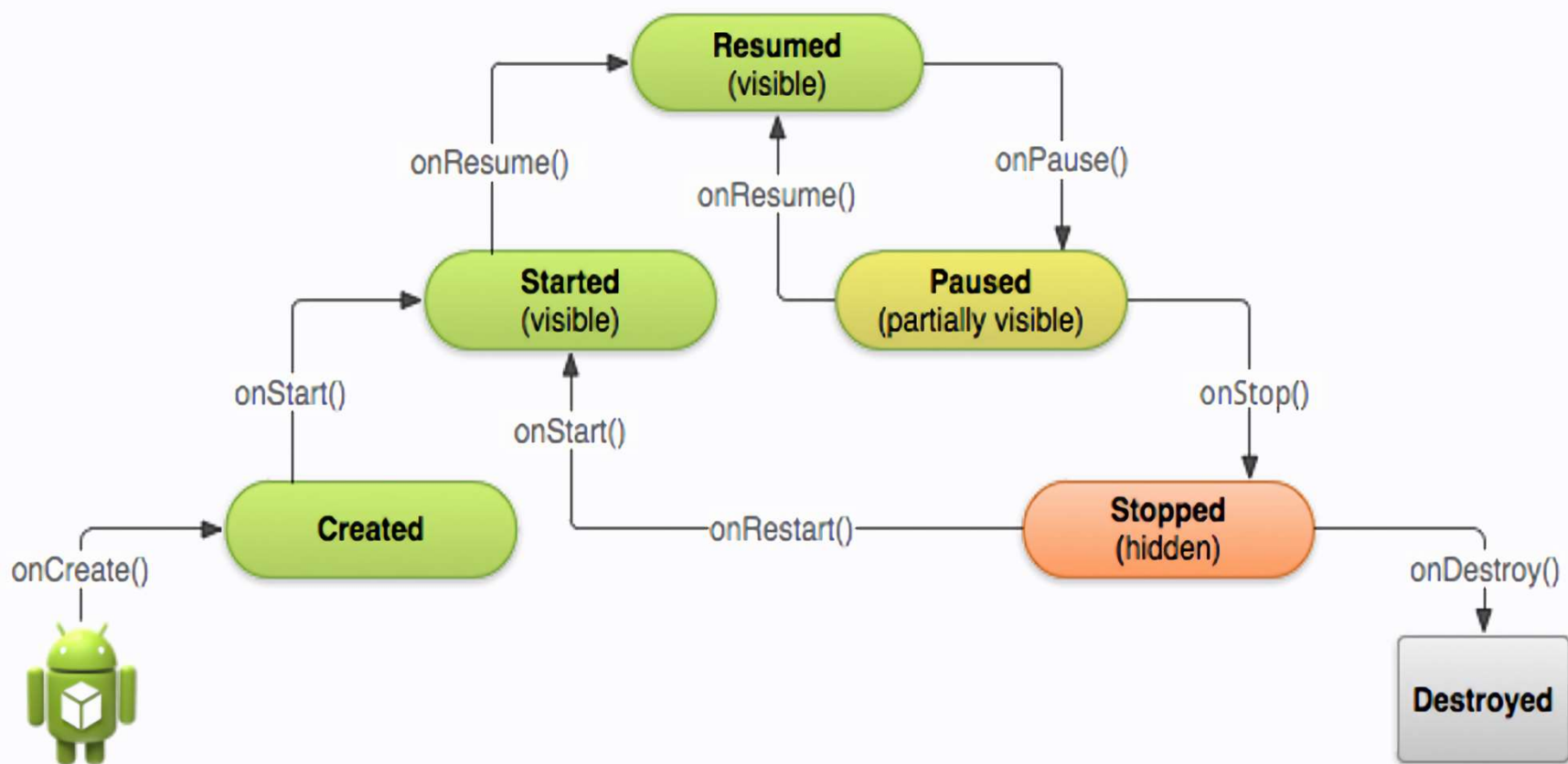


Activity 的三种状态

- 运行
 - 此 Activity 位于屏幕前台并具有用户焦点（运行中）
- 暂停
 - 被取代屏幕前台和用户焦点，但仍可见并处于完全活动状态（保留所有状态和成员信息在内存中并与窗口管理器保持连接）
 - 只有在内存极度不足的情况下可能会被系统终止。
- 停止
 - 被覆盖且位于后台，同样处于活动状态但不可见
 - 在需要内存时会被系统随时终止

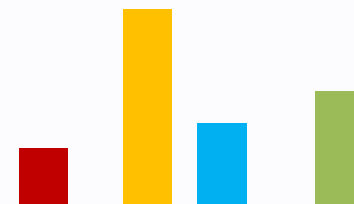


Activity 状态和回调图



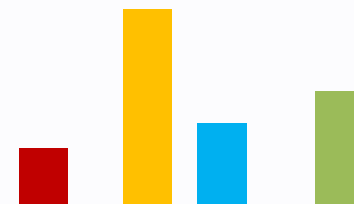
Activity生命周期中的三个嵌套循环

- 整个生命周期
 - onCreate() 调用与 onDestroy() 调用之间
- 可见生命周期
 - onStart() 调用与 onStop() 调用之间
- 前台生命周期
 - onResume() 调用与 onPause() 调用之间



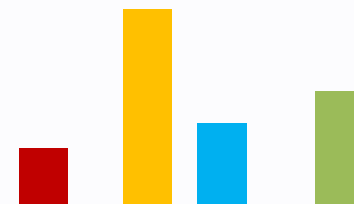
整个生命周期

- 在 onCreate() 中执行全局状态设置（例如定义布局）
- 释放 onDestroy() 中的所有其余资源
- 例如 Activity 有一个在后台运行的线程用于从网络上下载数据，应在 onCreate() 中创建该线程，然后在 onDestroy() 中停止该线程。



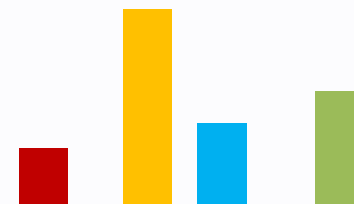
可见生命周期

- 用户可以在屏幕上看到 Activity 并与其交互
- 当一个新 Activity 启动并且此 Activity 不再可见时，系统会调用 `onStop()`。
 - 可以在调用这两个方法之间保留向用户显示 Activity 所需的资源。

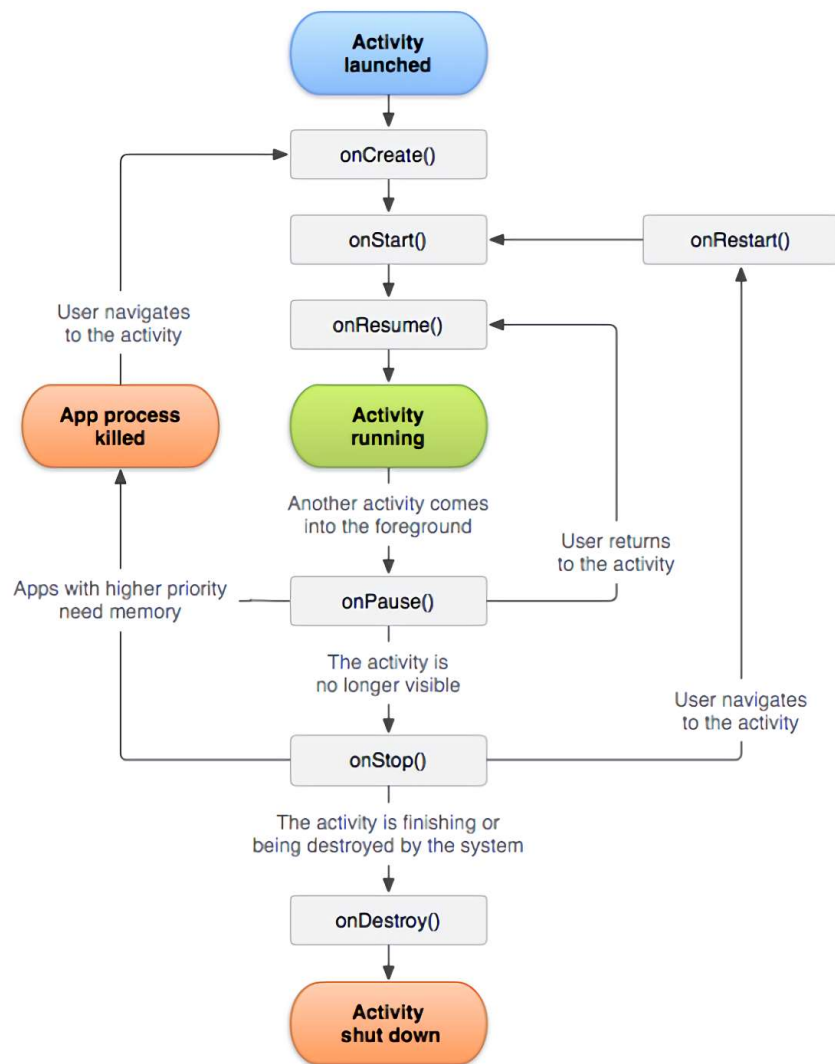


前台生命周期

- Activity 位于屏幕上的所有其他 Activity 之前，并具有用户输入焦点。
- Activity 可频繁转入和转出前台
- 例如，当设备转入休眠状态或出现对话框时，系统会调用 onPause()
- 由于此状态可能经常发生转变，因此这两个方法中应采用适度轻量级的代码，以避免因转变速度慢而让用户等待。

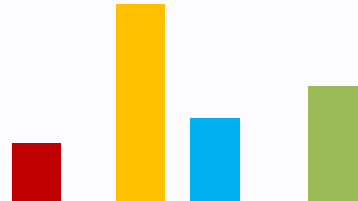


Activity 在状态转变期间可能经过的路径图



onCreate()说明

- 首次创建 Activity 时调用。
- 应该在此方法中执行所有正常的静态设置
 - 创建视图、将数据绑定到列表等等。
- 系统向此方法传递一个 Bundle 对象，其中包含 Activity 的上一状态，不过前提是捕获了该状态
- 始终后接 onStart()



onCreate()操作

- 此示例展示 Activity 的基本设置，例如声明界面（在 XML 布局文件中定义）、定义成员变量，以及配置某些界面。系统通过将文件的资源 ID `R.layout.main_activity` 传递给 `setContentView()` 来指定 XML 布局文件。



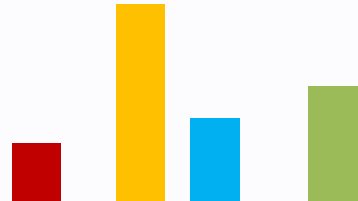
onCreate()示例

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    if (savedInstanceState != null) {  
        // 恢复实例状态  
        gameState = savedInstanceState.getString(GAME_STATE_KEY);  
    }  
    // 状态保存  
    setContentView(R.layout.main_activity);  
    // 设置布局文件  
    textView = (TextView) findViewById(R.id.text_view);  
    // 初始化  
}
```



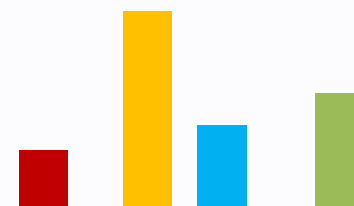
onStart()说明

- 在 Activity 即将对用户可见之前调用，进入前台并支持交互做准备
- 方法会非常快速地完成，并且与onCreate状态一样，系统不会长期处于onStart状态
- 如果 Activity 转入前台，则后接 onResume()
- 如果 Activity 转入隐藏状态，则后接 onStop()



onResume()说明

- 应用与用户交互的状态，处于 Activity 堆栈的顶层，并具有用户输入焦点。
- 会一直保持直到中断事件发生，并调用 onPause()，例如来电，切换应用，关闭屏幕。
- 此时生命周期组件可以启动任何需要在组件可见，且位于前台时运行的功能，例如启动摄像头预览。
- 如果从 onPause 状态返回会被再次调用，此时应初始化在 onPause 期间释放的组件并执行其他操作。



onResume()示例

```
public class CameraComponent implements LifecycleObserver {  
    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)  
    public void initializeCamera() {  
        if (camera == null) {  
            getCamera();  
        } //摄像头仅在应用可见且在前台运行状态时可用  
    } //在暂停状态且可见(例如多窗口模式)时可用应在ON_START后初始化  
} //在收到 ON_RESUME 事件时访问具有生命周期感知能力的摄像头组件
```



onPause()说明

- 系统将此方法视为用户正在离开 Activity 的第一个标志，尽管这并不总是意味着活动正在遭到销毁
- 使用 onPause() 方法暂停或调整不应继续的操作，释放用户不需要但可能影响电池续航时间的任何资源
- 此方法表示 Activity 不再位于前台，但如果用户处于多窗口模式仍然可见，因此多窗口应用应在onStop后释放资源
- 如果 Activity 返回前台，则后接 onResume()
- 如果 Activity 转入对用户不可见状态，则后接 onStop()。



进入onPause()状态的原因

- 最常见：某个事件中断应用执行
- 在 Android 7.0后有多个应用在多窗口模式下运行。但只有一个应用（窗口）拥有焦点，因此系统会暂停所有其他应用。
- 有新的半透明 Activity（例如对话框）处于开启状态。只要仍然部分可见但并未处于焦点之中，它便会一直暂停。



onPause()注意事项

- 执行非常简单，而且不一定要有足够的时间来执行保存操作
- 不应在此保存应用或用户数据、进行网络调用，或执行数据库事务
- 应在 onStop() 期间关闭高负载的操作



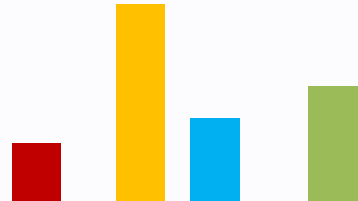
onPause()示例

```
public class JavaCameraComponent implements LifecycleObserver {  
    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)  
    public void releaseCamera() {  
        if (camera != null) {  
            camera.release();  
            camera = null;  
        }  
    }  
}
```



onStop()说明

- Activity 不再对用户可见或系统已结束运行并即将终止
- 应用应释放或调整应用对用户不可见时的无用资源：例如应用可以暂停动画效果，或从细粒度位置更新切换到粗粒度
- 执行 CPU 相对密集的关闭操作：例如将信息保存到数据库，或将草稿笔记内容保存到持久存储中
- 如果 Activity 恢复与用户的交互，则后接 onRestart()
- 如果 Activity 被销毁，则后接 onDestroy()



onStop()示例

```
protected void onStop() {  
    super.onStop();  
    ContentValues values = new ContentValues(); //保存笔记本草稿  
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());  
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());  
    asyncQueryHandler.startUpdate (  
        mToken,null,uri,values,null,null  
        //关联调用的令牌, cookie, note要更新的URI,  
        //列名和新值的映射, SELECT,WHERE条件  
    );  
}
```



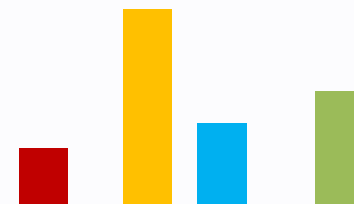
onDestroy()说明

- 在 Activity 被销毁前调用，也是最后的调用
- 由于用户彻底关闭或由于系统为节省空间而调用 finish()，可以通过 isFinishing() 方法区分这两种情形
- 由于配置变更（例如设备旋转或多窗口模式），系统暂时销毁 Activity 并在随后立即新建
- 应释放先前的回调尚未释放的所有资源

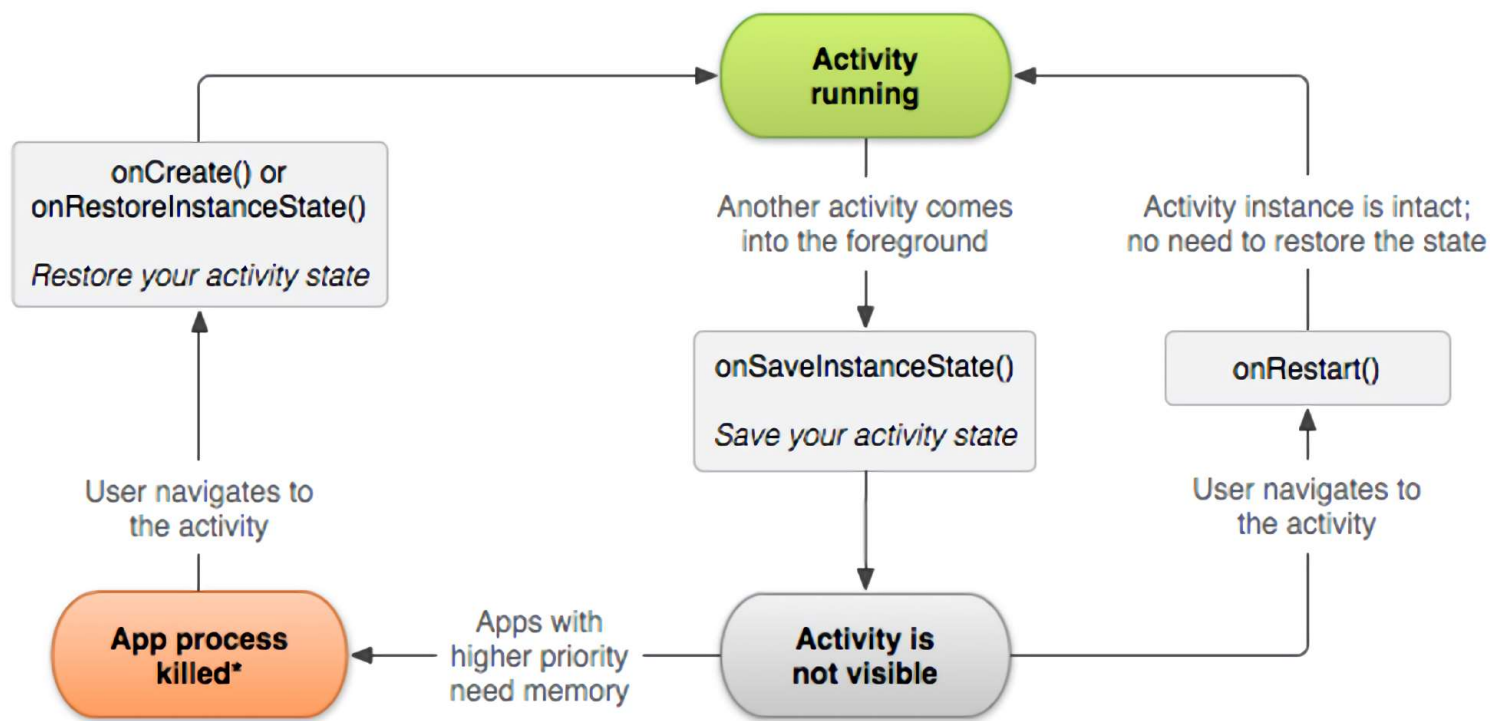


保存 Activity 状态

- 当 Activity 暂停或停止时，对象仍保留在内存中，有关其成员和当前状态的所有信息仍处于活动状态。所做的任何更改都会得到保留，返回前台继续时仍然存在
- 当系统为了恢复内存而销毁某项 Activity 时，对象也会被销毁，因此系统必须在用户返回时重建。但用户并不知道，因此他们认为状态毫无变化
- 为了完成信息保存，可以使用方法 `onSaveInstanceState()`



状态保存流程图



*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved



状态保存具体步骤

- 系统会先调用 `onSaveInstanceState()`
- 系统会向该方法传递一个 `Bundle`，使用 `putString()` 和 `putInt()` 等方法以名称-值对形式保存有关状态的信息
- 系统终止应用进程后会重建该 `Activity`，并将 `Bundle` 同时传递给 `onCreate()` 和 `onRestoreInstanceState()`
- 从 `Bundle` 提取保存的状态并恢复状态。如果没有状态信息需要恢复，则传递给您的 `Bundle` 是空值（比如首次创建）



状态保存示例

```
static final String STATE_SCORE = "playerScore";  
static final String STATE_LEVEL = "playerLevel";
```

```
@Override
```

```
public void onSaveInstanceState(Bundle savedInstanceState) {  
    savedInstanceState.putInt(STATE_SCORE, currentScore);  
    savedInstanceState.putInt(STATE_LEVEL, currentLevel);  
    super.onSaveInstanceState(savedInstanceState);  
}
```



使用保存的实例状态恢复 Activity 界面状态

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState); // 始终首先调用超类  
    //检查是否正在重新创建以前销毁的实例  
    if (savedInstanceState != null) { //恢复成员的值  
        currentScore = savedInstanceState.getInt(STATE_SCORE);  
        currentLevel = savedInstanceState.getInt(STATE_LEVEL);  
    }  
    else { //使用新值初始化 }  
}
```



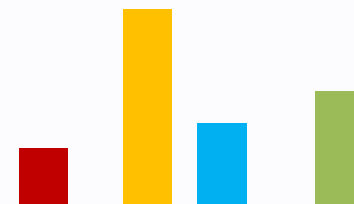
Activity 的回调顺序协调

- 启动第二个 Activity 的过程与停止第一个的过程存在重叠
- 因此生命周期回调的顺序经过明确定义
 - Activity A 的 onPause() 方法执行
 - Activity B 的 onCreate()、onStart() 和 onResume() 方法依次执行
 - 如果 Activity A 在屏幕上不再可见，则其 onStop() 方法执行
- 利用可预测的回调顺序管理信息转变
 - 如果要在第一个停止时向数据库写入数据，以便下一个能够读取该数据，则应在 onPause() 而不是 onStop() 执行期间向数据库写入数据



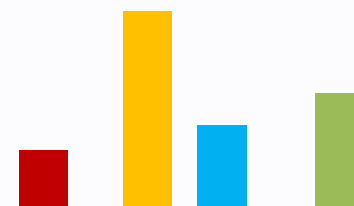
任务和返回栈

- 任务是指在执行特定作业时与用户交互的一系列 Activity。这些 Activity 按照各自的打开顺序排列在堆栈即返回栈中
- 当用户触摸应用图标时，该应用的任务将出现在前台。如果应用最近未曾使用则会创建一个新任务，并且该应用的主 Activity 将作为堆栈中的根 Activity 打开。



Activity 和任务的默认行为

- 每次新 Activity 启动时，前一 Activity 便会停止，但系统会在堆栈（返回栈）中保留该 Activity 的状态。
- 当新 Activity 启动时，系统会将其推送到返回栈上，并取得用户焦点。
- 返回栈遵循基本的后进先出堆栈机制，当用户完成当前 Activity 并按返回按钮时，系统会从堆栈中将其弹出并销毁，然后恢复前一 Activity。
- 即使来自其他任务，Activity 也可以多次实例化。



Back navigation

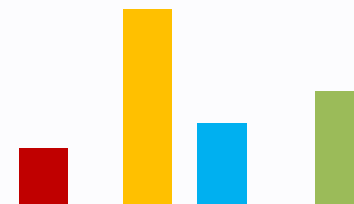
- 后退导航由Android系统的后台堆栈控制
- 每个任务都有自己的后台堆栈，在任务之间切换会激活
- 后台堆栈保留了最近查看的屏幕的历史记录
- 后退堆栈包含用户为当前任务以相反顺序启动的所有Activity实例



Up navigation

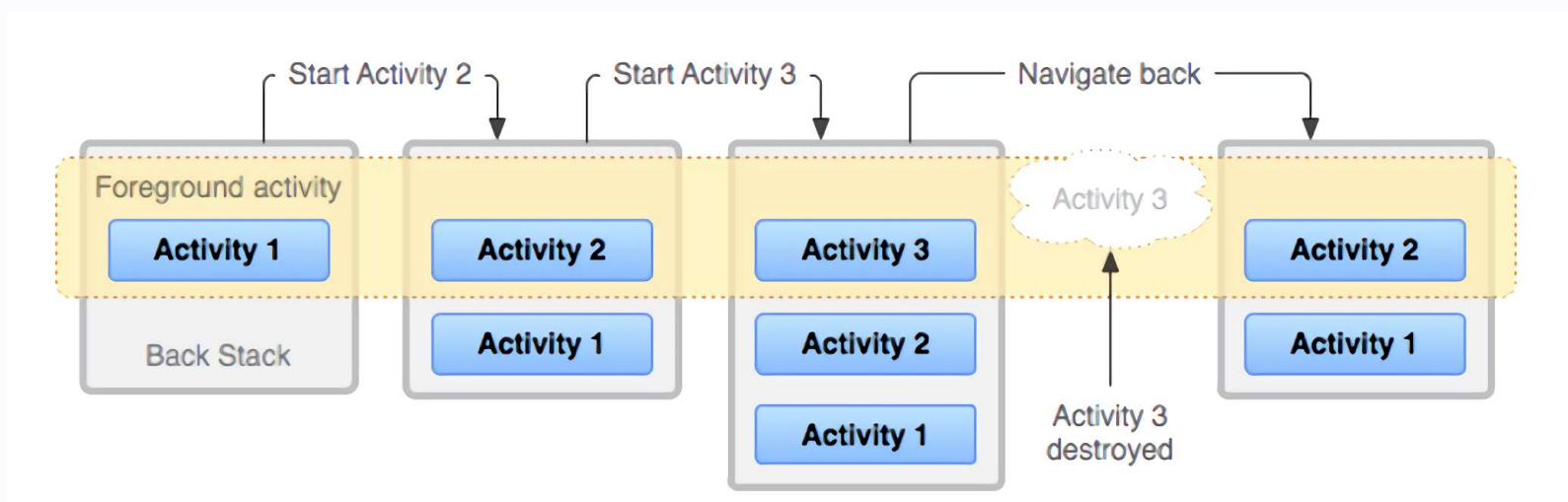
- 前进导航通过定义清单文件中活动之间的父子关系来控制
- 转到当前 Activity 在 Android 清单文件中定义的父级

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



任务和堆栈图示

- 如果用户按返回，堆栈中的相应 Activity 就会弹出，以显示前一个 Activity，直到用户返回主屏幕为止，或者返回任务开始时正在运行的任意 Activity。当所有 Activity 均从堆栈中移除后，任务即不复存在。



总结

- Android App由一系列Activity组成。理解Activity也是理解Android App的重要步骤
- Activity是有生命周期的，记住保存状态
- Intent告诉Android如何在Activity（界面）之间转化

