# Android四大组件：

**Activity、广播接收者、内容提供者、服务**

# 创建第二个Activity

* 创建一个类，继承Activity，可以不实现任何方法，但是显示会一片空白
* 为第二个Activity再创建一个布局文件，然后在第二个Activity中加载这个布局文件

|  |
| --- |
| **public** **class** SecondActivity **extends** Activity{  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_second*);  }  } |

* 在清单文件中配置第二个Activity，配置在application标签中

|  |
| --- |
| <activity  android:name=*"com.example.secondactivity.MainActivity"*  android:label=*"@string/app\_name"*  android:icon="@drawable/ic\_launcher" >  <intent-filter>  <action android:name=*"android.intent.action.MAIN"* />  <category android:name=*"android.intent.category.LAUNCHER"* />  </intent-filter>  </activity>  <activity android:name=*"com.example.secondactivity.SecondActivity"*>  </activity> |

# Activity的跳转

* Activity的跳转需要创建Intent对象，通过设置intent对象的参数指定要跳转Activity
* 通过设置Activity的包名和类名实现跳转，称为显式意图
* 通过指定动作实现跳转，称为隐式意图

## 显式跳转/意图：启动同一个应用中的activity，就用显式

* 跳转至同一项目下的另一个Activity，直接指定该Activity的字节码即可
* 跳转至其他应用中的Activity，需要指定该应用的包名和该Activity的类名
* 打开一个应用，可以在logcat中看到应用的包名和打开的activity

## 隐式跳转/意图：启动其他应用的activity，就用隐式

* 自定义的一个activity如果要隐式跳转，必须在清单文件中，为这个activity配置意图过滤器
* 隐式意图启动Activity，必须匹配activity的其中一个意图过滤器
* intent-filter节点及其子节点都可以同时定义多个，隐式启动时只需与任意一个匹配即可

## 隐式跳转至secondActivity

* 配置意图过滤器

|  |
| --- |
| <activity  android:name=*"com.example.jumpactivity.SecondActivity"*>  <intent-filter>  <action android:name=*"com.example.jc"*/>  <category android:name=*"android.intent.category.DEFAULT"*/>  </intent-filter>    <intent-filter>  <action android:name=*"com.example.jc2"*/>  <category android:name=*"android.intent.category.DEFAULT "*/>  <data  android:scheme=*"jc"*  android:mimeType=*"text/username"*/>  </intent-filter>    <intent-filter>  <action android:name=*"com.example.jc3"*/>  <action android:name=*"com.example.jc4"*/>  <data android:scheme=*"jc2"*/>  <data android:scheme=*"jc3"*/>  <category android:name=*"android.intent.category.DEFAULT"*/>  </intent-filter>  </activity> |

* 设置意图进行跳转
  + 必须与清单文件中设置的其中一个意图过滤器匹配
  + 意图过滤器中设置了几个子节点，那么意图就必须设置几个

|  |
| --- |
| /\*\*  \* 隐式跳转activity  \* **@param** v  \*/  **public** **void** click4(View v) {  Intent intent = **new** Intent();  //匹配第一个意图过滤器  /\*  intent.setAction("com.example.jc"); //action必须与清单文件中配置的一致  // 如果清单文件中配置的category是default，那么可以不写这行代码  // 因为系统会自动添加一个android.intent.category.DEFAULT  intent.addCategory(Intent.CATEGORY\_DEFAULT);  \*/  //匹配第二个意图过滤器  /\*  intent.setAction("com.example.jc2");  // intent.setData(Uri.parse("jc:123")); // 会自动清空type的数据  // intent.setType("text/username"); // 会自动清空data的数据    intent.setDataAndType(Uri.parse("jc:123"), "text/username"); // 同时设置data和type  \*/  // 匹配第三个意图过滤器  intent.setAction("com.example.jc3");  intent.setData(Uri.*parse*("jc2:456"));  startActivity(intent);  } |

## 跳转至其他应用的activity, 隐式跳转

|  |
| --- |
| /\*\*  \* 跳转至其他应用的Activity  \* 跳转至打电话Activity  \* 隐式跳转  \* **@param** v  \*/  **public** **void** click1(View v) {  Intent intent = **new** Intent(); // 创建意图对象  //隐式意图，通过指定aciton和data  intent.setAction(Intent.*ACTION\_CALL*); // 指定打电话的动作  intent.setData(Uri.*parse*("tel:15819072869")); // 指定要拨打的号码  //跳转  startActivity(intent);  } |

## 跳转至本应用的activity，显式跳转

|  |
| --- |
| /\*\*  \* 在本应用中跳转  \* 跳转到secondActivity  \* 显式跳转  \* **@param** v  \*/  **public** **void** click2(View v) {  Intent intent = **new** Intent(); // 创建意图对象  //显式意图，cls：直接指定目标Activity的类名  intent.setClass(**this**, SecondActivity.**class**);  //跳转  startActivity(intent);  } |

## 跳转到其他应用的activity，显式跳转

|  |
| --- |
| /\*\*  \* 跳转至其他应用的Activity  \* 跳转至打电话Activity  \* 显式跳转  \* **@param** v  \*/  **public** **void** click3(View v) {  Intent intent = **new** Intent();  //显式意图，指定目标Activity的包名和类名  intent.setClassName("com.android.browser", "com.android.browser.BrowserActivity");  startActivity(intent);  } |

## 显式意图和隐式意图的应用场景

* 显式意图用于启动同一应用中的Activity
* 隐式意图用于启动不同应用中的Activity
* 启动效率上，隐式远远低于显式
* 如果系统中存在多个Activity的intent-filter同时与你的intent匹配，那么系统会显示一个对话框，列出所有匹配的Activity，由用户选择启动哪一个
* 所有的浏览器都是这个intetn-filter: 这是一个规范。拨号等应用，也都有一个规范

|  |
| --- |
| <intent-filter>  <action android:name=*"android.intent.action.VIEW"*/>  <data android:scheme=*"jc3"*/>  <category android:name=*"android.intent.category.DEFAULT"*/>  </intent-filter> |

## 隐式启动浏览器

|  |
| --- |
| /\*\*  \* 隐式跳转至浏览器  \* **@param** v  \*/  **public** **void** click5(View v) {  Intent intent = **new** Intent();  intent.setAction(Intent.*ACTION\_VIEW*);  intent.setData(Uri.*parse*("http://www.baidu.com"));  startActivity(intent);  } |

## Activity跳转时的数据传递

1. 隐式跳转传数据：

Activity通过Intent启动时，可以通过Intent对象携带数据到目标Activity

|  |
| --- |
| Intent intent = **new** Intent();  intent.setAction("com.example.jc3");  intent.setData(Uri.*parse*("jc2:456"));  startActivity(intent); |

在目标Activity中取出数据

|  |
| --- |
| // 获取启动此Activity的intent对象  Intent intent = getIntent();  Uri uri = intent.getData();  System.*out*.println(uri.toString()); // jc2:456 |

1. 显式跳转传数据：

第一种：

* Activity通过Intent启动时，可以通过Intent对象携带数据到目标Activity

|  |
| --- |
| Intent intent = **new** Intent(**this**, SecondActivity.**class**);  Bundle bundle = **new** Bundle();  bundle.putString("name", "王八");  bundle.putString("name2", "王二");  intent.putExtras(bundle);  startActivity(intent); |

* 在目标Activity中取出数据

|  |
| --- |
| Intent intent = getIntent();  Bundle bundle = intent.getExtras();  String name = (String)bundle.get("name");  String name2 = bundle.getString("name2");  TextView tv = (TextView) findViewById(R.id.*tv*);  tv.setText(name + ":" + name2); |

第二种：

* Activity通过Intent启动时，可以通过Intent对象携带数据到目标Activity

|  |
| --- |
| Intent intent = new Intent(this, SecondActivity.class);  intent.putExtra("maleName", maleName);  intent.putExtra("femaleName", femaleName);  startActivity(intent); |

* 在目标Activity中取出数据

|  |
| --- |
| Intent intent = getIntent();  String maleName = intent.getStringExtra("maleName");  String femaleName = intent.getStringExtra("femaleName"); |

## 在非上下文对象中调用startActivity(Intent intent)方法时

* + 如果是在非上下文对象（即不是activity中）调用startActivity()方法，是不能启动activity的。
  + 这时就需要为intent设置一个flag才能启动，这个flag是：Intent.*FLAG\_ACTIVITY\_NEW\_TASK，*意思就是：

创建一个新的任务栈，在这个任务栈中启动activity

|  |
| --- |
| intent.setFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*); |

* 而在activity中startActivity()就不用，因为，所在的activity本就是在任务栈中的，所以不用设置。如果在一个没有任务栈的地方启动activity，就需要设置flag。例如：广播接收者中，广播接收者要运行是不需要依赖activity的。广播接收者在下一天的课程中

# Activity生命周期

## 生命周期方法：

* + void onCreate()：Activity已经被创建完毕
  + void onStart()：Activity已经显示在屏幕，但没有得到焦点
  + void onResume()：Activity得到焦点，可以与用户交互
  + void onPause()：Activity失去焦点，无法再与用户交互，但依然可见
  + void onStop()：Activity不可见，进入后台
  + void onDestroy()：Activity被销毁
  + void onRestart()：Activity从不可见变成可见时会执行此方法
* 使用场景：Activity创建时需要初始化资源，销毁时需要释放资源；或者播放器应用，在界面进入后台时需要自动暂停

## 调用过程

* 完整生命周期（entire lifetime）：onCreate-->onStart-->onResume-->onPause-->onStop-->onDestory
* 一个activity的情况：
  + 启动一个应用：onCreate-->onStart-->onResume
  + 按返回键：onPause-->onStop-->onDestory
  + 按home键：onPause-->onStop
  + 按home键后在打开应用：onRestart🡪onStart-->onResume
* 两个activity的情况：
  + 在一个activity中跳转到另二activity：主onPause🡪二onCreate🡪二onStart🡪二onResume🡪主onStop
  + 在第二个activity中按返回键：二onPause🡪主onRestart🡪主onStart🡪主onResume🡪二onStop🡪二onDestroy
  + 在第二个activity中按home键：二onPause🡪二onStop
  + 第二个home键之后，在打开应用：二onRestart🡪二onStart🡪二onResume

## 销毁

* 内存不足时，系统会优先杀死后台Activity所在的进程，都杀光了，如果内存还是不足，那么就会杀死暂停状态的Activity所在的进程，如果还是不够，有可能杀死前台进程
* 如果有多个后台进程，在选择杀死的目标时，采用最近最少使用算法（LRU）
* 销毁后的activity再次启动时：onCreate-->onStart-->onResume

## Activity失去焦点，无法再与用户交互，但依然可见

* 这个状态需要两个activity，把第二个activity设置为透明，在清单文件中设置

|  |
| --- |
| <activity  android:name=*"com.example.activityvita.SecondActivity"*  android:theme=*"@android:style/Theme.Translucent"*></activity> |

* 在第一个activity跳转到第二个activity的时候：主onPause🡪二onCreatee🡪二onStart二onResume
* 这时候，第一个activity就已经不能进行操作了，但是还能看见。只有第二个activity才能操作

## 返回键调用的方法：onBackPressed()

* 里面是调用Activity#finish()方法，销毁activity
* 如果要屏蔽返回键，使返回键无效，注释掉super.onBackPressed();即可

# Activity的四种启动模式

* 清单文件中设置启动模式

|  |
| --- |
| <activity  android:name=*"com.example3\_Display\_to\_screen.MainActivity"*  android:label=*"@string/app\_name"*  android:launchMode=*"standard"* >  <intent-filter>  <action android:name=*"android.intent.action.MAIN"* />  <category android:name=*"android.intent.category.LAUNCHER"* />  </intent-filter>  </activity> |

## 任务栈：每个应用会有一个Activity任务栈，存放已启动的Activity

* + 后进先出，最先进栈，就会最后出栈
  + 栈顶：当前屏幕显示的activity

|  |
| --- |
|  |

## standard 标准启动模式：默认就是这个模式

## singleTop 单一顶部模式

* 这个单一是相对于一个应用中的
  + 如果任务栈的栈顶是这个要开启的activity，不会重新的创建这个要开启的activity，而是复用已经存在的activity。保证栈顶如果存在，不会重复创建。
  + 如果任务栈的栈顶不是这个要开启的activity，就会重建创建这个要开启的activity，并显示在栈顶
  + 应用场景：浏览器的书签，防止同一个书签被多次打开

## singeTask 单一任务栈，在当前任务栈里面只能有一个实例存在

* 这个单一是相对于一个应用中的
* 当开启activity的时候，就去检查在任务栈里面是否有实例已经存在，如果有实例存在就复用这个已经存在的activity，并且把这个activity上面的所有的别的activity都清空，复用这个已经存在的activity。保证整个任务栈里面只有一个实例存在
* 应用场景：浏览器的activit
* 如果一个activity的创建需要占用大量的系统资源（cpu，内存）一般配置这个activity为singletask的启动模式。webkit内核 c代码

## singleInstance：单例模式

* 该activity启动时，会在一个新的任务栈中创建，并且只会创建一次，以后再次启动该activity是，其实只是把该activity所在的任务栈移动到前台。上一个任务栈就变成后台
* singleInstance的作用：保证整个系统的内存中只有一个该Activity的实例
* 应用场景： 电话拨打界面

# 横竖屏切换时Activity的生命周期

* 横竖屏切换的时候：onPause-->onStop-->onDestory-->onCreate-->onStart-->onResume
* 就是横竖屏切换的时候，activity会被销毁然后重新创建
* 横竖屏切换的时候，会调用生命周期方法
* 如果不想切换的时候调用生命周期方法，就在清单文件中，对activity进行配置

|  |
| --- |
| <activity  android:configChanges=*"orientation|keyboardHidden|screenSize"*  android:name=*"com.example.launchmode.SecondActivity"*>    </activity> |

* 进行以上设置之后，方向改变、键盘隐藏、屏幕大小改变，都不会调用生命周期方法了

# 写死横竖屏

* 第一种：清单文件中配置

|  |
| --- |
| <activity  android:screenOrientation=*"portrait"*  android:name=*"com.example.launchmode.SecondActivity"*>  </activity> |
| <activity  android:screenOrientation=*"landscape"*  android:name=*"com.example.launchmode.SecondActivity"*>  </activity> |

* 第二种，代码写死

|  |
| --- |
| **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  setRequestedOrientation(ActivityInfo.*SCREEN\_ORIENTATION\_LANDSCAPE*);  } |
| @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  setRequestedOrientation(ActivityInfo.*SCREEN\_ORIENTATION\_PORTRAIT*);  } |

* 如果两种方法都写了，那么，谁后执行谁生效。在这里，清单文件先执行，所以第二种方法生效

# 销毁Activity时传递数据

* 如果说要启动的那个activity销毁时会返回数据，就用这个api启动。
  + Activity#startActivityForResult(intent, requestCode)：告知系统，这个activity销毁时会返回数据。在销毁时，系统会回调Activity#onActivityResult()方法
  + 传入意图，传入请求码（自己定义）
* Activity销毁时返回数据也是通过意图，把数据封装到意图对象中，通过Activity#setResult(resultCode, intent)设置返回的数据
  + Activity#setResult(resultCode, intent)：销毁时会把数据传递到启动本activity的那个activity的：activity#onActivityResult方法中，启动时的请求码也会一起返回
  + ResultCode：结果码，用来判断activity返回的是那种数据
* onActivityResult(int requestCode, int resultCode, Intent data): 有activity销毁时返回数据就会会调用
  + requestCode：请求码，使用startActivityForResult启动activity时设置的。Activity销毁时会返回来，用来判断是哪个activity销毁
  + resultCode：结果码，用来判断activity返回的是那种数据
  + data：activity销毁时返回的装有数据的意图对象
* 给listview设置条目的点击侦听
  + listView#setOnItemClickListener(new OnItemClickListener() {})
* 销毁activity：Activity#finish()

案例：发送短信的选择联系人，快捷回复